
rocThrust Documentation

Release 0.8

Advanced Mirco Devices

Jul 22, 2019

CONTENTS:

1	Library API	1
1.1	Class Hierarchy	1
1.2	File Hierarchy	1
1.3	Full API	1
1.3.1	Namespaces	1
1.3.2	Classes and Structs	20
1.3.3	Functions	72
1.3.4	Variables	365
1.3.5	Defines	369
1.3.6	Typedefs	371
2	Indices and tables	373
	Index	375

LIBRARY API

1.1 Class Hierarchy

1.2 File Hierarchy

1.3 Full API

1.3.1 Namespaces

Namespace placeholders

Facilities for constructing simple functions inline.

Contents

- *Detailed Description*

Detailed Description

Objects in the `thrust::placeholders` namespace may be used to create simple arithmetic functions inline in an algorithm invocation. Combining placeholders such as `_1` and `_2` with arithmetic operations such as `+` creates an unnamed function object which applies the operation to their arguments. The type of placeholder objects is implementation-defined. The following code snippet demonstrates how to use the placeholders `_1` and `_2` with `thrust::transform` to implement the SAXPY computation: `#include<thrust/device_vector.h> #include<thrust/transform.h> #include<thrust/functional.h>`

```
int main() { thrust::device_vector<float> x(4), y(4); x[0]=1; x[1]=2; x[2]=3; x[3]=4;
y[0]=1; y[1]=1; y[2]=1; y[3]=1;
float a=2.0f;
using namespace thrust::placeholders;
thrust::transform(x.begin(), x.end(), y.begin(), y.begin(), a*_1+_2);
// y is now {3, 5, 7, 9} }
```

Namespace thrust

`thrust` is the top-level namespace which contains all Thrust functions and types.

Contents

- *Namespaces*
- *Classes*
- *Functions*
- *Variables*

Namespaces

- *Namespace `thrust::placeholders`*
- *Namespace `thrust::random`*
- *Namespace `thrust::system`*

Classes

- *Template Struct `binary_function`*
- *Template Struct `binary_negate`*
- *Template Struct `binary_traits`*
- *Template Struct `bit_and`*
- *Template Struct `bit_or`*
- *Template Struct `bit_xor`*
- *Template Struct `complex`*
- *Template Struct `device_allocator::rebind`*
- *Template Struct `device_allocator< void >::device_allocator< void >`*
- *Template Struct `device_execution_policy`*
- *Template Struct `device_malloc_allocator::rebind`*
- *Template Struct `device_new_allocator::rebind`*
- *Template Struct `divides`*
- *Template Struct `equal_to`*
- *Template Struct `greater`*
- *Template Struct `greater_equal`*
- *Template Struct `host_execution_policy`*
- *Template Struct `identity`*
- *Template Struct `less`*
- *Template Struct `less_equal`*

- *Template Struct logical_and*
- *Template Struct logical_not*
- *Template Struct logical_or*
- *Template Struct maximum*
- *Template Struct minimum*
- *Template Struct minus*
- *Template Struct modulus*
- *Template Struct multiplies*
- *Template Struct negate*
- *Template Struct not_equal_to*
- *Template Struct pair*
- *Template Struct plus*
- *Template Struct project1st*
- *Template Struct project2nd*
- *Template Struct tuple_element*
- *Template Struct tuple_size*
- *Template Struct unary_function*
- *Template Struct unary_negate*
- *Template Struct unary_traits*
- *Template Class device_allocator*
- *Template Class device_allocator< void >*
- *Template Class device_malloc_allocator*
- *Template Class device_new_allocator*
- *Template Class device_ptr*
- *Template Class device_reference*
- *Template Class device_vector*
- *Template Class host_vector*
- *Template Class tuple*

Functions

- *Template Function thrust::abs*
- *Template Function thrust::acos*
- *Template Function thrust::acosh*
- *Template Function thrust::adjacent_difference(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator)*

- Template Function `thrust::adjacent_difference(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, BinaryFunction)`
- Template Function `thrust::adjacent_difference(InputIterator, InputIterator, OutputIterator)`
- Template Function `thrust::adjacent_difference(InputIterator, InputIterator, OutputIterator, BinaryFunction)`
- Template Function `thrust::advance`
- Template Function `thrust::all_of(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`
- Template Function `thrust::all_of(InputIterator, InputIterator, Predicate)`
- Template Function `thrust::any_of(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`
- Template Function `thrust::any_of(InputIterator, InputIterator, Predicate)`
- Template Function `thrust::arg`
- Template Function `thrust::asin`
- Template Function `thrust::asinh`
- Template Function `thrust::atan`
- Template Function `thrust::atanh`
- Template Function `thrust::binary_search(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const LessThanComparable&)`
- Template Function `thrust::binary_search(ForwardIterator, ForwardIterator, const LessThanComparable&)`
- Template Function `thrust::binary_search(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`
- Template Function `thrust::binary_search(ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`
- Template Function `thrust::binary_search(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)`
- Template Function `thrust::binary_search(ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)`
- Template Function `thrust::binary_search(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)`
- Template Function `thrust::binary_search(ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)`
- Template Function `thrust::conj`
- Template Function `thrust::copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator)`
- Template Function `thrust::copy(InputIterator, InputIterator, OutputIterator)`
- Template Function `thrust::copy_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, Predicate)`
- Template Function `thrust::copy_if(InputIterator, InputIterator, OutputIterator, Predicate)`
- Template Function `thrust::copy_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate)`
- Template Function `thrust::copy_if(InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate)`

- *Template Function `thrust::copy_n(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, Size, OutputIterator)`*
- *Template Function `thrust::copy_n(InputIterator, Size, OutputIterator)`*
- *Template Function `thrust::cos`*
- *Template Function `thrust::cosh`*
- *Template Function `thrust::count(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, const EqualityComparable&)`*
- *Template Function `thrust::count(InputIterator, InputIterator, const EqualityComparable&)`*
- *Template Function `thrust::count_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`*
- *Template Function `thrust::count_if(InputIterator, InputIterator, Predicate)`*
- *Template Function `thrust::device_delete`*
- *Function `thrust::device_free`*
- *Function `thrust::device_malloc`*
- *Template Function `thrust::device_new(device_ptr<void>, const size_t)`*
- *Template Function `thrust::device_new(device_ptr<void>, const T&, const size_t)`*
- *Template Function `thrust::device_new(const size_t)`*
- *Template Function `thrust::device_pointer_cast(T*)`*
- *Template Function `thrust::device_pointer_cast(const device_ptr<T>&)`*
- *Template Function `thrust::distance`*
- *Template Function `thrust::equal(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2)`*
- *Template Function `thrust::equal(InputIterator1, InputIterator1, InputIterator2)`*
- *Template Function `thrust::equal(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, BinaryPredicate)`*
- *Template Function `thrust::equal(InputIterator1, InputIterator1, InputIterator2, BinaryPredicate)`*
- *Template Function `thrust::equal_range(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const LessThanComparable&)`*
- *Template Function `thrust::equal_range(ForwardIterator, ForwardIterator, const LessThanComparable&)`*
- *Template Function `thrust::equal_range(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`*
- *Template Function `thrust::equal_range(ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`*
- *Template Function `thrust::exclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator)`*
- *Template Function `thrust::exclusive_scan(InputIterator, InputIterator, OutputIterator)`*
- *Template Function `thrust::exclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, T)`*
- *Template Function `thrust::exclusive_scan(InputIterator, InputIterator, OutputIterator, T)`*

- Template Function `thrust::exclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, T, AssociativeOperator)`
- Template Function `thrust::exclusive_scan(InputIterator, InputIterator, OutputIterator, T, AssociativeOperator)`
- Template Function `thrust::exclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator)`
- Template Function `thrust::exclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator)`
- Template Function `thrust::exclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, T)`
- Template Function `thrust::exclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator, T)`
- Template Function `thrust::exclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, T, BinaryPredicate)`
- Template Function `thrust::exclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator, T, BinaryPredicate)`
- Template Function `thrust::exclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, T, BinaryPredicate, AssociativeOperator)`
- Template Function `thrust::exclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator, T, BinaryPredicate, AssociativeOperator)`
- Template Function `thrust::exp`
- Template Function `thrust::fill(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&)`
- Template Function `thrust::fill(ForwardIterator, ForwardIterator, const T&)`
- Template Function `thrust::fill_n(const thrust::detail::execution_policy_base<DerivedPolicy>&, OutputIterator, Size, const T&)`
- Template Function `thrust::fill_n(OutputIterator, Size, const T&)`
- Template Function `thrust::find(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, const T&)`
- Template Function `thrust::find(InputIterator, InputIterator, const T&)`
- Template Function `thrust::find_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`
- Template Function `thrust::find_if(InputIterator, InputIterator, Predicate)`
- Template Function `thrust::find_if_not(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`
- Template Function `thrust::find_if_not(InputIterator, InputIterator, Predicate)`
- Template Function `thrust::for_each(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, UnaryFunction)`
- Template Function `thrust::for_each(InputIterator, InputIterator, UnaryFunction)`
- Template Function `thrust::for_each_n(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, Size, UnaryFunction)`
- Template Function `thrust::for_each_n(InputIterator, Size, UnaryFunction)`

- Template Function `thrust::free`
- Template Function `thrust::gather(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, RandomAccessIterator, OutputIterator)`
- Template Function `thrust::gather(InputIterator, InputIterator, RandomAccessIterator, OutputIterator)`
- Template Function `thrust::gather_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator, OutputIterator)`
- Template Function `thrust::gather_if(InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator, OutputIterator)`
- Template Function `thrust::gather_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator, OutputIterator, Predicate)`
- Template Function `thrust::gather_if(InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator, OutputIterator, Predicate)`
- Template Function `thrust::generate(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Generator)`
- Template Function `thrust::generate(ForwardIterator, ForwardIterator, Generator)`
- Template Function `thrust::generate_n(const thrust::detail::execution_policy_base<DerivedPolicy>&, OutputIterator, Size, Generator)`
- Template Function `thrust::generate_n(OutputIterator, Size, Generator)`
- Template Function `thrust::get(detail::cons<HT, TT>&)`
- Template Function `thrust::get(const detail::cons<HT, TT>&)`
- Template Function `thrust::get_temporary_buffer`
- Template Function `thrust::inclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator)`
- Template Function `thrust::inclusive_scan(InputIterator, InputIterator, OutputIterator)`
- Template Function `thrust::inclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, AssociativeOperator)`
- Template Function `thrust::inclusive_scan(InputIterator, InputIterator, OutputIterator, AssociativeOperator)`
- Template Function `thrust::inclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator)`
- Template Function `thrust::inclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator)`
- Template Function `thrust::inclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryPredicate)`
- Template Function `thrust::inclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryPredicate)`
- Template Function `thrust::inclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryPredicate, AssociativeOperator)`
- Template Function `thrust::inclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryPredicate, AssociativeOperator)`
- Template Function `thrust::inner_product(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputType)`

- Template Function `thrust::inner_product(InputIterator1, InputIterator1, InputIterator2, OutputType)`
- Template Function `thrust::inner_product(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputType, BinaryFunction1, BinaryFunction2)`
- Template Function `thrust::inner_product(InputIterator1, InputIterator1, InputIterator2, OutputType, BinaryFunction1, BinaryFunction2)`
- Template Function `thrust::is_partitioned(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`
- Template Function `thrust::is_partitioned(InputIterator, InputIterator, Predicate)`
- Template Function `thrust::is_sorted(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`
- Template Function `thrust::is_sorted(ForwardIterator, ForwardIterator)`
- Template Function `thrust::is_sorted(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Compare)`
- Template Function `thrust::is_sorted(ForwardIterator, ForwardIterator, Compare)`
- Template Function `thrust::is_sorted_until(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`
- Template Function `thrust::is_sorted_until(ForwardIterator, ForwardIterator)`
- Template Function `thrust::is_sorted_until(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Compare)`
- Template Function `thrust::is_sorted_until(ForwardIterator, ForwardIterator, Compare)`
- Template Function `thrust::log`
- Template Function `thrust::log10`
- Template Function `thrust::lower_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const LessThanComparable&)`
- Template Function `thrust::lower_bound(ForwardIterator, ForwardIterator, const LessThanComparable&)`
- Template Function `thrust::lower_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`
- Template Function `thrust::lower_bound(ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`
- Template Function `thrust::lower_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)`
- Template Function `thrust::lower_bound(ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)`
- Template Function `thrust::lower_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)`
- Template Function `thrust::lower_bound(ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)`
- Template Function `thrust::make_pair`
- Template Function `thrust::make_tuple(const T0&)`
- Template Function `thrust::make_tuple(const T0&, const T1&)`
- Template Function `thrust::malloc(const thrust::detail::execution_policy_base<DerivedPolicy>&, std::size_t)`

- Template Function `thrust::malloc(const thrust::detail::execution_policy_base<DerivedPolicy>&, std::size_t)`
- Template Function `thrust::max_element(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`
- Template Function `thrust::max_element(ForwardIterator, ForwardIterator)`
- Template Function `thrust::max_element(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, BinaryPredicate)`
- Template Function `thrust::max_element(ForwardIterator, ForwardIterator, BinaryPredicate)`
- Template Function `thrust::merge(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`
- Template Function `thrust::merge(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`
- Template Function `thrust::merge(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`
- Template Function `thrust::merge(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`
- Template Function `thrust::merge_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)`
- Template Function `thrust::merge_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)`
- Template Function `thrust::merge_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, Compare)`
- Template Function `thrust::merge_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)`
- Template Function `thrust::min_element(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`
- Template Function `thrust::min_element(ForwardIterator, ForwardIterator)`
- Template Function `thrust::min_element(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, BinaryPredicate)`
- Template Function `thrust::min_element(ForwardIterator, ForwardIterator, BinaryPredicate)`
- Template Function `thrust::minmax_element(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`
- Template Function `thrust::minmax_element(ForwardIterator, ForwardIterator)`
- Template Function `thrust::minmax_element(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, BinaryPredicate)`
- Template Function `thrust::minmax_element(ForwardIterator, ForwardIterator, BinaryPredicate)`
- Template Function `thrust::mismatch(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2)`
- Template Function `thrust::mismatch(InputIterator1, InputIterator1, InputIterator2)`
- Template Function `thrust::mismatch(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, BinaryPredicate)`
- Template Function `thrust::mismatch(InputIterator1, InputIterator1, InputIterator2, BinaryPredicate)`

- Template Function `thrust::none_of(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`
- Template Function `thrust::none_of(InputIterator, InputIterator, Predicate)`
- Template Function `thrust::norm`
- Template Function `thrust::not1`
- Template Function `thrust::not2`
- Template Function `thrust::operator!=(const complex<T0>&, const complex<T1>&)`
- Template Function `thrust::operator!=(const complex<T0>&, const std::complex<T1>&)`
- Template Function `thrust::operator!=(const std::complex<T0>&, const complex<T1>&)`
- Template Function `thrust::operator!=(const T0&, const complex<T1>&)`
- Template Function `thrust::operator!=(const complex<T0>&, const T1&)`
- Template Function `thrust::operator!=(const pair<T1, T2>&, const pair<T1, T2>&)`
- Template Function `thrust::operator*(const complex<T0>&, const complex<T1>&)`
- Template Function `thrust::operator*(const complex<T0>&, const T1&)`
- Template Function `thrust::operator*(const T0&, const complex<T1>&)`
- Template Function `thrust::operator+(const complex<T>&)`
- Template Function `thrust::operator+(const complex<T0>&, const complex<T1>&)`
- Template Function `thrust::operator+(const complex<T0>&, const T1&)`
- Template Function `thrust::operator+(const T0&, const complex<T1>&)`
- Template Function `thrust::operator-(const complex<T0>&, const T1&)`
- Template Function `thrust::operator-(const T0&, const complex<T1>&)`
- Template Function `thrust::operator-(const complex<T>&)`
- Template Function `thrust::operator-(const complex<T0>&, const complex<T1>&)`
- Template Function `thrust::operator/(const complex<T0>&, const complex<T1>&)`
- Template Function `thrust::operator/(const complex<T0>&, const T1&)`
- Template Function `thrust::operator/(const T0&, const complex<T1>&)`
- Template Function `thrust::operator<`
- Template Function `thrust::operator<<`
- Template Function `thrust::operator<=`
- Template Function `thrust::operator==(const complex<T0>&, const complex<T1>&)`
- Template Function `thrust::operator==(const complex<T0>&, const std::complex<T1>&)`
- Template Function `thrust::operator==(const std::complex<T0>&, const complex<T1>&)`
- Template Function `thrust::operator==(const T0&, const complex<T1>&)`
- Template Function `thrust::operator==(const complex<T0>&, const T1&)`
- Template Function `thrust::operator==(const pair<T1, T2>&, const pair<T1, T2>&)`
- Template Function `thrust::operator>`

- Template Function `thrust::operator>=`
- Template Function `thrust::operator>>`
- Template Function `thrust::partition(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate)`
- Template Function `thrust::partition(ForwardIterator, ForwardIterator, Predicate)`
- Template Function `thrust::partition(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, Predicate)`
- Template Function `thrust::partition(ForwardIterator, ForwardIterator, InputIterator, Predicate)`
- Template Function `thrust::partition_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)`
- Template Function `thrust::partition_copy(InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)`
- Template Function `thrust::partition_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, Predicate)`
- Template Function `thrust::partition_copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, Predicate)`
- Template Function `thrust::partition_point(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate)`
- Template Function `thrust::partition_point(ForwardIterator, ForwardIterator, Predicate)`
- Template Function `thrust::polar`
- Template Function `thrust::pow(const complex<T0>&, const complex<T1>&)`
- Template Function `thrust::pow(const complex<T0>&, const T1&)`
- Template Function `thrust::pow(const T0&, const complex<T1>&)`
- Template Function `thrust::proj`
- Template Function `thrust::raw_pointer_cast`
- Template Function `thrust::raw_reference_cast(T&)`
- Template Function `thrust::raw_reference_cast(const T&)`
- Template Function `thrust::reduce(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator)`
- Template Function `thrust::reduce(InputIterator, InputIterator)`
- Template Function `thrust::reduce(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, T)`
- Template Function `thrust::reduce(InputIterator, InputIterator, T)`
- Template Function `thrust::reduce(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, T, BinaryFunction)`
- Template Function `thrust::reduce(InputIterator, InputIterator, T, BinaryFunction)`
- Template Function `thrust::reduce_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)`
- Template Function `thrust::reduce_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)`

- Template Function `thrust::reduce_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate)`
- Template Function `thrust::reduce_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate)`
- Template Function `thrust::reduce_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate, BinaryFunction)`
- Template Function `thrust::reduce_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate, BinaryFunction)`
- Template Function `thrust::remove(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&)`
- Template Function `thrust::remove(ForwardIterator, ForwardIterator, const T&)`
- Template Function `thrust::remove_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, const T&)`
- Template Function `thrust::remove_copy(InputIterator, InputIterator, OutputIterator, const T&)`
- Template Function `thrust::remove_copy_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, Predicate)`
- Template Function `thrust::remove_copy_if(InputIterator, InputIterator, OutputIterator, Predicate)`
- Template Function `thrust::remove_copy_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate)`
- Template Function `thrust::remove_copy_if(InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate)`
- Template Function `thrust::remove_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate)`
- Template Function `thrust::remove_if(ForwardIterator, ForwardIterator, Predicate)`
- Template Function `thrust::remove_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, Predicate)`
- Template Function `thrust::remove_if(ForwardIterator, ForwardIterator, InputIterator, Predicate)`
- Template Function `thrust::replace(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, const T&)`
- Template Function `thrust::replace(ForwardIterator, ForwardIterator, const T&, const T&)`
- Template Function `thrust::replace_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, const T&, const T&)`
- Template Function `thrust::replace_copy(InputIterator, InputIterator, OutputIterator, const T&, const T&)`
- Template Function `thrust::replace_copy_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, Predicate, const T&)`
- Template Function `thrust::replace_copy_if(InputIterator, InputIterator, OutputIterator, Predicate, const T&)`
- Template Function `thrust::replace_copy_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate, const T&)`
- Template Function `thrust::replace_copy_if(InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate, const T&)`

- Template Function `thrust::replace_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate, const T&)`
- Template Function `thrust::replace_if(ForwardIterator, ForwardIterator, Predicate, const T&)`
- Template Function `thrust::replace_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, Predicate, const T&)`
- Template Function `thrust::replace_if(ForwardIterator, ForwardIterator, InputIterator, Predicate, const T&)`
- Template Function `thrust::return_temporary_buffer`
- Template Function `thrust::reverse(const thrust::detail::execution_policy_base<DerivedPolicy>&, BidirectionalIterator, BidirectionalIterator)`
- Template Function `thrust::reverse(BidirectionalIterator, BidirectionalIterator)`
- Template Function `thrust::reverse_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, BidirectionalIterator, BidirectionalIterator, OutputIterator)`
- Template Function `thrust::reverse_copy(BidirectionalIterator, BidirectionalIterator, OutputIterator)`
- Template Function `thrust::scatter(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator)`
- Template Function `thrust::scatter(InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator)`
- Template Function `thrust::scatter_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator3, RandomAccessIterator)`
- Template Function `thrust::scatter_if(InputIterator1, InputIterator1, InputIterator2, InputIterator3, RandomAccessIterator)`
- Template Function `thrust::scatter_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator3, RandomAccessIterator, Predicate)`
- Template Function `thrust::scatter_if(InputIterator1, InputIterator1, InputIterator2, InputIterator3, RandomAccessIterator, Predicate)`
- Template Function `thrust::sequence(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`
- Template Function `thrust::sequence(ForwardIterator, ForwardIterator)`
- Template Function `thrust::sequence(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, T)`
- Template Function `thrust::sequence(ForwardIterator, ForwardIterator, T)`
- Template Function `thrust::sequence(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, T, T)`
- Template Function `thrust::sequence(ForwardIterator, ForwardIterator, T, T)`
- Template Function `thrust::set_difference(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`
- Template Function `thrust::set_difference(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`
- Template Function `thrust::set_difference(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`
- Template Function `thrust::set_difference(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`

- [illegible]

- Template Function `thrust::set_symmetric_difference_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)`
- Template Function `thrust::set_union(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`
- Template Function `thrust::set_union(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`
- Template Function `thrust::set_union(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`
- Template Function `thrust::set_union(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`
- Template Function `thrust::set_union_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)`
- Template Function `thrust::set_union_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)`
- Template Function `thrust::set_union_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)`
- Template Function `thrust::set_union_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)`
- Template Function `thrust::sin`
- Template Function `thrust::sinh`
- Template Function `thrust::sort(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator, RandomAccessIterator)`
- Template Function `thrust::sort(RandomAccessIterator, RandomAccessIterator)`
- Template Function `thrust::sort(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator, RandomAccessIterator, StrictWeakOrdering)`
- Template Function `thrust::sort(RandomAccessIterator, RandomAccessIterator, StrictWeakOrdering)`
- Template Function `thrust::sort_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2)`
- Template Function `thrust::sort_by_key(RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2)`
- Template Function `thrust::sort_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2, StrictWeakOrdering)`
- Template Function `thrust::sort_by_key(RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2, StrictWeakOrdering)`
- Template Function `thrust::sqrt`
- Template Function `thrust::stable_partition(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate)`
- Template Function `thrust::stable_partition(ForwardIterator, ForwardIterator, Predicate)`
- Template Function `thrust::stable_partition(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, Predicate)`

- Template Function `thrust::stable_partition(ForwardIterator, ForwardIterator, InputIterator, Predicate)`
- Template Function `thrust::stable_partition_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)`
- Template Function `thrust::stable_partition_copy(InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)`
- Template Function `thrust::stable_partition_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, Predicate)`
- Template Function `thrust::stable_partition_copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, Predicate)`
- Template Function `thrust::stable_sort(RandomAccessIterator, RandomAccessIterator, StrictWeakOrdering)`
- Template Function `thrust::stable_sort(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator, RandomAccessIterator)`
- Template Function `thrust::stable_sort(RandomAccessIterator, RandomAccessIterator)`
- Template Function `thrust::stable_sort(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator, RandomAccessIterator, StrictWeakOrdering)`
- Template Function `thrust::stable_sort_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2)`
- Template Function `thrust::stable_sort_by_key(RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2)`
- Template Function `thrust::stable_sort_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2, StrictWeakOrdering)`
- Template Function `thrust::stable_sort_by_key(RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2, StrictWeakOrdering)`
- Template Function `thrust::swap(device_reference<T>&, device_reference<T>&)`
- Template Function `thrust::swap(pair<T1, T2>&, pair<T1, T2>&)`
- Template Function `thrust::swap(Assignable1&, Assignable2&)`
- Template Function `thrust::swap(tuple<T0, T1, T2, T3, T4, T5, T6, T7, T8, T9>&, tuple<U0, U1, U2, U3, U4, U5, U6, U7, U8, U9>&)`
- Template Function `thrust::swap_ranges(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator1, ForwardIterator1, ForwardIterator2)`
- Template Function `thrust::swap_ranges(ForwardIterator1, ForwardIterator1, ForwardIterator2)`
- Template Function `thrust::tabulate(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, UnaryOperation)`
- Template Function `thrust::tabulate(ForwardIterator, ForwardIterator, UnaryOperation)`
- Template Function `thrust::tan`
- Template Function `thrust::tanh`
- Template Function `thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T&, const T&, BinaryPredicate)`
- Template Function `thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T&, const T&)`
- Template Function `thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T&, const T&, BinaryPredicate)`

- Template Function `thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T&, const T&)`
- Template Function `thrust::tie(T0&)`
- Template Function `thrust::tie(T0&, T1&)`
- Template Function `thrust::transform(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, UnaryFunction)`
- Template Function `thrust::transform(InputIterator, InputIterator, OutputIterator, UnaryFunction)`
- Template Function `thrust::transform(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryFunction)`
- Template Function `thrust::transform(InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryFunction)`
- Template Function `thrust::transform_exclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, UnaryFunction, T, AssociativeOperator)`
- Template Function `thrust::transform_exclusive_scan(InputIterator, InputIterator, OutputIterator, UnaryFunction, T, AssociativeOperator)`
- Template Function `thrust::transform_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, ForwardIterator, UnaryFunction, Predicate)`
- Template Function `thrust::transform_if(InputIterator, InputIterator, ForwardIterator, UnaryFunction, Predicate)`
- Template Function `thrust::transform_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, ForwardIterator, UnaryFunction, Predicate)`
- Template Function `thrust::transform_if(InputIterator1, InputIterator1, InputIterator2, ForwardIterator, UnaryFunction, Predicate)`
- Template Function `thrust::transform_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator3, ForwardIterator, BinaryFunction, Predicate)`
- Template Function `thrust::transform_if(InputIterator1, InputIterator1, InputIterator2, InputIterator3, ForwardIterator, BinaryFunction, Predicate)`
- Template Function `thrust::transform_inclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, UnaryFunction, AssociativeOperator)`
- Template Function `thrust::transform_inclusive_scan(InputIterator, InputIterator, OutputIterator, UnaryFunction, AssociativeOperator)`
- Template Function `thrust::transform_reduce(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, UnaryFunction, OutputType, BinaryFunction)`
- Template Function `thrust::transform_reduce(InputIterator, InputIterator, UnaryFunction, OutputType, BinaryFunction)`
- Template Function `thrust::uninitialized_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, ForwardIterator)`
- Template Function `thrust::uninitialized_copy(InputIterator, InputIterator, ForwardIterator)`
- Template Function `thrust::uninitialized_copy_n(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, Size, ForwardIterator)`
- Template Function `thrust::uninitialized_copy_n(InputIterator, Size, ForwardIterator)`
- Template Function `thrust::uninitialized_fill(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&)`

- Template Function `thrust::uninitialized_fill(ForwardIterator, ForwardIterator, const T&)`
- Template Function `thrust::uninitialized_fill_n(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, Size, const T&)`
- Template Function `thrust::uninitialized_fill_n(ForwardIterator, Size, const T&)`
- Template Function `thrust::unique(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`
- Template Function `thrust::unique(ForwardIterator, ForwardIterator)`
- Template Function `thrust::unique(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, BinaryPredicate)`
- Template Function `thrust::unique(ForwardIterator, ForwardIterator, BinaryPredicate)`
- Template Function `thrust::unique_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator1, ForwardIterator1, ForwardIterator2)`
- Template Function `thrust::unique_by_key(ForwardIterator1, ForwardIterator1, ForwardIterator2)`
- Template Function `thrust::unique_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator1, ForwardIterator1, ForwardIterator2, BinaryPredicate)`
- Template Function `thrust::unique_by_key(ForwardIterator1, ForwardIterator1, ForwardIterator2, BinaryPredicate)`
- Template Function `thrust::unique_by_key_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)`
- Template Function `thrust::unique_by_key_copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)`
- Template Function `thrust::unique_by_key_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate)`
- Template Function `thrust::unique_by_key_copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate)`
- Template Function `thrust::unique_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator)`
- Template Function `thrust::unique_copy(InputIterator, InputIterator, OutputIterator)`
- Template Function `thrust::unique_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, BinaryPredicate)`
- Template Function `thrust::unique_copy(InputIterator, InputIterator, OutputIterator, BinaryPredicate)`
- Template Function `thrust::upper_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const LessThanComparable&)`
- Template Function `thrust::upper_bound(ForwardIterator, ForwardIterator, const LessThanComparable&)`
- Template Function `thrust::upper_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`
- Template Function `thrust::upper_bound(ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`
- Template Function `thrust::upper_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)`
- Template Function `thrust::upper_bound(ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)`

- Template Function `thrust::upper_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)`
- Template Function `thrust::upper_bound(ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)`

Variables

- Variable `thrust::device`
- Variable `thrust::host`

Namespace `thrust::placeholders`

Contents

- *Variables*

Variables

- Variable `thrust::placeholders::_1`
- Variable `thrust::placeholders::_10`
- Variable `thrust::placeholders::_2`
- Variable `thrust::placeholders::_3`
- Variable `thrust::placeholders::_4`
- Variable `thrust::placeholders::_5`
- Variable `thrust::placeholders::_6`
- Variable `thrust::placeholders::_7`
- Variable `thrust::placeholders::_8`
- Variable `thrust::placeholders::_9`

Namespace `thrust::random`

`thrust::random` is the namespace which contains random number engine class templates, random number engine adaptor class templates, engines with predefined parameters, and random number distribution class templates. They are provided in a separate namespace for import convenience but are also aliased in the top-level `thrust` namespace for easy access.

Contents

- *Typedefs*

Typedefs

- *Typedef* `thrust::random::default_random_engine`
- *Typedef* `thrust::random::ranlux24`
- *Typedef* `thrust::random::ranlux48`
- *Typedef* `thrust::random::taus88`

Namespace `thrust::system`

`thrust::system` is the namespace which contains functionality for manipulating memory specific to one of Thrust's backend systems. It also contains functionality for reporting error conditions originating from the operating system or other low-level application program interfaces such as the HIP runtime. They are provided in a separate namespace for import convenience but are also aliased in the top-level `thrust` namespace for easy access.

1.3.2 Classes and Structs

Template Struct `binary_function`

- Defined in `file_thrust_functional.h`

Struct Documentation

template<typename **Argument1**, typename **Argument2**, typename **Result**>
struct `binary_function`

binary_function is an empty base class: it contains no member functions or member variables, but only type information. The only reason it exists is to make it more convenient to define types that are models of the concept Adaptable Binary Function. Specifically, any model of Adaptable Binary Function must define nested typedefs. Those typedefs are provided by the base class *binary_function*.

The following code snippet demonstrates how to construct an Adaptable Binary Function using *binary_function*.

```
struct exponentiate : public thrust::binary_function<float, float, float>
{
    __host__ __device__
    float operator() (float x, float y) { return powf(x, y); }
};
```

Note Because C++11 language support makes the functionality of *binary_function* obsolete, its use is optional if C++11 language features are enabled.

See http://www.sgi.com/tech/stl/binary_function.html

See *unary_function*

Public Types

typedef `first_argument_type`

The type of the function object's first argument.


```
typedef second_argument_type
```

The type of the function object's second argument.

```
typedef result_type
```

The type of the function object's result;.

Template Struct `binary_negate`

- Defined in file `thrust_functional.h`

Inheritance Relationships

Base Type

- `public thrust::binary_function< Predicate::first_argument_type, Predicate::second_argument_type, bool >` (*Template Struct `binary_function`*)

Struct Documentation

```
template<typename Predicate>
```

```
struct binary_negate : public thrust::binary_function<Predicate::first_argument_type, Predicate::second_argument_type, bool>
```

`binary_negate` is a function object adaptor: it is an Adaptable Binary Predicate that represents the logical negation of some other Adaptable Binary Predicate. That is: if `f` is an object of class `binary_negate<AdaptablePredicate>`, then there exists an object `pred` of class `AdaptableBinaryPredicate` such that `f(x, y)` always returns the same value as `!pred(x, y)`. There is rarely any reason to construct a `binary_negate` directly; it is almost always easier to use the helper function `not2`.

See http://www.sgi.com/tech/stl/binary_negate.html

Public Functions

```
__host__ __device__ thrust::binary_negate::binary_negate(Predicate p)
```

Constructor takes a `Predicate` object to negate.

Parameters

- `p`: The `Predicate` object to negate.

```
__host__ __device__ bool thrust::binary_negate::operator() (const typename Predicate::first_argument_type& x, const typename Predicate::second_argument_type& y)
```

Function call operator. The return value is `!pred(x, y)`.

Template Struct `binary_traits`

- Defined in file `thrust_functional.h`

Struct Documentation

```
template<typename Operation>
```

```
struct binary_traits
```

Template Struct `bit_and`

- Defined in `file_thrust_functional.h`

Struct Documentation

`template<typename T>`

struct `bit_and`

`bit_and` is a function object. Specifically, it is an Adaptable Binary Function. If `f` is an object of class `bit_and<T>`, and `x` and `y` are objects of class `T`, then `f(x, y)` returns `x&y`.

The following code snippet demonstrates how to use `bit_and` to take the bitwise AND of one *device_vector* of ints by another.

Template Parameters

- `T`: is a model of `Assignable`, and if `x` and `y` are objects of type `T`, then `x&y` must be defined and must have a return type that is convertible to `T`.

```
#include <thrust/device_vector.h>
#include <thrust/functional.h>
#include <thrust/sequence.h>
#include <thrust/fill.h>
#include <thrust/transform.h>
...
const int N = 1000;
thrust::device_vector<int> V1(N);
thrust::device_vector<int> V2(N);
thrust::device_vector<int> V3(N);

thrust::sequence(V1.begin(), V1.end(), 1);
thrust::fill(V2.begin(), V2.end(), 13);

thrust::transform(V1.begin(), V1.end(), V2.begin(), V3.begin(),
                  thrust::bit_and<int>());
// V3 is now {1&13, 2&13, 3&13, ..., 1000&13}
```

See *binary_function*

Public Types

typedef `first_argument_type`

The type of the function object's first argument.

typedef `second_argument_type`

The type of the function object's second argument.

typedef `result_type`

The type of the function object's result;.

Public Functions

`__host__ __device__ T thrust::bit_and::operator()(const T & lhs, const T & rhs) const`

Function call operator. The return value is `lhs & rhs`.

Template Struct `bit_or`

- Defined in `file_thrust_functional.h`

Struct Documentation

`template<typename T>`

struct `bit_or`

`bit_or` is a function object. Specifically, it is an Adaptable Binary Function. If `f` is an object of class `bit_and<T>`, and `x` and `y` are objects of class `T`, then `f(x, y)` returns `x | y`.

The following code snippet demonstrates how to use `bit_or` to take the bitwise OR of one *device_vector* of ints by another.

Template Parameters

- `T`: is a model of `Assignable`, and if `x` and `y` are objects of type `T`, then `x | y` must be defined and must have a return type that is convertible to `T`.

```
#include <thrust/device_vector.h>
#include <thrust/functional.h>
#include <thrust/sequence.h>
#include <thrust/fill.h>
#include <thrust/transform.h>
...
const int N = 1000;
thrust::device_vector<int> V1(N);
thrust::device_vector<int> V2(N);
thrust::device_vector<int> V3(N);

thrust::sequence(V1.begin(), V1.end(), 1);
thrust::fill(V2.begin(), V2.end(), 13);

thrust::transform(V1.begin(), V1.end(), V2.begin(), V3.begin(),
    thrust::bit_or<int>());
// V3 is now {1|13, 2|13, 3|13, ..., 1000|13}
```

See *binary_function*

Public Types

typedef `first_argument_type`

The type of the function object's first argument.

typedef `second_argument_type`

The type of the function object's second argument.

typedef `result_type`

The type of the function object's result;.

Public Functions

`__host__ __device__ T thrust::bit_or::operator()(const T & lhs, const T & rhs) const`

Function call operator. The return value is `lhs | rhs`.

Template Struct `bit_xor`

- Defined in `file_thrust_functional.h`

Struct Documentation

`template<typename T>`

struct `bit_xor`

bit_xor is a function object. Specifically, it is an Adaptable Binary Function. If *f* is an object of class `bit_and<T>`, and *x* and *y* are objects of class *T*, then *f* (*x*, *y*) returns *x*^*y*.

The following code snippet demonstrates how to use *bit_xor* to take the bitwise XOR of one *device_vector* of *ints* by another.

Template Parameters

- *T*: is a model of `Assignable`, and if *x* and *y* are objects of type *T*, then *x*^*y* must be defined and must have a return type that is convertible to *T*.

```
#include <thrust/device_vector.h>
#include <thrust/functional.h>
#include <thrust/sequence.h>
#include <thrust/fill.h>
#include <thrust/transform.h>
...
const int N = 1000;
thrust::device_vector<int> V1(N);
thrust::device_vector<int> V2(N);
thrust::device_vector<int> V3(N);

thrust::sequence(V1.begin(), V1.end(), 1);
thrust::fill(V2.begin(), V2.end(), 13);

thrust::transform(V1.begin(), V1.end(), V2.begin(), V3.begin(),
                  thrust::bit_xor<int>());
// V3 is now {1^13, 2^13, 3^13, ..., 1000^13}
```

See *binary_function*

Public Types

typedef `first_argument_type`

The type of the function object's first argument.

typedef `second_argument_type`

The type of the function object's second argument.

typedef `result_type`

The type of the function object's result;.

Public Functions

`__host__ __device__ T thrust::bit_xor::operator()(const T & lhs, const T & rhs) const`

Function call operator. The return value is *lhs* ^ *rhs*.

Template Struct complex

- Defined in file_thrust_complex.h

Struct Documentation

template<typename T>

struct complex

`complex` is the Thrust equivalent to `std::complex`. It is functionally identical to it, but can also be used in device code which `std::complex` currently cannot.

Template Parameters

- T: The type used to hold the real and imaginary parts. Should be `float` or `double`. Others types are not supported.

Public Types

typedef T value_type

`value_type` is the type of `complex`'s real and imaginary parts.

Public Functions

`__host__ __device__ thrust::complex::complex(const T & re)`

Construct a complex number with an imaginary part of 0.

Parameters

- re: The real part of the number.

`__host__ __device__ thrust::complex::complex(const T & re, const T & im)`

Construct a complex number from its real and imaginary parts.

Parameters

- re: The real part of the number.
- im: The imaginary part of the number.

`__host__ __device__ thrust::complex::complex()`

Default construct a complex number.

`__host__ __device__ thrust::complex::complex(const complex < T > & z)`

This copy constructor copies from a `complex` with a type that is convertible to this `complex`'s `value_type`.

Parameters

- z: The complex to copy from.

`template<typename U> __host__ __device__ thrust::complex::complex(const complex < U > & z)`

This converting copy constructor copies from a `complex` with a type that is convertible to this `complex`'s `value_type`.

Parameters

- `z`: The `complex` to copy from.

Template Parameters

- `U`: is convertible to `value_type`.

`__host__ THRUST_STD_COMPLEX_DEVICE thrust::complex::complex(const std::complex< T > & z)`
This converting copy constructor copies from a `std::complex` with a type that is convertible to this `complex`'s `value_type`.

Parameters

- `z`: The `complex` to copy from.

`template<typename U> __host__ THRUST_STD_COMPLEX_DEVICE thrust::complex::complex(const U & z)`
This converting copy constructor copies from a `std::complex` with a type that is convertible to this `complex`'s `value_type`.

Parameters

- `z`: The `complex` to copy from.

Template Parameters

- `U`: is convertible to `value_type`.

`__host__ __device__ complex& thrust::complex::operator=(const T & re)`
Assign `re` to the real part of this `complex` and set the imaginary part to 0.

Parameters

- `re`: The real part of the number.

`__host__ __device__ complex& thrust::complex::operator=(const complex < T > & z)`
Assign `z.real()` and `z.imag()` to the real and imaginary parts of this `complex` respectively.

Parameters

- `z`: The `complex` to copy from.

`template<typename U> __host__ __device__ complex& thrust::complex::operator=(const U & z)`
Assign `z.real()` and `z.imag()` to the real and imaginary parts of this `complex` respectively.

Parameters

- `z`: The `complex` to copy from.

Template Parameters

- `U`: is convertible to `value_type`.

`__host__ THRUST_STD_COMPLEX_DEVICE complex& thrust::complex::operator=(const std::complex< T > & z)`
Assign `z.real()` and `z.imag()` to the real and imaginary parts of this `complex` respectively.

Parameters

- `z`: The `complex` to copy from.

`template<typename U> __host__ THRUST_STD_COMPLEX_DEVICE complex& thrust::complex::operator=(const U & z)`
Assign `z.real()` and `z.imag()` to the real and imaginary parts of this `complex` respectively.

Parameters

- `z`: The `complex` to copy from.

Template Parameters

- U: is convertible to `value_type`.

```
template<typename U>__host__ __device__ complex<T>& thrust::complex::operator+=(const  
Adds a complex to this complex and assigns the result to this complex.
```

Parameters

- z: The complex to be added.

Template Parameters

- U: is convertible to `value_type`.

```
template<typename U>__host__ __device__ complex<T>& thrust::complex::operator-=(const  
Subtracts a complex from this complex and assigns the result to this complex.
```

Parameters

- z: The complex to be subtracted.

Template Parameters

- U: is convertible to `value_type`.

```
template<typename U>__host__ __device__ complex<T>& thrust::complex::operator*=(const  
Multiplies this complex by another complex and assigns the result to this complex.
```

Parameters

- z: The complex to be multiplied.

Template Parameters

- U: is convertible to `value_type`.

```
template<typename U>__host__ __device__ complex<T>& thrust::complex::operator/=(const  
Divides this complex by another complex and assigns the result to this complex.
```

Parameters

- z: The complex to be divided.

Template Parameters

- U: is convertible to `value_type`.

```
template<typename U>__host__ __device__ complex<T>& thrust::complex::operator+=(const  
Adds a scalar to this complex and assigns the result to this complex.
```

Parameters

- z: The complex to be added.

Template Parameters

- U: is convertible to `value_type`.

```
template<typename U>__host__ __device__ complex<T>& thrust::complex::operator-=(const  
Subtracts a scalar from this complex and assigns the result to this complex.
```

Parameters

- z: The scalar to be subtracted.

Template Parameters

- U: is convertible to `value_type`.

```
template<typename U> __host__ __device__ complex<T>& thrust::complex::operator*=(const U& z)
```

Multiplies this `complex` by a scalar and assigns the result to this `complex`.

Parameters

- z: The scalar to be multiplied.

Template Parameters

- U: is convertible to `value_type`.

```
template<typename U> __host__ __device__ complex<T>& thrust::complex::operator/=(const U& z)
```

Divides this `complex` by a scalar and assigns the result to this `complex`.

Parameters

- z: The scalar to be divided.

Template Parameters

- U: is convertible to `value_type`.

```
__host__ __device__ T thrust::complex::real() const volatile
```

Returns the real part of this `complex`.

```
__host__ __device__ T thrust::complex::imag() const volatile
```

Returns the imaginary part of this `complex`.

```
__host__ __device__ T thrust::complex::real() const
```

Returns the real part of this `complex`.

```
__host__ __device__ T thrust::complex::imag() const
```

Returns the imaginary part of this `complex`.

```
__host__ __device__ void thrust::complex::real(T re) volatile
```

Sets the real part of this `complex`.

Parameters

- re: The new real part of this `complex`.

```
__host__ __device__ void thrust::complex::imag(T im) volatile
```

Sets the imaginary part of this `complex`.

Parameters

- im: The new imaginary part of this `complex`.

```
__host__ __device__ void thrust::complex::real(T re)
```

Sets the real part of this `complex`.

Parameters

- re: The new real part of this `complex`.

```
__host__ __device__ void thrust::complex::imag(T im)
```

Sets the imaginary part of this `complex`.

Parameters

- `im`: The new imaginary part of this `complex`.

`__host__ __device__ thrust::complex::operator std::complex< T >() const`
Casts this `complex` to a `std::complex` of the same type.

Template Struct `device_allocator::rebind`

- Defined in `file_thrust_device_allocator.h`

Nested Relationships

This struct is a nested type of *Template Class `device_allocator`*.

Struct Documentation

`template<typename U>`

struct `rebind`

The `rebind` metafunction provides the type of a *`device_allocator`* instantiated with another type.

Template Parameters

- `U`: The other type to use for instantiation.

Public Types

`template<>`

typedef `device_allocator<U> other`

The typedef `other` gives the type of the rebound *`device_allocator`*.

Template Struct `device_allocator< void >::device_allocator< void >`

- Defined in `file_thrust_device_allocator.h`

Nested Relationships

This struct is a nested type of *Template Class `device_allocator< void >`*.

Struct Documentation

`template<typename U>`

struct `rebind`

The `rebind` metafunction provides the type of a *`device_allocator`* instantiated with another type.

Template Parameters

- `U`: The other type to use for instantiation.

Public Types

```
template<>
```

```
typedef device_allocator<U> other
```

The typedef `other` gives the type of the rebound `device_allocator`.

Template Struct `device_execution_policy`

- Defined in file `thrust_execution_policy.h`

Inheritance Relationships

Base Type

- `public thrust::system::__THRUST_DEVICE_SYSTEM_NAMESPACE::execution_policy<DerivedPolicy >`

Struct Documentation

```
template<typename DerivedPolicy>
```

```
struct device_execution_policy : public thrust::system::__THRUST_DEVICE_SYSTEM_NAMESPACE::execution_p
```

`device_execution_policy` is the base class for all Thrust parallel execution policies which are derived from Thrust's default device backend system configured with the `THRUST_DEVICE_SYSTEM` macro.

Custom user-defined backends which wish to inherit the functionality of Thrust's device backend system should derive a policy from this type in order to interoperate with Thrust algorithm dispatch.

The following code snippet demonstrates how to derive a standalone custom execution policy from `thrust::device_execution_policy` to implement a backend which specializes `for_each` while inheriting the behavior of every other algorithm from the device system:

```
#include <thrust/execution_policy.h>
#include <iostream>

// define a type derived from thrust::device_execution_policy to distinguish our
↳ custom execution policy:
struct my_policy : thrust::device_execution_policy<my_policy> {};

// overload for_each on my_policy
template<typename Iterator, typename Function>
Iterator for_each(my_policy, Iterator first, Iterator last, Function f)
{
    std::cout << "Hello, world from for_each(my_policy)!" << std::endl;

    for(; first < last; ++first)
    {
        f(*first);
    }

    return first;
}

struct ignore_argument
```

(continues on next page)

(continued from previous page)

```

{
    void operator() (int) {}
};

int main()
{
    int data[4];

    // dispatch thrust::for_each using our custom policy:
    my_policy exec;
    thrust::for_each(exec, data, data + 4, ignore_argument());

    // dispatch thrust::transform whose behavior our policy inherits
    thrust::transform(exec, data, data, + 4, data, thrust::identity<int>());

    return 0;
}

```

See `execution_policy`

See `host_execution_policy`

Template Struct `device_malloc_allocator::rebind`

- Defined in `file_thrust_device_malloc_allocator.h`

Nested Relationships

This struct is a nested type of *Template Class* `device_malloc_allocator`.

Struct Documentation

template<typename `U`>

struct `rebind`

The `rebind` metafunction provides the type of a `device_malloc_allocator` instantiated with another type.

Template Parameters

- `U`: The other type to use for instantiation.

Public Types

template<>

typedef `device_malloc_allocator<U>` `other`

The typedef `other` gives the type of the rebound `device_malloc_allocator`.

Template Struct `device_new_allocator::rebind`

- Defined in `file_thrust_device_new_allocator.h`

Nested Relationships

This struct is a nested type of *Template Class device_new_allocator*.

Struct Documentation

```
template<typename U>
struct rebind
```

The `rebind` metafunction provides the type of a *device_new_allocator* instantiated with another type.

Template Parameters

- `U`: The other type to use for instantiation.

Public Types

```
template<>
typedef device_new_allocator<U> other
```

The typedef `other` gives the type of the rebound *device_new_allocator*.

Template Struct divides

- Defined in `file_thrust_functional.h`

Struct Documentation

```
template<typename T>
struct divides
```

`divides` is a function object. Specifically, it is an Adaptable Binary Function. If `f` is an object of class `divides<T>`, and `x` and `y` are objects of class `T`, then `f(x, y)` returns `x/y`.

The following code snippet demonstrates how to use `divides` to divide one `device_vectors` of `floats` by another.

Template Parameters

- `T`: is a model of `Assignable`, and if `x` and `y` are objects of type `T`, then `x/y` must be defined and must have a return type that is convertible to `T`.

```
#include <thrust/device_vector.h>
#include <thrust/functional.h>
#include <thrust/sequence.h>
#include <thrust/fill.h>
#include <thrust/transform.h>
...
const int N = 1000;
thrust::device_vector<float> V1(N);
thrust::device_vector<float> V2(N);
thrust::device_vector<float> V3(N);

thrust::sequence(V1.begin(), V1.end(), 1);
thrust::fill(V2.begin(), V2.end(), 75);
```

(continues on next page)

(continued from previous page)

```
thrust::transform(V1.begin(), V1.end(), V2.begin(), V3.begin(),
    thrust::divides<float>());
// V3 is now {1/75, 2/75, 3/75, ..., 1000/75}
```

See <http://www.sgi.com/tech/stl/divides.html>

See *binary_function*

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

__host__ __device__ T thrust::divides::operator()(const T & lhs, const T & rhs) const

Function call operator. The return value is `lhs / rhs`.

Template Struct `equal_to`

- Defined in file `thrust_functional.h`

Struct Documentation

template<typename T>

struct equal_to

`equal_to` is a function object. Specifically, it is an Adaptable Binary Predicate, which means it is a function object that tests the truth or falsehood of some condition. If `f` is an object of class `equal_to<T>` and `x` and `y` are objects of class `T`, then `f(x, y)` returns `true` if `x == y` and `false` otherwise.

See http://www.sgi.com/tech/stl/equal_to.html

See *binary_function*

Template Parameters

- `T`: is a model of `Equality Comparable`.

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

__host__ __device__ bool thrust::equal_to::operator() (const T & lhs, const T & rhs) const

Function call operator. The return value is `lhs == rhs`.

Template Struct greater

- Defined in `file_thrust_functional.h`

Struct Documentation

template<typename T>

struct greater

`greater` is a function object. Specifically, it is an Adaptable Binary Predicate, which means it is a function object that tests the truth or falsehood of some condition. If `f` is an object of class `greater<T>` and `x` and `y` are objects of class `T`, then `f(x,y)` returns `true` if `x > y` and `false` otherwise.

See <http://www.sgi.com/tech/stl/greater.html>

See *binary_function*

Template Parameters

- `T`: is a model of [LessThan Comparable](#).

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

__host__ __device__ bool thrust::greater::operator() (const T & lhs, const T & rhs) const

Function call operator. The return value is `lhs > rhs`.

Template Struct greater_equal

- Defined in `file_thrust_functional.h`

Struct Documentation

template<typename T>

struct greater_equal

greater_equal is a function object. Specifically, it is an Adaptable Binary Predicate, which means it is a function object that tests the truth or falsehood of some condition. If *f* is an object of class `greater_equal<T>` and *x* and *y* are objects of class *T*, then *f*(*x*, *y*) returns `true` if *x* >= *y* and `false` otherwise.

See http://www.sgi.com/tech/stl/greater_equal.html

See *binary_function*

Template Parameters

- *T*: is a model of `LessThan Comparable`.

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

__host__ __device__ bool thrust::greater_equal::operator()(const T & lhs, const T & rhs)

Function call operator. The return value is `lhs >= rhs`.

Template Struct `host_execution_policy`

- Defined in `file_thrust_execution_policy.h`

Inheritance Relationships

Base Type

- `public thrust::system::__THRUST_HOST_SYSTEM_NAMESPACE::execution_policy<DerivedPolicy>`

Struct Documentation

template<typename **DerivedPolicy**>

struct host_execution_policy: **public** thrust::system::__THRUST_HOST_SYSTEM_NAMESPACE::execution_policy<

host_execution_policy is the base class for all Thrust parallel execution policies which are derived from Thrust's default host backend system configured with the `THRUST_HOST_SYSTEM` macro.

Custom user-defined backends which wish to inherit the functionality of Thrust's host backend system should derive a policy from this type in order to interoperate with Thrust algorithm dispatch.

The following code snippet demonstrates how to derive a standalone custom execution policy from `thrust::host_execution_policy` to implement a backend which specializes `for_each` while inheriting the behavior of every other algorithm from the host system:

```
#include <thrust/execution_policy.h>
#include <iostream>

// define a type derived from thrust::host_execution_policy to distinguish our
// custom execution policy:
struct my_policy : thrust::host_execution_policy<my_policy> {};

// overload for_each on my_policy
template<typename Iterator, typename Function>
Iterator for_each(my_policy, Iterator first, Iterator last, Function f)
{
    std::cout << "Hello, world from for_each(my_policy)!" << std::endl;

    for(; first < last; ++first)
    {
        f(*first);
    }

    return first;
}

struct ignore_argument
{
    void operator()(int) {}
};

int main()
{
    int data[4];

    // dispatch thrust::for_each using our custom policy:
    my_policy exec;
    thrust::for_each(exec, data, data + 4, ignore_argument());

    // dispatch thrust::transform whose behavior our policy inherits
    thrust::transform(exec, data, data, + 4, data, thrust::identity<int>());

    return 0;
}
```

See `execution_policy`

See `device_execution_policy`

Template Struct identity

- Defined in `file_thrust_functional.h`

Struct Documentation

template<typename T>

struct identity

`identity` is a Unary Function that represents the identity function: it takes a single argument `x`, and returns `x`.

The following code snippet demonstrates that `identity` returns its argument.

Template Parameters

- T: No requirements on T.

```
#include <thrust/functional.h>
#include <assert.h>
...
int x = 137;
thrust::identity<int> id;
assert(x == id(x));
```

See <http://www.sgi.com/tech/stl/identity.html>

See *unary_function*

Public Types

typedef argument_type

The type of the function object's first argument.

typedef result_type

The type of the function object's result;.

Public Functions

`__host__ __device__ const T& thrust::identity::operator()(const T & x) const`

Function call operator. The return value is `x`.

Template Struct less

- Defined in `file_thrust_functional.h`

Struct Documentation

template<typename T>

struct less

`less` is a function object. Specifically, it is an Adaptable Binary Predicate, which means it is a function object that tests the truth or falsehood of some condition. If `f` is an object of class `less<T>` and `x` and `y` are objects of class `T`, then `f(x, y)` returns `true` if `x < y` and `false` otherwise.

See <http://www.sgi.com/tech/stl/less.html>

See *binary_function*

Template Parameters

- T: is a model of [LessThan Comparable](#).

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

__host__ __device__ bool thrust::less::operator() (const T & lhs, const T & rhs) const
Function call operator. The return value is `lhs < rhs`.

Template Struct less_equal

- Defined in file `thrust_functional.h`

Struct Documentation

template<typename T>

struct less_equal

less_equal is a function object. Specifically, it is an Adaptable Binary Predicate, which means it is a function object that tests the truth or falsehood of some condition. If *f* is an object of class `less_equal<T>` and *x* and *y* are objects of class *T*, then *f* (*x*, *y*) returns `true` if *x* `<=` *y* and `false` otherwise.

See http://www.sgi.com/tech/stl/less_equal.html

See [binary_function](#)

Template Parameters

- T: is a model of [LessThan Comparable](#).

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

__host__ __device__ bool thrust::less_equal::operator() (const T & lhs, const T & rhs)
Function call operator. The return value is `lhs <= rhs`.

Template Struct `logical_and`

- Defined in `file_thrust_functional.h`

Struct Documentation

template<typename T>

struct `logical_and`

logical_and is a function object. Specifically, it is an Adaptable Binary Predicate, which means it is a function object that tests the truth or falsehood of some condition. If *f* is an object of class `logical_and<T>` and *x* and *y* are objects of class *T* (where *T* is convertible to `bool`) then *f* (*x*, *y*) returns `true` if and only if both *x* and *y* are `true`.

See http://www.sgi.com/tech/stl/logical_and.html

See *binary_function*

Template Parameters

- *T*: must be convertible to `bool`.

Public Types

typedef `first_argument_type`

The type of the function object's first argument.

typedef `second_argument_type`

The type of the function object's second argument.

typedef `result_type`

The type of the function object's result;.

Public Functions

`__host__ __device__ bool thrust::logical_and::operator() (const T & lhs, const T & rhs)`

Function call operator. The return value is `lhs && rhs`.

Template Struct `logical_not`

- Defined in `file_thrust_functional.h`

Struct Documentation

template<typename T>

struct `logical_not`

logical_not is a function object. Specifically, it is an Adaptable Predicate, which means it is a function object that tests the truth or falsehood of some condition. If *f* is an object of class `logical_not<T>` and *x* is an object of class *T* (where *T* is convertible to `bool`) then *f* (*x*) returns `true` if and only if *x* is `false`.

The following code snippet demonstrates how to use *logical_not* to transform a *device_vector* of `bools` into its logical complement.

Template Parameters

- T: must be convertible to bool.

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
...
thrust::device_vector<bool> V;
...
thrust::transform(V.begin(), V.end(), V.begin(), thrust::logical_not<bool>());
// The elements of V are now the logical complement of what they were prior
```

See http://www.sgi.com/tech/stl/logical_not.html

See *unary_function*

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

__host__ __device__ bool thrust::logical_not::operator() (const T & x) const
Function call operator. The return value is !x.

Template Struct logical_or

- Defined in file_thrust_functional.h

Struct Documentation

template<typename T>
struct logical_or

logical_or is a function object. Specifically, it is an Adaptable Binary Predicate, which means it is a function object that tests the truth or falsehood of some condition. If *f* is an object of class *logical_or*<T> and *x* and *y* are objects of class T (where T is convertible to bool) then *f* (*x*, *y*) returns true if and only if either *x* or *y* are true.

See http://www.sgi.com/tech/stl/logical_or.html

See *binary_function*

Template Parameters

- T: must be convertible to bool.

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

__host__ __device__ bool thrust::logical_or::operator() (const T & lhs, const T & rhs)

Function call operator. The return value is `lhs || rhs`.

Template Struct maximum

- Defined in `file_thrust_functional.h`

Struct Documentation

template<typename T>

struct maximum

`maximum` is a function object that takes two arguments and returns the greater of the two. Specifically, it is an Adaptable Binary Function. If `f` is an object of class `maximum<T>` and `x` and `y` are objects of class `T` `f(x, y)` returns `x` if `x > y` and `y`, otherwise.

The following code snippet demonstrates that `maximum` returns its greater argument.

Template Parameters

- `T`: is a model of [LessThan Comparable](#).

```
#include <thrust/functional.h>
#include <assert.h>
...
int x = 137;
int y = -137;
thrust::maximum<int> mx;
assert(x == mx(x, y));
```

See [minimum](#)

See [min](#)

See [binary_function](#)

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

__host__ __device__ T thrust::maximum::operator()(const T & lhs, const T & rhs) const

Function call operator. The return value is $\text{rhs} < \text{lhs} ? \text{lhs} : \text{rhs}$.

Template Struct minimum

- Defined in file_thrust_functional.h

Struct Documentation

template<typename T>

struct minimum

`minimum` is a function object that takes two arguments and returns the lesser of the two. Specifically, it is an Adaptable Binary Function. If `f` is an object of class `minimum<T>` and `x` and `y` are objects of class `T` `f(x, y)` returns `x` if `x < y` and `y`, otherwise.

The following code snippet demonstrates that `minimum` returns its lesser argument.

Template Parameters

- `T`: is a model of [LessThan Comparable](#).

```
#include <thrust/functional.h>
#include <assert.h>
...
int x = 137;
int y = -137;
thrust::minimum<int> mn;
assert(y == mn(x, y));
```

See [maximum](#)

See [max](#)

See [binary_function](#)

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

`__host__ __device__ T thrust::minimum::operator()(const T & lhs, const T & rhs) const`
 Function call operator. The return value is `lhs < rhs ? lhs : rhs`.

Template Struct minus

- Defined in file `thrust_functional.h`

Struct Documentation

template<typename T>

struct minus

`minus` is a function object. Specifically, it is an Adaptable Binary Function. If `f` is an object of class `minus<T>`, and `x` and `y` are objects of class `T`, then `f(x, y)` returns `x-y`.

The following code snippet demonstrates how to use `minus` to subtract a *device_vector* of floats from another.

Template Parameters

- `T`: is a model of [Assignable](#), and if `x` and `y` are objects of type `T`, then `x-y` must be defined and must have a return type that is convertible to `T`.

```
#include <thrust/device_vector.h>
#include <thrust/functional.h>
#include <thrust/sequence.h>
#include <thrust/fill.h>
#include <thrust/transform.h>
...
const int N = 1000;
thrust::device_vector<float> V1(N);
thrust::device_vector<float> V2(N);
thrust::device_vector<float> V3(N);

thrust::sequence(V1.begin(), V1.end(), 1);
thrust::fill(V2.begin(), V2.end(), 75);

thrust::transform(V1.begin(), V1.end(), V2.begin(), V3.begin(),
    thrust::minus<float>());
// V3 is now {-74, -75, -76, ..., -925}
```

See <http://www.sgi.com/tech/stl/minus.html>

See *binary_function*

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

`__host__ __device__ T thrust::minus::operator()(const T & lhs, const T & rhs) const`
Function call operator. The return value is `lhs - rhs`.

Template Struct modulus

- Defined in file `thrust_functional.h`

Struct Documentation

template<typename T>

struct modulus

`modulus` is a function object. Specifically, it is an Adaptable Binary Function. If `f` is an object of class `modulus<T>`, and `x` and `y` are objects of class `T`, then `f(x, y)` returns `x % y`.

The following code snippet demonstrates how to use `modulus` to take the modulus of one `device_vectors` of `floats` by another.

Template Parameters

- `T`: is a model of [Assignable](#), and if `x` and `y` are objects of type `T`, then `x % y` must be defined and must have a return type that is convertible to `T`.

```
#include <thrust/device_vector.h>
#include <thrust/functional.h>
#include <thrust/sequence.h>
#include <thrust/fill.h>
#include <thrust/transform.h>
...
const int N = 1000;
thrust::device_vector<float> V1(N);
thrust::device_vector<float> V2(N);
thrust::device_vector<float> V3(N);

thrust::sequence(V1.begin(), V1.end(), 1);
thrust::fill(V2.begin(), V2.end(), 75);

thrust::transform(V1.begin(), V1.end(), V2.begin(), V3.begin(),
    thrust::modulus<int>());
// V3 is now {1%75, 2%75, 3%75, ..., 1000%75}
```

See <http://www.sgi.com/tech/stl/modulus.html>

See [binary_function](#)

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

`__host__ __device__ T thrust::modulus::operator()(const T & lhs, const T & rhs) const`
 Function call operator. The return value is `lhs % rhs`.

Template Struct multiplies

- Defined in `file_thrust_functional.h`

Struct Documentation

template<typename T>

struct multiplies

`multiplies` is a function object. Specifically, it is an Adaptable Binary Function. If `f` is an object of class `minus<T>`, and `x` and `y` are objects of class `T`, then `f(x, y)` returns `x*y`.

The following code snippet demonstrates how to use `multiplies` to multiply two `device_vectors` of `floats`.

Template Parameters

- `T`: is a model of [Assignable](#), and if `x` and `y` are objects of type `T`, then `x*y` must be defined and must have a return type that is convertible to `T`.

```
#include <thrust/device_vector.h>
#include <thrust/functional.h>
#include <thrust/sequence.h>
#include <thrust/fill.h>
#include <thrust/transform.h>
...
const int N = 1000;
thrust::device_vector<float> V1(N);
thrust::device_vector<float> V2(N);
thrust::device_vector<float> V3(N);

thrust::sequence(V1.begin(), V1.end(), 1);
thrust::fill(V2.begin(), V2.end(), 75);

thrust::transform(V1.begin(), V1.end(), V2.begin(), V3.begin(),
                  thrust::multiplies<float>());
// V3 is now {75, 150, 225, ..., 75000}
```

See <http://www.sgi.com/tech/stl/multiplies.html>

See [binary_function](#)

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

`__host__ __device__ T thrust::multiplies::operator() (const T & lhs, const T & rhs) con`
Function call operator. The return value is `lhs * rhs`.

Template Struct negate

- Defined in file `_thrust_functional.h`

Struct Documentation

`template<typename T>`

struct negate

`negate` is a function object. Specifically, it is an Adaptable Unary Function. If `f` is an object of class `negate<T>`, and `x` is an object of class `T`, then `f(x)` returns `-x`.

The following code snippet demonstrates how to use `negate` to negate the element of a *device_vector* of floats.

Template Parameters

- `T`: is a model of [Assignable](#), and if `x` is an object of type `T`, then `-x` must be defined and must have a return type that is convertible to `T`.

```
#include <thrust/device_vector.h>
#include <thrust/functional.h>
#include <thrust/sequence.h>
#include <thrust/transform.h>
...
const int N = 1000;
thrust::device_vector<float> V1(N);
thrust::device_vector<float> V2(N);

thrust::sequence(V1.begin(), V1.end(), 1);

thrust::transform(V1.begin(), V1.end(), V2.begin(),
                 thrust::negate<float>());
// V2 is now {-1, -2, -3, ..., -1000}
```

See <http://www.sgi.com/tech/stl/negate.html>

See *unary_function*

Public Types

typedef argument_type

The type of the function object's argument.

typedef result_type

The type of the function object's result;.

Public Functions

`__host__ __device__ T thrust::negate::operator() (const T & x) const`
 Function call operator. The return value is $-x$.

Template Struct `not_equal_to`

- Defined in `file_thrust_functional.h`

Struct Documentation

template<typename T>

struct not_equal_to

`not_equal_to` is a function object. Specifically, it is an Adaptable Binary Predicate, which means it is a function object that tests the truth or falsehood of some condition. If `f` is an object of class `not_equal_to<T>` and `x` and `y` are objects of class `T`, then `f(x, y)` returns `true` if `x != y` and `false` otherwise.

See http://www.sgi.com/tech/stl/not_equal_to.html

See *binary_function*

Template Parameters

- `T`: is a model of [Equality Comparable](#).

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

`__host__ __device__ bool thrust::not_equal_to::operator() (const T & lhs, const T & rhs)`
 Function call operator. The return value is `lhs != rhs`.

Template Struct `pair`

- Defined in `file_thrust_pair.h`

Struct Documentation

template<typename T1, typename T2>

struct pair

`pair` is a generic data structure encapsulating a heterogeneous pair of values.

Template Parameters

- T1: The type of `pair`'s first object type. There are no requirements on the type of T1. T1's type is provided by `pair::first_type`.
- T2: The type of `pair`'s second object type. There are no requirements on the type of T2. T2's type is provided by `pair::second_type`.

Public Types

typedef T1 first_type

`first_type` is the type of `pair`'s first object type.

typedef T2 second_type

`second_type` is the type of `pair`'s second object type.

Public Functions

__host__ __device__ thrust::pair::pair(void)

`pair`'s default constructor constructs `first` and `second` using `first_type` & `second_type`'s default constructors, respectively.

__host__ __device__ thrust::pair::pair(const T1 & x, const T2 & y)

This constructor accepts two objects to copy into this `pair`.

Parameters

- x: The object to copy into `first`.
- y: The object to copy into `second`.

template<typename U1, typename U2>__host__ __device__ thrust::pair::pair(const pair & p)

This copy constructor copies from a `pair` whose types are convertible to this `pair`'s `first_type` and `second_type`, respectively.

Parameters

- p: The `pair` to copy from.

Template Parameters

- U1: is convertible to `first_type`.
- U2: is convertible to `second_type`.

template<typename U1, typename U2>__host__ __device__ thrust::pair::pair(const std::pair & p)

This copy constructor copies from a `std::pair` whose types are convertible to this `pair`'s `first_type` and `second_type`, respectively.

Parameters

- p: The `std::pair` to copy from.

Template Parameters

- U1: is convertible to `first_type`.
- U2: is convertible to `second_type`.

__host__ __device__ void thrust::pair::swap(pair & p)

`swap` swaps the elements of two `pairs`.

Parameters

- `p`: The other pair with which to swap.

Public Members

first_type **first**

The pair's first object.

second_type **second**

The pair's second object.

Template Struct plus

- Defined in file `thrust_functional.h`

Struct Documentation

template<typename **T**>

struct plus

`plus` is a function object. Specifically, it is an Adaptable Binary Function. If `f` is an object of class `plus<T>`, and `x` and `y` are objects of class `T`, then `f(x, y)` returns `x+y`.

The following code snippet demonstrates how to use `plus` to sum two `device_vectors` of `floats`.

Template Parameters

- `T`: is a model of [Assignable](#), and if `x` and `y` are objects of type `T`, then `x+y` must be defined and must have a return type that is convertible to `T`.

```
#include <thrust/device_vector.h>
#include <thrust/functional.h>
#include <thrust/sequence.h>
#include <thrust/fill.h>
#include <thrust/transform.h>
...
const int N = 1000;
thrust::device_vector<float> V1(N);
thrust::device_vector<float> V2(N);
thrust::device_vector<float> V3(N);

thrust::sequence(V1.begin(), V1.end(), 1);
thrust::fill(V2.begin(), V2.end(), 75);

thrust::transform(V1.begin(), V1.end(), V2.begin(), V3.begin(),
                  thrust::plus<float>());
// V3 is now {76, 77, 78, ..., 1075}
```

See <http://www.sgi.com/tech/stl/plus.html>

See [binary_function](#)

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

__host__ __device__ T thrust::plus::operator()(const T & lhs, const T & rhs) const

Function call operator. The return value is `lhs + rhs`.

Template Struct `project1st`

- Defined in `file_thrust_functional.h`

Struct Documentation

template<typename **T1**, typename **T2**>

struct project1st

project1st is a function object that takes two arguments and returns its first argument; the second argument is unused. It is essentially a generalization of identity to the case of a Binary Function.

```
#include <thrust/functional.h>
#include <assert.h>
...
int x = 137;
int y = -137;
thrust::project1st<int> pj1;
assert(x == pj1(x,y));
```

See *identity*

See *project2nd*

See *binary_function*

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

__host__ __device__ const T1& thrust::project1st::operator()(const T1 & lhs, const T2 & rhs) const

Function call operator. The return value is `lhs`.

Template Struct project2nd

- Defined in file_thrust_functional.h

Struct Documentation

template<typename **T1**, typename **T2**>

struct project2nd

project2nd is a function object that takes two arguments and returns its second argument; the first argument is unused. It is essentially a generalization of identity to the case of a Binary Function.

```
#include <thrust/functional.h>
#include <assert.h>
...
int x = 137;
int y = -137;
thrust::project2nd<int> pj2;
assert(y == pj2(x,y));
```

See *identity*

See *project1st*

See *binary_function*

Public Types

typedef first_argument_type

The type of the function object's first argument.

typedef second_argument_type

The type of the function object's second argument.

typedef result_type

The type of the function object's result;.

Public Functions

__host__ __device__ const T2& thrust::project2nd::operator() (const T1 &, const T2 & rhs)

Function call operator. The return value is *rhs*.

Template Struct tuple_element

- Defined in file_thrust_pair.h

Struct Documentation

template<int **N**, class **T**>

struct tuple_element

This convenience metafunction is included for compatibility with `tuple`. It returns either the type of a pair's `first_type` or `second_type` in its nested type, `type`.

This metafunction returns the type of a `tuple`'s *N*th element.

Template Parameters

- N: This parameter selects the member of interest.
- T: A `pair` type of interest.

See [*pair*](#)

See [*tuple*](#)

Template Parameters

- N: This parameter selects the element of interest.
- T: A `tuple` type of interest.

Public Types

typedef tuple_element<N - 1, Next>::type **type**

The result of this metafunction is returned in `type`.

Template Struct tuple_size

- Defined in `file_thrust_pair.h`

Struct Documentation

template<class **T**>

struct tuple_size

This convenience metafunction is included for compatibility with `tuple`. It returns 2, the number of elements of a `pair`, in its nested data member, `value`.

This metafunction returns the number of elements of a `tuple` type of interest.

Template Parameters

- Pair: A `pair` type of interest.

See [*pair*](#)

See [*tuple*](#)

Template Parameters

- T: A `tuple` type of interest.

Public Static Attributes

const int **value** = 1 + tuple_size<typename T::tail_type>::value

The result of this metafunction is returned in `value`.

Template Struct unary_function

- Defined in `file_thrust_functional.h`

Struct Documentation

template<typename **Argument**, typename **Result**>

struct unary_function

unary_function is an empty base class: it contains no member functions or member variables, but only type information. The only reason it exists is to make it more convenient to define types that are models of the concept Adaptable Unary Function. Specifically, any model of Adaptable Unary Function must define nested typedefs. Those typedefs are provided by the base class *unary_function*.

The following code snippet demonstrates how to construct an Adaptable Unary Function using *unary_function*.

```
struct sine : public thrust::unary_function<float, float>
{
    __host__ __device__
    float operator() (float x) { return sinf(x); }
};
```

Note Because C++11 language support makes the functionality of *unary_function* obsolete, its use is optional if C++11 language features are enabled.

See http://www.sgi.com/tech/stl/unary_function.html

See *binary_function*

Public Types

typedef argument_type

The type of the function object's argument.

typedef result_type

The type of the function object's result.

Template Struct unary_negate

- Defined in file_thrust_functional.h

Inheritance Relationships

Base Type

- public thrust::unary_function< Predicate::argument_type, bool > *(Template Struct unary_function)*

Struct Documentation

template<typename **Predicate**>

struct unary_negate : public thrust::unary_function<*Predicate*::argument_type, bool>

unary_negate is a function object adaptor: it is an Adaptable Predicate that represents the logical negation of some other Adaptable Predicate. That is: if *f* is an object of class *unary_negate*<AdaptablePredicate>, then there exists an object *pred* of class

`AdaptablePredicate` such that `f(x)` always returns the same value as `!pred(x)`. There is rarely any reason to construct a *`unary_negate`* directly; it is almost always easier to use the helper function `not1`.

See http://www.sgi.com/tech/stl/unary_negate.html

See *`not1`*

Public Functions

`__host__ __device__ thrust::unary_negate::unary_negate(Predicate p)`

Constructor takes a `Predicate` object to negate.

Parameters

- `p`: The `Predicate` object to negate.

`__host__ __device__ bool thrust::unary_negate::operator()(const typename Predicate::argument_type& x) const`

Function call operator. The return value is `!pred(x)`.

Template Struct `unary_traits`

- Defined in `file_thrust_functional.h`

Struct Documentation

```
template<typename Operation>
struct unary_traits
```

Template Class `device_allocator`

- Defined in `file_thrust_device_allocator.h`

Nested Relationships

Nested Types

- *Template Struct `device_allocator::rebind`*

Inheritance Relationships

Base Type

- `public thrust::device_new_allocator< T > (Template Class device_new_allocator)`

Class Documentation

template<typename T>

class device_allocator : public thrust::device_new_allocator<T>

device_allocator is a device memory allocator. This implementation inherits from *device_new_allocator*.

See *device_ptr*

See *device_new_allocator*

See <http://www.sgi.com/tech/stl/Allocators.html>

Public Functions

__host__ __device__ thrust::device_allocator::device_allocator()

No-argument constructor has no effect.

__host__ __device__ thrust::device_allocator::device_allocator(device_allocator const&)

Copy constructor has no effect.

template<typename U> __host__ __device__ thrust::device_allocator::device_allocator(device_allocator const& other)

Constructor from other allocator has no effect.

template<typename U>

struct rebind

The rebind metafunction provides the type of a *device_allocator* instantiated with another type.

Template Parameters

- U: The other type to use for instantiation.

Public Types

template<>

typedef device_allocator<U> other

The typedef *other* gives the type of the rebound *device_allocator*.

Template Class device_allocator< void >

- Defined in file_thrust_device_allocator.h

Nested Relationships

Nested Types

- *Template Struct device_allocator< void >::device_allocator< void >*

Class Documentation

template<>

class `device_allocator<void>`

device_allocator<void> is a device memory allocator. This class is a specialization for `void`.

See *device_ptr*

See <http://www.sgi.com/tech/stl/Allocators.html>

Public Types

typedef `void` **value_type**

Type of element allocated, `void`.

typedef *device_ptr*<`void`> **pointer**

Pointer to allocation, *device_ptr*<`void`>.

typedef *device_ptr*<**const** `void`> **const_pointer**

const pointer to allocation, *device_ptr*<**const** `void`>.

typedef `std::size_t` **size_type**

Type of allocation size, `std::size_t`.

typedef *pointer*::difference_type **difference_type**

Type of allocation difference, *pointer*::difference_type.

template<typename `U`>

struct **rebind**

The `rebind` metafunction provides the type of a *device_allocator* instantiated with another type.

Template Parameters

- `U`: The other type to use for instantiation.

Public Types

template<>

typedef `device_allocator<U>` **other**

The typedef `other` gives the type of the rebound *device_allocator*.

Template Class `device_malloc_allocator`

- Defined in `file_thrust_device_malloc_allocator.h`

Nested Relationships

Nested Types

- *Template Struct* `device_malloc_allocator::rebind`

Class Documentation

template<typename T>

class device_malloc_allocator

device_malloc_allocator is a device memory allocator that employs the `device_malloc` function for allocation.

See *device_malloc*

See *device_ptr*

See <http://www.sgi.com/tech/stl/Allocators.html>

Public Types

typedef T value_type

Type of element allocated, T.

typedef device_ptr<T> pointer

Pointer to allocation, `device_ptr<T>`.

typedef device_ptr<const T> const_pointer

const pointer to allocation, `device_ptr<const T>`.

typedef device_reference<T> reference

Reference to allocated element, `device_reference<T>`.

typedef device_reference<const T> const_reference

const reference to allocated element, `device_reference<const T>`.

typedef std::size_t size_type

Type of allocation size, `std::size_t`.

typedef pointer::difference_type difference_type

Type of allocation difference, `pointer::difference_type`.

Public Functions

__host__ __device__ thrust::device_malloc_allocator::device_malloc_allocator()

No-argument constructor has no effect.

__host__ __device__ thrust::device_malloc_allocator::~~device_malloc_allocator()

No-argument destructor has no effect.

__host__ __device__ thrust::device_malloc_allocator::device_malloc_allocator(device_ma

Copy constructor has no effect.

template<typename U> __host__ __device__ thrust::device_malloc_allocator::device_malloc

Constructor from other *device_malloc_allocator* has no effect.

__host__ __device__ pointer thrust::device_malloc_allocator::address(reference r)

Returns the address of an allocated object.

Return &r.

__host__ __device__ const_pointer thrust::device_malloc_allocator::address(const_refer

Returns the address an allocated object.

Return &r.

`__host__ pointer thrust::device_malloc_allocator::allocate(size_type cnt, const_pointer`

Allocates storage for `cnt` objects.

Return A `pointer` to uninitialized storage for `cnt` objects.

Note Memory allocated by this function must be deallocated with `deallocate`.

Parameters

- `cnt`: The number of objects to allocate.

`__host__ void thrust::device_malloc_allocator::deallocate(pointer p, size_type cnt)`

Deallocates storage for objects allocated with `allocate`.

Note Memory deallocated by this function must previously have been allocated with `allocate`.

Parameters

- `p`: A `pointer` to the storage to deallocate.
- `cnt`: The size of the previous allocation.

`size_type max_size() const`

Returns the largest value `n` for which `allocate(n)` might succeed.

Return The largest value `n` for which `allocate(n)` might succeed.

`__host__ __device__ bool thrust::device_malloc_allocator::operator==(device_malloc_all`

Compares against another `device_malloc_allocator` for equality.

Return `true`

`__host__ __device__ bool thrust::device_malloc_allocator::operator!=(device_malloc_all`

Compares against another `device_malloc_allocator` for inequality.

Return `false`

`template<typename U>`

`struct rebind`

The `rebind` metafunction provides the type of a `device_malloc_allocator` instantiated with another type.

Template Parameters

- `U`: The other type to use for instantiation.

Public Types

`template<>`

`typedef device_malloc_allocator<U> other`

The typedef `other` gives the type of the rebound `device_malloc_allocator`.

Template Class `device_new_allocator`

- Defined in `file_thrust_device_new_allocator.h`

Nested Relationships

Nested Types

- *Template Struct `device_new_allocator::rebind`*

Inheritance Relationships

Derived Type

- `public thrust::device_allocator< T >` (*Template Class `device_allocator`*)

Class Documentation

template<typename T>

class `device_new_allocator`

device_new_allocator is a device memory allocator that employs the `device_new` function for allocation.

See *device_new*

See *device_ptr*

See <http://www.sgi.com/tech/stl/Allocators.html>

Subclassed by *thrust::device_allocator< T >*

Public Types

typedef T value_type

Type of element allocated, T.

typedef *device_ptr*<T> pointer

Pointer to allocation, *device_ptr*<T>.

typedef *device_ptr*<const T> const_pointer

const pointer to allocation, *device_ptr*<const T>.

typedef *device_reference*<T> reference

Reference to allocated element, *device_reference*<T>.

typedef *device_reference*<const T> const_reference

const reference to allocated element, *device_reference*<const T>.

typedef std::size_t size_type

Type of allocation size, `std::size_t`.

typedef *pointer*::difference_type difference_type

Type of allocation difference, *pointer*::*difference_type*.

Public Functions

`__host__ __device__ thrust::device_new_allocator::device_new_allocator()`

No-argument constructor has no effect.

`__host__ __device__ thrust::device_new_allocator::~~device_new_allocator()`

No-argument destructor has no effect.

`__host__ __device__ thrust::device_new_allocator::device_new_allocator(device_new_allocator const&)`

Copy constructor has no effect.

`template<typename U> __host__ __device__ thrust::device_new_allocator::device_new_allocator(device_malloc_allocator const&)`

Constructor from other *device_malloc_allocator* has no effect.

`__host__ __device__ pointer thrust::device_new_allocator::address(reference r)`

Returns the address of an allocated object.

Return `&r`.

`__host__ __device__ const_pointer thrust::device_new_allocator::address(const_reference r)`

Returns the address an allocated object.

Return `&r`.

`__host__ pointer thrust::device_new_allocator::allocate(size_type cnt, const_pointer = 0)`

Allocates storage for `cnt` objects.

Return A `pointer` to uninitialized storage for `cnt` objects.

Note Memory allocated by this function must be deallocated with `deallocate`.

Parameters

- `cnt`: The number of objects to allocate.

`__host__ void thrust::device_new_allocator::deallocate(pointer p, size_type cnt)`

Deallocates storage for objects allocated with `allocate`.

Note Memory deallocated by this function must previously have been allocated with `allocate`.

Parameters

- `p`: A `pointer` to the storage to deallocate.
- `cnt`: The size of the previous allocation.

`__host__ __device__ size_type thrust::device_new_allocator::max_size() const`

Returns the largest value `n` for which `allocate(n)` might succeed.

Return The largest value `n` for which `allocate(n)` might succeed.

`__host__ __device__ bool thrust::device_new_allocator::operator==(device_new_allocator const&)`

Compares against another *device_malloc_allocator* for equality.

Return `true`

`__host__ __device__ bool thrust::device_new_allocator::operator!=(device_new_allocator const&)`

Compares against another *device_malloc_allocator* for inequality.

Return `false`

`template<typename U>`

`struct rebind`

The `rebind` metafunction provides the type of a *device_new_allocator* instantiated with another type.

Template Parameters

- `U`: The other type to use for instantiation.

Public Types

```
template<>
```

```
typedef device_new_allocator<U> other
```

The typedef `other` gives the type of the rebound `device_new_allocator`.

Template Class `device_ptr`

- Defined in `file_thrust_device_malloc_allocator.h`

Inheritance Relationships

Base Type

- `public thrust::pointer< T, thrust::device_system_tag, thrust::device_reference< T >, thrust::device_ptr< T > >`

Class Documentation

```
template<typename T>
```

```
class device_ptr : public thrust::pointer<T, thrust::device_system_tag, thrust::device_reference<T>, thrust::device_ptr<T>>
```

device_ptr stores a pointer to an object allocated in device memory. This type provides type safety when dispatching standard algorithms on ranges resident in device memory.

device_ptr has pointer semantics: it may be dereferenced safely from the host and may be manipulated with pointer arithmetic.

device_ptr can be created with the functions `device_malloc`, `device_new`, or `device_pointer_cast`, or by explicitly calling its constructor with a raw pointer.

The raw pointer encapsulated by a *device_ptr* may be obtained by either its `get` method or the `raw_pointer_cast` free function.

Note *device_ptr* is not a smart pointer; it is the programmer's responsibility to deallocate memory pointed to by *device_ptr*.

See *device_malloc*

See *device_new*

See *device_pointer_cast*

See *raw_pointer_cast*

Public Functions

```
__host__ __device__ thrust::device_ptr::device_ptr()
```

device_ptr's null constructor initializes its raw pointer to 0.

```
template<typename OtherT>__host__ __device__ thrust::device_ptr::device_ptr(OtherT * p
    device_ptr's copy constructor is templated to allow copying to a device_ptr<const T> from a
    T *.
```

Parameters

- *ptr*: A raw pointer to copy from, presumed to point to a location in device memory.

```
__host__ __device__ thrust::device_ptr::device_ptr(T * ptr)
```

```
template<typename OtherT>__host__ __device__ thrust::device_ptr::device_ptr(const dev
    device_ptr's copy constructor allows copying from another device_ptr with related type.
```

Parameters

- *other*: The *device_ptr* to copy from.

```
template<typename OtherT>__host__ __device__ device_ptr& thrust::device_ptr::operator=
    device_ptr's assignment operator allows assigning from another device_ptr with related type.
```

Return **this*

Parameters

- *other*: The other *device_ptr* to copy from.

Template Class *device_reference*

- Defined in file *thrust_device_ptr.h*

Inheritance Relationships

Base Type

- public `thrust::reference< T, thrust::device_ptr< T >, thrust::device_reference< T >>`

Class Documentation

```
template<typename T>
```

```
class device_reference : public thrust::reference<T, thrust::device_ptr<T>, thrust::device_reference<T>>
    device_reference acts as a reference-like object to an object stored in device memory.
    device_reference is not intended to be used directly; rather, this type is the result of deferencing a
    device_ptr. Similarly, taking the address of a device_reference yields a device_ptr.
```

device_reference may often be used from host code in place of operations defined on its associated *value_type*. For example, when *device_reference* refers to an arithmetic type, arithmetic operations on it are legal:

```
#include <thrust/device_vector.h>

int main(void)
{
    thrust::device_vector<int> vec(1, 13);

    thrust::device_reference<int> ref_to_thirteen = vec[0];
```

(continues on next page)

(continued from previous page)

```

    int x = ref_to_thirteen + 1;

    // x is 14

    return 0;
}

```

Similarly, we can print the value of `ref_to_thirteen` in the above code by using an `iostream`:

```

#include <thrust/device_vector.h>
#include <iostream>

int main(void)
{
    thrust::device_vector<int> vec(1, 13);

    thrust::device_reference<int> ref_to_thirteen = vec[0];

    std::cout << ref_to_thirteen << std::endl;

    // 13 is printed

    return 0;
}

```

Of course, we needn't explicitly create a `device_reference` in the previous example, because one is returned by `device_vector`'s bracket operator. A more natural way to print the value of a `device_vector` element might be:

```

#include <thrust/device_vector.h>
#include <iostream>

int main(void)
{
    thrust::device_vector<int> vec(1, 13);

    std::cout << vec[0] << std::endl;

    // 13 is printed

    return 0;
}

```

These kinds of operations should be used sparingly in performance-critical code, because they imply a potentially expensive copy between host and device space.

Some operations which are possible with regular objects are impossible with their corresponding `device_reference` objects due to the requirements of the C++ language. For example, because the member access operator cannot be overloaded, member variables and functions of a referent object cannot be directly accessed through its `device_reference`.

The following code, which generates a compiler error, illustrates:

```

#include <thrust/device_vector.h>

struct foo

```

(continues on next page)

(continued from previous page)

```
{
    int x;
};

int main(void)
{
    thrust::device_vector<foo> foo_vec(1);

    thrust::device_reference<foo> foo_ref = foo_vec[0];

    foo_ref.x = 13; // ERROR: x cannot be accessed through foo_ref

    return 0;
}
```

Instead, a host space copy must be created to access `foo`'s `x` member:

```
#include <thrust/device_vector.h>

struct foo
{
    int x;
};

int main(void)
{
    thrust::device_vector<foo> foo_vec(1);

    // create a local host-side foo object
    foo host_foo;
    host_foo.x = 13;

    thrust::device_reference<foo> foo_ref = foo_vec[0];

    foo_ref = host_foo;

    // foo_ref's x member is 13

    return 0;
}
```

Another common case where a `device_reference` cannot directly be used in place of its referent object occurs when passing them as parameters to functions like `printf` which have varargs parameters. Because varargs parameters must be Plain Old Data, a `device_reference` to a POD type requires a cast when passed to `printf`:

```
#include <stdio.h>
#include <thrust/device_vector.h>

int main(void)
{
    thrust::device_vector<int> vec(1,13);

    // vec[0] must be cast to int when passing to printf
    printf("%d\n", (int) vec[0]);
}
```

(continues on next page)

(continued from previous page)

```

    return 0;
}

```

See *device_ptr*

See *device_vector*

Public Types

typedef super_t::value_type **value_type**

The type of the value referenced by this type of *device_reference*.

typedef super_t::pointer **pointer**

The type of the expression &ref, where ref is a *device_reference*.

Public Functions

template<typename OtherT>__host__ __device__ thrust::device_reference::device_reference

This copy constructor accepts a const reference to another *device_reference*. After this *device_reference* is constructed, it shall refer to the same object as other.

The following code snippet demonstrates the semantics of this copy constructor.

Parameters

- other: A *device_reference* to copy from.

```

#include <thrust/device_vector.h>
#include <assert.h>
...
thrust::device_vector<int> v(1,0);
thrust::device_reference<int> ref = v[0];

// ref equals the object at v[0]
assert(ref == v[0]);

// the address of ref equals the address of v[0]
assert(&ref == &v[0]);

// modifying v[0] modifies ref
v[0] = 13;
assert(ref == 13);

```

Note This constructor is templated primarily to allow initialization of *device_reference*<const T> from *device_reference*<T>.

__host__ __device__ thrust::device_reference::device_reference(const pointer & ptr)

This copy constructor initializes this *device_reference* to refer to an object pointed to by the given *device_ptr*. After this *device_reference* is constructed, it shall refer to the object pointed to by ptr.

The following code snippet demonstrates the semantic of this copy constructor.

Parameters

- ptr: A *device_ptr* to copy from.

```
#include <thrust/device_vector.h>
#include <assert.h>
...
thrust::device_vector<int> v(1,0);
thrust::device_ptr<int> ptr = &v[0];
thrust::device_reference<int> ref(ptr);

// ref equals the object pointed to by ptr
assert(ref == *ptr);

// the address of ref equals ptr
assert(&ref == ptr);

// modifying *ptr modifies ref
*ptr = 13;
assert(ref == 13);
```

template<typename OtherT> __host__ __device__ device_reference& thrust::device_reference

This assignment operator assigns the value of the object referenced by the given *device_reference* to the object referenced by this *device_reference*.

Return *this

Parameters

- other: The *device_reference* to assign from.

__host__ __device__ device_reference& thrust::device_reference::operator=(const value

Assignment operator assigns the value of the given value to the value referenced by this *device_reference*.

Return *this

Parameters

- x: The value to assign from.

Template Class device_vector

- Defined in file_thrust_device_vector.h

Inheritance Relationships

Base Type

- public detail::vector_base< T, Alloc >

Class Documentation

template<typename T, typename Alloc = thrust::device_malloc_allocator<T>>

class device_vector : public detail::vector_base<T, Alloc>

A *device_vector* is a container that supports random access to elements, constant time removal of elements at the end, and linear time insertion and removal of elements at the beginning or in the middle. The number

of elements in a *device_vector* may vary dynamically; memory management is automatic. The memory associated with a *device_vector* resides in the memory space of a parallel device.

See <http://www.sgi.com/tech/stl/Vector.html>

See *host_vector*

Public Functions

`__host__ device_vector (void)`

This constructor creates an empty *device_vector*.

`__host__ ~device_vector (void)`

The destructor erases the elements.

`__host__ device_vector (size_type n)`

This constructor creates a *device_vector* with the given size.

Parameters

- *n*: The number of elements to initially create.

`__host__ device_vector (size_type n, const value_type &value)`

This constructor creates a *device_vector* with copies of an exemplar element.

Parameters

- *n*: The number of elements to initially create.
- *value*: An element to copy.

`__host__ device_vector (const device_vector &v)`

Copy constructor copies from an exemplar *device_vector*.

Parameters

- *v*: The *device_vector* to copy.

`__host__ device_vector& thrust::device_vector::operator=(const device_vector & v)`

Copy assign operator copies another *device_vector* with the same type.

Parameters

- *v*: The *device_vector* to copy.

`template<typename OtherT, typename OtherAlloc> __host__ __device__ thrust::device_vector`

Copy constructor copies from an exemplar *device_vector* with different type.

Parameters

- *v*: The *device_vector* to copy.

`template<typename OtherT, typename OtherAlloc> __host__ device_vector& thrust::device_v`

Assign operator copies from an exemplar *device_vector* with different type.

Parameters

- *v*: The *device_vector* to copy.

`template<typename OtherT, typename OtherAlloc>`

`__host__ device_vector (const std::vector<OtherT, OtherAlloc> &v)`

Copy constructor copies from an exemplar `std::vector`.

Parameters

- `v`: The `std::vector` to copy.

```
template<typename OtherT, typename OtherAlloc>__host__ device_vector& thrust::device_vector<OtherT, OtherAlloc>::operator=(const host_vector<OtherT, OtherAlloc>& v)
```

Assign operator copies from an exemplar `std::vector`.

Parameters

- `v`: The `std::vector` to copy.

```
template<typename OtherT, typename OtherAlloc>__host__ device_vector<OtherT, OtherAlloc>::device_vector(const host_vector<OtherT, OtherAlloc>& v)
```

Copy constructor copies from an exemplar `host_vector` with possibly different type.

Parameters

- `v`: The `host_vector` to copy.

```
template<typename OtherT, typename OtherAlloc>__host__ device_vector& thrust::device_vector<OtherT, OtherAlloc>::operator=(const host_vector<OtherT, OtherAlloc>& v)
```

Assign operator copies from an exemplar `host_vector`.

Parameters

- `v`: The `host_vector` to copy.

```
template<typename InputIterator>__host__ device_vector<InputIterator, InputIterator, InputIterator, InputIterator>::device_vector(InputIterator first, InputIterator last)
```

This constructor builds a `device_vector` from a range.

Parameters

- `first`: The beginning of the range.
- `last`: The end of the range.

Template Class `host_vector`

- Defined in `file_thrust_device_vector.h`

Inheritance Relationships

Base Type

- `public detail::vector_base< T, Alloc >`

Class Documentation

```
template<typename T, typename Alloc = std::allocator<T>>class host_vector : public detail::vector_base<T, Alloc>
```

A `host_vector` is a container that supports random access to elements, constant time removal of elements at the end, and linear time insertion and removal of elements at the beginning or in the middle. The number of elements in a `host_vector` may vary dynamically; memory management is automatic. The memory associated with a `host_vector` resides in the memory space of the host associated with a parallel device.

See <http://www.sgi.com/tech/stl/Vector.html>

See `device_vector`

Public Functions

`__host__ host_vector` (void)

This constructor creates an empty *host_vector*.

`__host__ ~host_vector` (void)

The destructor erases the elements.

`__host__ host_vector` (size_type *n*)

This constructor creates a *host_vector* with the given size.

Parameters

- *n*: The number of elements to initially create.

`__host__ host_vector` (size_type *n*, const value_type &*value*)

This constructor creates a *host_vector* with copies of an exemplar element.

Parameters

- *n*: The number of elements to initially create.
- *value*: An element to copy.

`__host__ host_vector` (const *host_vector* &*v*)

Copy constructor copies from an exemplar *host_vector*.

Parameters

- *v*: The *host_vector* to copy.

`__host__ host_vector& thrust::host_vector::operator=(const host_vector & v)`

Assign operator copies from an exemplar *host_vector*.

Parameters

- *v*: The *host_vector* to copy.

template<typename **OtherT**, typename **OtherAlloc**>

`__host__ host_vector` (const *host_vector*<*OtherT*, *OtherAlloc*> &*v*)

Copy constructor copies from an exemplar *host_vector* with different type.

Parameters

- *v*: The *host_vector* to copy.

template<typename **OtherT**, typename **OtherAlloc**> __host__ host_vector& thrust::host_vector::operator=(const *host_vector*<*OtherT*, *OtherAlloc*> &*v*)

Assign operator copies from an exemplar *host_vector* with different type.

Parameters

- *v*: The *host_vector* to copy.

template<typename **OtherT**, typename **OtherAlloc**>

`__host__ host_vector` (const std::vector<*OtherT*, *OtherAlloc*> &*v*)

Copy constructor copies from an exemplar std::vector.

Parameters

- *v*: The std::vector to copy.

template<typename **OtherT**, typename **OtherAlloc**> __host__ host_vector& thrust::host_vector::operator=(const std::vector<*OtherT*, *OtherAlloc*> &*v*)

Assign operator copies from an exemplar std::vector.

Parameters

- `v`: The `std::vector` to copy.

```
template<typename OtherT, typename OtherAlloc>
__host__ host_vector(const device_vector<OtherT, OtherAlloc> &v)
```

Copy constructor copies from an exemplar *device_vector* with possibly different type.

Parameters

- `v`: The *device_vector* to copy.

```
template<typename OtherT, typename OtherAlloc>__host__ host_vector& thrust::host_vector
```

Assign operator copies from an exemplar *device_vector*.

Parameters

- `v`: The *device_vector* to copy.

```
template<typename InputIterator>
__host__ host_vector(InputIterator first, InputIterator last)
```

This constructor builds a *host_vector* from a range.

Parameters

- `first`: The beginning of the range.
- `last`: The end of the range.

Template Class tuple

- Defined in `file_thrust_tuple.h`

Inheritance Relationships

Base Type

- `public detail::map_tuple_to_cons::type< T0, T1, T2, T3, T4, T5, T6, T7, T8, T9 >`

Class Documentation

```
template<class T0, class T1, class T2, class T3, class T4, class T5, class T6, class T7, class T8, class T9>
```

```
class tuple: public detail::map_tuple_to_cons::type<T0, T1, T2, T3, T4, T5, T6, T7, T8, T9>
```

`tuple` is a class template that can be instantiated with up to ten arguments. Each template argument specifies the type of element in the `tuple`. Consequently, tuples are heterogeneous, fixed-size collections of values. An instantiation of `tuple` with two arguments is similar to an instantiation of `pair` with the same two arguments. Individual elements of a `tuple` may be accessed with the `get` function.

The following code snippet demonstrates how to create a new `tuple` object and inspect and modify the value of its elements.

Template Parameters

- `TN`: The type of the `N` `tuple` element. Thrust's `tuple` type currently supports up to ten elements.

```

#include <thrust/tuple.h>
#include <iostream>

...
// create a tuple containing an int, a float, and a string
thrust::tuple<int, float, const char*> t(13, 0.1f, "thrust");

// individual members are accessed with the free function get
std::cout << "The first element's value is " << thrust::get<0>(t) << std::endl;

// or the member function get
std::cout << "The second element's value is " << t.get<1>() << std::endl;

// we can also modify elements with the same function
thrust::get<0>(t) += 10;

```

See *pair*

See *get*

See *make_tuple*

See *tuple_element*

See *tuple_size*

See *tie*

Public Functions

__host__ __device__ thrust::tuple::tuple(void)

tuple's no-argument constructor initializes each element.

__host__ __device__ thrust::tuple::tuple(typename access_traits< T0 >::parameter_type t0)

tuple's one-argument constructor copy constructs the first element from the given parameter and initializes all other elements.

Parameters

- t0: The value to assign to this tuple's first element.

__host__ __device__ thrust::tuple::tuple(typename access_traits< T0 >::parameter_type t0, typename access_traits< T1 >::parameter_type t1)

tuple's one-argument constructor copy constructs the first two elements from the given parameters and initializes all other elements.

Note tuple's constructor has ten variants of this form, the rest of which are omitted here for brevity.

Parameters

- t0: The value to assign to this tuple's first element.
- t1: The value to assign to this tuple's second element.

template<class U1, class U2>__host__ __device__ tuple& thrust::tuple::operator=(const pair<U1, U2>& k)

This assignment operator allows assigning the first two elements of this tuple from a pair.

Parameters

- k: A pair to assign from.

__host__ __device__ void thrust::tuple::swap(tuple & t)

swap swaps the elements of two tuples.

Parameters

- `t`: The other `tuple` with which to swap.

1.3.3 Functions

Template Function `thrust::abs`

- Defined in `file_thrust_complex.h`

Function Documentation

```
template<typename T>__host__ __device__ T thrust::abs(const complex < T > & z)
```

Returns the magnitude (also known as absolute value) of a `complex`.

Parameters

- `z`: The `complex` from which to calculate the absolute value.

Template Function `thrust::acos`

- Defined in `file_thrust_complex.h`

Function Documentation

```
template<typename T>__host__ __device__ complex<T> thrust::acos(const complex < T > & z)
```

Returns the complex arc cosine of a `complex` number.

The range of the real part of the result is $[0, \pi]$ and the range of the imaginary part is $[-\infty, +\infty]$

Parameters

- `z`: The `complex` argument.

Template Function `thrust::acosh`

- Defined in `file_thrust_complex.h`

Function Documentation

```
template<typename T>__host__ __device__ complex<T> thrust::acosh(const complex < T > & z)
```

Returns the complex inverse hyperbolic cosine of a `complex` number.

The range of the real part of the result is $[0, +\infty]$ and the range of the imaginary part is $[-\pi, \pi]$

Parameters

- `z`: The `complex` argument.

Template Function `thrust::adjacent_difference(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator)`

- Defined in file `thrust_adjacent_difference.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::adjacent_difference`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
↳typename BinaryFunction>__host__ __device__ OutputIterator thrust::adjacent_
↳difference(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳InputIterator, InputIterator, OutputIterator, BinaryFunction)
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator>_
↳__host__ __device__ OutputIterator thrust::adjacent_difference(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
↳InputIterator, OutputIterator)
- template<typename InputIterator, typename OutputIterator, typename BinaryFunction>
  OutputIterator thrust::adjacent_difference(InputIterator, InputIterator,
↳OutputIterator, BinaryFunction)
- template<typename InputIterator, typename OutputIterator>
  OutputIterator thrust::adjacent_difference(InputIterator, InputIterator,
↳OutputIterator)
```

Template Function `thrust::adjacent_difference(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, BinaryFunction)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::adjacent_difference`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, BinaryFunction)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
↳typename BinaryFunction>__host__ __device__ OutputIterator thrust::adjacent_
↳difference(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳InputIterator, InputIterator, OutputIterator, BinaryFunction)
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator>_
↳__host__ __device__ OutputIterator thrust::adjacent_difference(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
↳InputIterator, OutputIterator)
- template<typename InputIterator, typename OutputIterator, typename BinaryFunction>
  OutputIterator thrust::adjacent_difference(InputIterator, InputIterator,
↳OutputIterator, BinaryFunction)
- template<typename InputIterator, typename OutputIterator>
  OutputIterator thrust::adjacent_difference(InputIterator, InputIterator,
↳OutputIterator)
```

Template Function `thrust::adjacent_difference(InputIterator, InputIterator, OutputIterator)`

Function Documentation

```
template<typename InputIterator, typename OutputIterator>  
OutputIterator thrust::adjacent_difference(InputIterator first, InputIterator last, OutputIterator  
                                           result)
```

`adjacent_difference` calculates the differences of adjacent elements in the range `[first, last)`. That is, `*first` is assigned to `*result`, and, for each iterator `i` in the range `[first + 1, last)`, the difference of `*i` and `*(i - 1)` is assigned to `*(result + (i - first))`.

This version of `adjacent_difference` uses `operator-` to calculate differences.

The following code snippet demonstrates how to use `adjacent_difference` to compute the difference between adjacent elements of a range.

Return The iterator `result + (last - first)`

Remark Note that `result` is permitted to be the same iterator as `first`. This is useful for computing differences “in place”.

Parameters

- `first`: The beginning of the input range.
- `last`: The end of the input range.
- `result`: The beginning of the output range.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#), and `x` and `y` are objects of `InputIterator`'s `value_type`, then `x - y` is defined, and `InputIterator`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`, and the return type of `x - y` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `OutputIterator`: is a model of [Output Iterator](#).

```
#include <thrust/adjacent_difference.h>  
#include <thrust/device_vector.h>  
...  
int h_data[8] = {1, 2, 1, 2, 1, 2, 1, 2};  
thrust::device_vector<int> d_data(h_data, h_data + 8);  
thrust::device_vector<int> d_result(8);  
  
thrust::adjacent_difference(d_data.begin(), d_data.end(), d_result.begin());  
  
// d_result is now [1, 1, -1, 1, -1, 1, -1, 1]
```

See http://www.sgi.com/tech/stl/adjacent_difference.html

See `inclusive_scan`

Template Function `thrust::adjacent_difference(InputIterator, InputIterator, OutputIterator, BinaryFunction)`

Function Documentation

```
template<typename InputIterator, typename OutputIterator, typename BinaryFunction>
```

OutputIterator thrust::adjacent_difference(*InputIterator* first, *InputIterator* last, *OutputIterator* result, *BinaryFunction* binary_op)

adjacent_difference calculates the differences of adjacent elements in the range [first, last). That is, *first is assigned to *result, and, for each iterator i in the range [first + 1, last), binary_op(*i, *(i - 1)) is assigned to *(result + (i - first)).

This version of adjacent_difference uses the binary function binary_op to calculate differences.

The following code snippet demonstrates how to use adjacent_difference to compute the sum between adjacent elements of a range.

Return The iterator result + (last - first)

Remark Note that result is permitted to be the same iterator as first. This is useful for computing differences “in place”.

Parameters

- first: The beginning of the input range.
- last: The end of the input range.
- result: The beginning of the output range.
- binary_op: The binary function used to compute differences.

Template Parameters

- InputIterator: is a model of [Input Iterator](#), and InputIterator's value_type is convertible to BinaryFunction's first_argument_type and second_argument_type, and InputIterator's value_type is convertible to a type in OutputIterator's set of value_types.
- OutputIterator: is a model of [Output Iterator](#).
- BinaryFunction's: result_type is convertible to a type in OutputIterator's set of value_types.

```
#include <thrust/adjacent_difference.h>
#include <thrust/functional.h>
#include <thrust/device_vector.h>
...
int h_data[8] = {1, 2, 1, 2, 1, 2, 1, 2};
thrust::device_vector<int> d_data(h_data, h_data + 8);
thrust::device_vector<int> d_result(8);

thrust::adjacent_difference(d_data.begin(), d_data.end(), d_result.begin(),
    thrust::plus<int>());

// d_result is now [1, 3, 3, 3, 3, 3, 3, 3]
```

See http://www.sgi.com/tech/stl/adjacent_difference.html

See [inclusive_scan](#)

Template Function thrust::advance

- Defined in file thrust_advance.h

Function Documentation

template<typename InputIterator, typename Distance>__host__ __device__ void thrust::advance

`advance(i, n)` increments the iterator `i` by the distance `n`. If `n > 0` it is equivalent to executing `++i n` times, and if `n < 0` it is equivalent to executing `i n` times. If `n == 0`, the call has no effect.

The following code snippet demonstrates how to use `advance` to increment an iterator a given number of times.

Pre `n` shall be negative only for bidirectional and random access iterators.

Parameters

- `i`: The iterator to be advanced.
- `n`: The distance by which to advance the iterator.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#).
- `Distance`: is an integral type that is convertible to `InputIterator`'s distance type.

```
#include <thrust/advance.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> vec(13);
thrust::device_vector<int>::iterator iter = vec.begin();

thrust::advance(iter, 7);

// iter - vec.begin() == 7
```

See <http://www.sgi.com/tech/stl/advance.html>

Template Function `thrust::all_of(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`

- Defined in `file_thrust_logical.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::all_of`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename Predicate>__
  ↪__host__ __device__ bool thrust::all_of(const thrust::detail::execution_policy_base
  ↪< DerivedPolicy > &, InputIterator, InputIterator, Predicate)
- template<typename InputIterator, typename Predicate>
  bool thrust::all_of(InputIterator, InputIterator, Predicate)
```

Template Function `thrust::all_of(InputIterator, InputIterator, Predicate)`

Function Documentation

template<typename **InputIterator**, typename **Predicate**>

bool thrust::all_of(*InputIterator first, InputIterator last, Predicate pred*)

all_of determines whether all elements in a range satisfy a predicate. Specifically, all_of returns true if pred(*i) is true for every iterator i in the range [first, last) and false otherwise.

```
#include <thrust/logical.h>
#include <thrust/functional.h>
...
bool A[3] = {true, true, false};

thrust::all_of(A, A + 2, thrust::identity<bool>()); // returns true
thrust::all_of(A, A + 3, thrust::identity<bool>()); // returns false

// empty range
thrust::all_of(A, A, thrust::identity<bool>()); // returns false
```

Return true, if all elements satisfy the predicate; false, otherwise.

Parameters

- first: The beginning of the sequence.
- last: The end of the sequence.
- pred: A predicate used to test range elements.

Template Parameters

- InputIterator: is a model of [Input Iterator](#),
- Predicate: must be a model of [Predicate](#).

See any_of

See none_of

See transform_reduce

Template Function thrust::any_of(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)

- Defined in file_thrust_logical.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::any_of” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename Predicate>__
  ↪ host __device__ bool thrust::any_of(const thrust::detail::execution_policy_base
  ↪ DerivedPolicy &, InputIterator, InputIterator, Predicate)
- template<typename InputIterator, typename Predicate>
  bool thrust::any_of(InputIterator, InputIterator, Predicate)
```

Template Function `thrust::any_of(InputIterator, InputIterator, Predicate)`

Function Documentation

```
template<typename InputIterator, typename Predicate>
```

```
bool thrust::any_of(InputIterator first, InputIterator last, Predicate pred)
```

`any_of` determines whether any element in a range satisfies a predicate. Specifically, `any_of` returns true if `pred(*i)` is true for any iterator `i` in the range `[first, last)` and false otherwise.

```
#include <thrust/logical.h>
#include <thrust/functional.h>
...
bool A[3] = {true, true, false};

thrust::any_of(A, A + 2, thrust::identity<bool>()); // returns true
thrust::any_of(A, A + 3, thrust::identity<bool>()); // returns true

thrust::any_of(A + 2, A + 3, thrust::identity<bool>()); // returns false

// empty range
thrust::any_of(A, A, thrust::identity<bool>()); // returns false
```

Return true, if any element satisfies the predicate; false, otherwise.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.
- `pred`: A predicate used to test range elements.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#),
- `Predicate`: must be a model of [Predicate](#).

See `all_of`

See `none_of`

See `transform_reduce`

Template Function `thrust::arg`

- Defined in `file_thrust_complex.h`

Function Documentation

```
template<typename T>__host__ __device__ T thrust::arg(const complex < T > & z)
```

Returns the phase angle (also known as argument) in radians of a complex.

Parameters

- `z`: The complex from which to calculate the phase angle.

Template Function `thrust::asin`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::asin(const complex < T > & z)

Returns the complex arc sine of a complex number.

The range of the real part of the result is $[-\pi/2, \pi/2]$ and the range of the imaginary part is $[-\text{inf}, +\text{inf}]$

Parameters

- `z`: The complex argument.

Template Function `thrust::asinh`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::asinh(const complex < T > & z)

Returns the complex inverse hyperbolic sine of a complex number.

The range of the real part of the result is $[-\text{inf}, +\text{inf}]$ and the range of the imaginary part is $[-\pi/2, \pi/2]$

Parameters

- `z`: The complex argument.

Template Function `thrust::atan`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::atan(const complex < T > & z)

Returns the complex arc tangent of a complex number.

The range of the real part of the result is $[-\pi/2, \pi/2]$ and the range of the imaginary part is $[-\text{inf}, +\text{inf}]$

Parameters

- `z`: The complex argument.

Template Function `thrust::atanh`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::atanh(const complex < T > & z)

Returns the complex inverse hyperbolic tangent of a complex number.

The range of the real part of the result is $[-\infty, +\infty]$ and the range of the imaginary part is $[-\pi/2, \pi/2]$

Parameters

- z: The complex argument.

Template Function thrust::binary_search(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const LessThanComparable&)

- Defined in file_thrust_binary_search.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::binary_search” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const LessThanComparable&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<class ForwardIterator, class InputIterator, class OutputIterator, class
↳ StrictWeakOrdering>
  OutputIterator thrust::binary_search(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)
- template<class ForwardIterator, class InputIterator, class OutputIterator>
  OutputIterator thrust::binary_search(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator)
- template<class ForwardIterator, class LessThanComparable>
  bool thrust::binary_search(ForwardIterator, ForwardIterator, const
↳ LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  bool thrust::binary_search(ForwardIterator, ForwardIterator, const T&,
↳ StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator, typename StrictWeakOrdering>__host__ __device__
↳ OutputIterator thrust::binary_search(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, ForwardIterator, ForwardIterator, InputIterator, InputIterator,
↳ OutputIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::binary_
↳ search(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳ LessThanComparable>__host__ __device__ bool thrust::binary_search(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
↳ StrictWeakOrdering>__host__ __device__ bool thrust::binary_search(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const T &, StrictWeakOrdering)
```

Template Function `thrust::binary_search(ForwardIterator, ForwardIterator, const LessThanComparable&)`

Function Documentation

template<class **ForwardIterator**, class **LessThanComparable**>

bool thrust::binary_search(ForwardIterator first, ForwardIterator last, const LessThanComparable &value)

`binary_search` is a version of binary search: it attempts to find the element value in an ordered range `[first, last)`. It returns `true` if an element that is equivalent to `value` is present in `[first, last)` and `false` if no such element exists. Specifically, this version returns `true` if and only if there exists an iterator `i` in `[first, last)` such that `*i < value` and `value < *i` are both false.

The following code snippet demonstrates how to use `binary_search` to search for values in a ordered range.

Return `true` if an equivalent element exists in `[first, last)`, otherwise `false`.

Parameters

- `first`: The beginning of the ordered sequence.
- `last`: The end of the ordered sequence.
- `value`: The value to be searched.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#).
- `LessThanComparable`: is a model of [LessThanComparable](#).

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::binary_search(input.begin(), input.end(), 0); // returns true
thrust::binary_search(input.begin(), input.end(), 1); // returns false
thrust::binary_search(input.begin(), input.end(), 2); // returns true
thrust::binary_search(input.begin(), input.end(), 3); // returns false
thrust::binary_search(input.begin(), input.end(), 8); // returns true
thrust::binary_search(input.begin(), input.end(), 9); // returns false
```

See http://www.sgi.com/tech/stl/binary_search.html

See `lower_bound`

See `upper_bound`

See `equal_range`

Template Function `thrust::binary_search(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::binary_search” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<class ForwardIterator, class InputIterator, class OutputIterator, class
↳StrictWeakOrdering>
  OutputIterator thrust::binary_search(ForwardIterator, ForwardIterator,
↳InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)
- template<class ForwardIterator, class InputIterator, class OutputIterator>
  OutputIterator thrust::binary_search(ForwardIterator, ForwardIterator,
↳InputIterator, InputIterator, OutputIterator)
- template<class ForwardIterator, class LessThanComparable>
  bool thrust::binary_search(ForwardIterator, ForwardIterator, const
↳LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  bool thrust::binary_search(ForwardIterator, ForwardIterator, const T&,
↳StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳typename OutputIterator, typename StrictWeakOrdering>__host__ __device__
↳OutputIterator thrust::binary_search(const thrust::detail::execution_policy_base<
↳DerivedPolicy > &, ForwardIterator, ForwardIterator, InputIterator, InputIterator,
↳OutputIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳typename OutputIterator>__host__ __device__ OutputIterator thrust::binary_
↳search(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳LessThanComparable>__host__ __device__ bool thrust::binary_search(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
↳StrictWeakOrdering>__host__ __device__ bool thrust::binary_search(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ForwardIterator, const T &, StrictWeakOrdering)
```

Template Function thrust::binary_search(ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)

Function Documentation

```
template<class ForwardIterator, class T, class StrictWeakOrdering>
```

```
bool thrust::binary_search(ForwardIterator first, ForwardIterator last, const T &value,
  StrictWeakOrdering comp)
```

binary_search is a version of binary search: it attempts to find the element value in an ordered range [first, last). It returns true if an element that is equivalent to value is present in [first, last) and false if no such element exists. Specifically, this version returns true if and only if there exists an iterator i in [first, last) such that comp(*i, value) and comp(value, *i) are both false.

The following code snippet demonstrates how to use binary_search to search for values in a ordered range.

Return true if an equivalent element exists in [first, last), otherwise false.

Parameters

- first: The beginning of the ordered sequence.
- last: The end of the ordered sequence.
- value: The value to be searched.
- comp: The comparison operator.

Template Parameters

- ForwardIterator: is a model of [Forward Iterator](#).
- T: is comparable to ForwardIterator's value_type.
- StrictWeakOrdering: is a model of [Strict Weak Ordering](#).

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
#include <thrust/functional.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::binary_search(input.begin(), input.end(), 0, thrust::less<int>()); // returns true
thrust::binary_search(input.begin(), input.end(), 1, thrust::less<int>()); // returns false
thrust::binary_search(input.begin(), input.end(), 2, thrust::less<int>()); // returns true
thrust::binary_search(input.begin(), input.end(), 3, thrust::less<int>()); // returns false
thrust::binary_search(input.begin(), input.end(), 8, thrust::less<int>()); // returns true
thrust::binary_search(input.begin(), input.end(), 9, thrust::less<int>()); // returns false
```

See http://www.sgi.com/tech/stl/binary_search.html

See `lower_bound`

See `upper_bound`

See `equal_range`

Template Function `thrust::binary_search(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::binary_search” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<class ForwardIterator, class InputIterator, class OutputIterator, class
↳ StrictWeakOrdering>
  OutputIterator thrust::binary_search(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)
- template<class ForwardIterator, class InputIterator, class OutputIterator>
  OutputIterator thrust::binary_search(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator)
- template<class ForwardIterator, class LessThanComparable>
  bool thrust::binary_search(ForwardIterator, ForwardIterator, const
↳ LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  bool thrust::binary_search(ForwardIterator, ForwardIterator, const T&,
↳ StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator, typename StrictWeakOrdering>__host__ __device__
↳ OutputIterator thrust::binary_search(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, ForwardIterator, ForwardIterator, InputIterator, InputIterator,
↳ OutputIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::binary_
↳ search(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳ LessThanComparable>__host__ __device__ bool thrust::binary_search(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
↳ StrictWeakOrdering>__host__ __device__ bool thrust::binary_search(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const T &, StrictWeakOrdering)
```

Template Function thrust::binary_search(ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)

Function Documentation

```
template<class ForwardIterator, class InputIterator, class OutputIterator>
```

```
OutputIterator thrust::binary_search(ForwardIterator first, ForwardIterator last, InputIterator val-
ues_first, InputIterator values_last, OutputIterator result)
```

binary_search is a vectorized version of binary search: for each iterator v in [values_first, values_last) it attempts to find the value *v in an ordered range [first, last). It returns true if an element that is equivalent to value is present in [first, last) and false if no such element exists.

The following code snippet demonstrates how to use binary_search to search for multiple values in a ordered range.

Pre The ranges [first, last) and [result, result + (last - first)) shall not overlap.

Parameters

- first: The beginning of the ordered sequence.

- last: The end of the ordered sequence.
- values_first: The beginning of the search values sequence.
- values_last: The end of the search values sequence.
- result: The beginning of the output sequence.

Template Parameters

- ForwardIterator: is a model of [Forward Iterator](#).
- InputIterator: is a model of [Input Iterator](#). and InputIterator's value_type is [LessThanComparable](#).
- OutputIterator: is a model of [Output Iterator](#). and bool is convertible to OutputIterator's value_type.

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::device_vector<int> values(6);
values[0] = 0;
values[1] = 1;
values[2] = 2;
values[3] = 3;
values[4] = 8;
values[5] = 9;

thrust::device_vector<bool> output(6);

thrust::binary_search(input.begin(), input.end(),
                      values.begin(), values.end(),
                      output.begin());

// output is now [true, false, true, false, true, false]
```

See http://www.sgi.com/tech/stl/binary_search.html

See [lower_bound](#)

See [upper_bound](#)

See [equal_range](#)

Template Function `thrust::binary_search(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::binary_search” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<class ForwardIterator, class InputIterator, class OutputIterator, class
↳ StrictWeakOrdering>
  OutputIterator thrust::binary_search(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)
- template<class ForwardIterator, class InputIterator, class OutputIterator>
  OutputIterator thrust::binary_search(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator)
- template<class ForwardIterator, class LessThanComparable>
  bool thrust::binary_search(ForwardIterator, ForwardIterator, const
↳ LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  bool thrust::binary_search(ForwardIterator, ForwardIterator, const T&,
↳ StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator, typename StrictWeakOrdering>__host__ __device__
↳ OutputIterator thrust::binary_search(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, ForwardIterator, ForwardIterator, InputIterator, InputIterator,
↳ OutputIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::binary_
↳ search(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳ LessThanComparable>__host__ __device__ bool thrust::binary_search(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
↳ StrictWeakOrdering>__host__ __device__ bool thrust::binary_search(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const T &, StrictWeakOrdering)
```

Template Function thrust::binary_search(ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)

Function Documentation

```
template<class ForwardIterator, class InputIterator, class OutputIterator, class StrictWeakOrdering>
OutputIterator thrust::binary_search(ForwardIterator first, ForwardIterator last, InputIterator val-
ues_first, InputIterator values_last, OutputIterator result,
StrictWeakOrdering comp)
```

binary_search is a vectorized version of binary search: for each iterator v in [values_first, values_last) it attempts to find the value *v in an ordered range [first, last). It returns true if an element that is equivalent to value is present in [first, last) and false if no such element exists. This version of binary_search uses function object comp for comparison.

The following code snippet demonstrates how to use binary_search to search for multiple values in a ordered range.

Pre The ranges `[first, last)` and `[result, result + (last - first))` shall not overlap.

Parameters

- `first`: The beginning of the ordered sequence.
- `last`: The end of the ordered sequence.
- `values_first`: The beginning of the search values sequence.
- `values_last`: The end of the search values sequence.
- `result`: The beginning of the output sequence.
- `comp`: The comparison operator.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#).
- `InputIterator`: is a model of [Input Iterator](#). and `InputIterator`'s `value_type` is [LessThanComparable](#).
- `OutputIterator`: is a model of [Output Iterator](#). and `bool` is convertible to `OutputIterator`'s `value_type`.
- `StrictWeakOrdering`: is a model of [Strict Weak Ordering](#).

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
#include <thrust/functional.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::device_vector<int> values(6);
values[0] = 0;
values[1] = 1;
values[2] = 2;
values[3] = 3;
values[4] = 8;
values[5] = 9;

thrust::device_vector<bool> output(6);

thrust::binary_search(input.begin(), input.end(),
                      values.begin(), values.end(),
                      output.begin(),
                      thrust::less<T>());

// output is now [true, false, true, false, true, false]
```

See http://www.sgi.com/tech/stl/binary_search.html

See `lower_bound`

See `upper_bound`

See `equal_range`

Template Function `thrust::conj`

- Defined in `file_thrust_complex.h`

Function Documentation

`template<typename T>__host__ __device__ complex<T> thrust::conj(const complex < T > & z)`

Returns the complex conjugate of a `complex`.

Parameters

- `z`: The `complex` from which to calculate the complex conjugate.

Template Function `thrust::copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator)`

- Defined in `file_thrust_copy.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::copy`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator>_
  ↪__host__ __device__ OutputIterator thrust::copy(const thrust::detail::execution_
  ↪policy_base< DerivedPolicy > &, InputIterator, InputIterator, OutputIterator)
- template<typename InputIterator, typename OutputIterator>
  OutputIterator thrust::copy(InputIterator, InputIterator, OutputIterator)
```

Template Function `thrust::copy(InputIterator, InputIterator, OutputIterator)`

Function Documentation

`template<typename InputIterator, typename OutputIterator>`

OutputIterator `thrust::copy (InputIterator first, InputIterator last, OutputIterator result)`

`copy` copies elements from the range `[first, last)` to the range `[result, result + (last - first))`. That is, it performs the assignments `*result = *first`, `*(result + 1) = *(first + 1)`, and so on. Generally, for every integer `n` from 0 to `last - first`, `copy` performs the assignment `*(result + n) = *(first + n)`. Unlike `std::copy`, `copy` offers no guarantee on order of operation. As a result, calling `copy` with overlapping source and destination ranges has undefined behavior.

The return value is `result + (last - first)`.

The following code snippet demonstrates how to use `copy` to copy from one range to another.

Return The end of the destination sequence.

See <http://www.sgi.com/tech/stl/copy.html>

Pre result may be equal to first, but result shall not be in the range [first, last) otherwise.

Parameters

- first: The beginning of the sequence to copy.
- last: The end of the sequence to copy.
- result: The destination sequence.

Template Parameters

- InputIterator: must be a model of [Input Iterator](#) and InputIterator's value_type must be convertible to OutputIterator's value_type.
- OutputIterator: must be a model of [Output Iterator](#).

```
#include <thrust/copy.h>
#include <thrust/device_vector.h>
...

thrust::device_vector<int> vec0(100);
thrust::device_vector<int> vec1(100);
...

thrust::copy(vec0.begin(), vec0.end(),
             vec1.begin());

// vec1 is now a copy of vec0
```

Template Function `thrust::copy_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, Predicate)`

- Defined in file_thrust_copy.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::copy_if” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, Predicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
→ typename Predicate>__host__ __device__ OutputIterator thrust::copy_if(const
→ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
→ InputIterator, OutputIterator, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
→ typename OutputIterator, typename Predicate>__host__ __device__ OutputIterator
→ thrust::copy_if(const thrust::detail::execution_policy_base< DerivedPolicy > &,
→ InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate)
- template<typename InputIterator, typename OutputIterator, typename Predicate>
  OutputIterator thrust::copy_if(InputIterator, InputIterator, OutputIterator,
→ Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
→ OutputIterator, typename Predicate>
  OutputIterator thrust::copy_if(InputIterator1, InputIterator1, InputIterator2,
→ OutputIterator, Predicate)
```

Template Function `thrust::copy_if(InputIterator, InputIterator, OutputIterator, Predicate)`

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**, typename **Predicate**>
OutputIterator thrust::copy_if(*InputIterator* first, *InputIterator* last, *OutputIterator* result, *Predicate*
pred)

This version of `copy_if` copies elements from the range `[first, last)` to a range beginning at `result`, except that any element which causes `pred` to false is not copied. `copy_if` is stable, meaning that the relative order of elements that are copied is unchanged.

More precisely, for every integer `n` such that `0 <= n < last-first`, `copy_if` performs the assignment `*result = *(first+n)` and `result` is advanced one position if `pred(*(first+n))`. Otherwise, no assignment occurs and `result` is not advanced.

The following code snippet demonstrates how to use `copy_if` to perform stream compaction to copy even numbers to an output range.

Return `result + n`, where `n` is equal to the number of times `pred` evaluated to true in the range `[first, last)`.

Pre The ranges `[first, last)` and `[result, result + (last - first))` shall not overlap.

Parameters

- `first`: The beginning of the sequence from which to copy.
- `last`: The end of the sequence from which to copy.
- `result`: The beginning of the sequence into which to copy.
- `pred`: The predicate to test on every value of the range `[first, last)`.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `OutputIterator`: is a model of [Output Iterator](#).
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/copy.h>
...
struct is_even
{
    __host__ __device__
    bool operator() (const int x)
    {
        return (x % 2) == 0;
    }
};
...
const int N = 6;
int V[N] = {-2, 0, -1, 0, 1, 2};
int result[4];

thrust::copy_if(V, V + N, result, is_even());

// V remains {-2, 0, -1, 0, 1, 2}
// result is now {-2, 0, 0, 2}
```

See `remove_copy_if`

Template Function `thrust::copy_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::copy_if” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
  ↳ typename Predicate>__host__ __device__ OutputIterator thrust::copy_if(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳ InputIterator, OutputIterator, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename Predicate>__host__ __device__ OutputIterator
  ↳ thrust::copy_if(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate)
- template<typename InputIterator, typename OutputIterator, typename Predicate>
  ↳ OutputIterator thrust::copy_if(InputIterator, InputIterator, OutputIterator,
  ↳ Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename Predicate>
  ↳ OutputIterator thrust::copy_if(InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputIterator, Predicate)
```

Template Function `thrust::copy_if(InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**, typename **Predicate**>
OutputIterator thrust::copy_if(*InputIterator1* first, *InputIterator1* last, *InputIterator2* stencil, *OutputIterator* result, *Predicate* pred)

This version of `copy_if` copies elements from the range `[first, last)` to a range beginning at `result`, except that any element whose corresponding stencil element causes `pred` to be false is not copied. `copy_if` is stable, meaning that the relative order of elements that are copied is unchanged.

More precisely, for every integer `n` such that `0 <= n < last - first`, `copy_if` performs the assignment `*result = *(first + n)` and `result` is advanced one position if `pred(*(stencil + n))`. Otherwise, no assignment occurs and `result` is not advanced.

The following code snippet demonstrates how to use `copy_if` to perform stream compaction to copy numbers to an output range when corresponding stencil elements are even:

Return `result + n`, where `n` is equal to the number of times `pred` evaluated to true in the range `[stencil, stencil + (last - first))`.

Pre The ranges `[first, last)` and `[result, result + (last - first))` shall not overlap.

Pre The ranges `[stencil, stencil + (last - first))` and `[result, result + (last - first))` shall not overlap.

Parameters

- `first`: The beginning of the sequence from which to copy.
- `last`: The end of the sequence from which to copy.
- `stencil`: The beginning of the stencil sequence.
- `result`: The beginning of the sequence into which to copy.
- `pred`: The predicate to test on every value of the range `[stencil, stencil + (last-first))`.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#).
- `InputIterator2`: is a model of [Input Iterator](#), and `InputIterator2`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `OutputIterator`: is a model of [Output Iterator](#).
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/copy.h>
...
struct is_even
{
    __host__ __device__
    bool operator()(const int x)
    {
        return (x % 2) == 0;
    }
};
...
int N = 6;
int data[N] = { 0, 1, 2, 3, 4, 5};
int stencil[N] = {-2, 0, -1, 0, 1, 2};
int result[4];

thrust::copy_if(data, data + N, stencil, result, is_even());

// data remains      = { 0, 1, 2, 3, 4, 5};
// stencil remains   = {-2, 0, -1, 0, 1, 2};
// result is now      { 0, 1, 3, 5}
```

See `remove_copy_if`

Template Function `thrust::copy_n(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, Size, OutputIterator)`

- Defined in `file_thrust_copy.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::copy_n” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, Size, OutputIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename Size, typename
↳OutputIterator>__host__ __device__ OutputIterator thrust::copy_n(const_
↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator, Size,
↳OutputIterator)
- template<typename InputIterator, typename Size, typename OutputIterator>
  OutputIterator thrust::copy_n(InputIterator, Size, OutputIterator)
```

Template Function thrust::copy_n(InputIterator, Size, OutputIterator)

Function Documentation

template<typename **InputIterator**, typename **Size**, typename **OutputIterator**>

OutputIterator thrust::copy_n(*InputIterator first*, *Size n*, *OutputIterator result*)

copy_n copies elements from the range [first, first + n) to the range [result, result + n). That is, it performs the assignments *result = *first, *(result + 1) = *(first + 1), and so on. Generally, for every integer i from 0 to n, copy performs the assignment *(result + i) = *(first + i). Unlike std::copy_n, copy_n offers no guarantee on order of operation. As a result, calling copy_n with overlapping source and destination ranges has undefined behavior.

The return value is result + n.

The following code snippet demonstrates how to use copy to copy from one range to another.

Return The end of the destination range.

Pre result may be equal to first, but result shall not be in the range [first, first + n) otherwise.

Parameters

- first: The beginning of the range to copy.
- n: The number of elements to copy.
- result: The beginning destination range.

Template Parameters

- InputIterator: must be a model of [Input Iterator](#) and InputIterator's value_type must be convertible to OutputIterator's value_type.
- Size: is an integral type.
- OutputIterator: must be a model of [Output Iterator](#).

```
#include <thrust/copy.h>
#include <thrust/device_vector.h>
...
size_t n = 100;
thrust::device_vector<int> vec0(n);
thrust::device_vector<int> vec1(n);
...
thrust::copy_n(vec0.begin(), n, vec1.begin());

// vec1 is now a copy of vec0
```

See http://www.sgi.com/tech/stl/copy_n.html

See `thrust::copy`

Template Function `thrust::cos`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::cos(const complex < T > & z)
Returns the complex cosine of a complex number.

Parameters

- `z`: The complex argument.

Template Function `thrust::cosh`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::cosh(const complex < T > & z)
Returns the complex hyperbolic cosine of a complex number.

Parameters

- `z`: The complex argument.

Template Function `thrust::count(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, const EqualityComparable&)`

- Defined in `file_thrust_count.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::count`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, const EqualityComparable&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename
↳EqualityComparable>__host__ __device__ thrust::iterator_traits<InputIterator>
↳::difference_type thrust::count(const thrust::detail::execution_policy_base<
↳DerivedPolicy > &, InputIterator, InputIterator, const EqualityComparable &)
- template<typename InputIterator, typename EqualityComparable>
  thrust::iterator_traits<InputIterator>::difference_type
↳thrust::count(InputIterator, InputIterator, const EqualityComparable&)
```

Template Function `thrust::count(InputIterator, InputIterator, const EqualityComparable&)`

Function Documentation

template<typename **InputIterator**, typename **EqualityComparable**>

`thrust::iterator_traits<InputIterator>::difference_type thrust::count (InputIterator first, InputIterator last, const EqualityComparable &value)`

`count` finds the number of elements in `[first, last)` that are equal to `value`. More precisely, `count` returns the number of iterators `i` in `[first, last)` such that `*i == value`.

The following code snippet demonstrates how to use `count` to count the number of instances in a range of a value of interest.

```
#include <thrust/count.h>
#include <thrust/device_vector.h>
...
// put 3 1s in a device_vector
thrust::device_vector<int> vec(5,0);
vec[1] = 1;
vec[3] = 1;
vec[4] = 1;

// count the 1s
int result = thrust::count(vec.begin(), vec.end(), 1);
// result == 3
```

Return The number of elements equal to `value`.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.
- `value`: The value to be counted.

Template Parameters

- `InputIterator`: must be a model of [Input Iterator](#) and `InputIterator`'s `value_type` must be a model of [Equality Comparable](#).
- `EqualityComparable`: must be a model of [Equality Comparable](#) and can be compared for equality with `InputIterator`'s `value_type`

See <http://www.sgi.com/tech/stl/count.html>

Template Function `thrust::count_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`

- Defined in file `_thrust_count.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::count_if” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename Predicate>__
  ↳host__ __device__ thrust::iterator_traits<InputIterator>::difference_type_
  ↳thrust::count_if(const thrust::detail::execution_policy_base< DerivedPolicy > &, _
  ↳InputIterator, InputIterator, Predicate)
- template<typename InputIterator, typename Predicate>
  thrust::iterator_traits<InputIterator>::difference_type thrust::count_
  ↳if(InputIterator, InputIterator, Predicate)
```

Template Function thrust::count_if(InputIterator, InputIterator, Predicate)

Function Documentation

```
template<typename InputIterator, typename Predicate>
```

```
thrust::iterator_traits<InputIterator>::difference_type thrust::count_if (InputIterator first, InputIterator
                                                                    last, Predicate pred)
```

count_if finds the number of elements in [first, last) for which a predicate is true. More precisely, count_if returns the number of iterators i in [first, last) such that pred(*i) == true.

The following code snippet demonstrates how to use count to count the number of odd numbers in a range.

```
#include <thrust/count.h>
#include <thrust/device_vector.h>
...
struct is_odd
{
    __host__ __device__
    bool operator() (int &x)
    {
        return x & 1;
    }
};
...
// fill a device_vector with even & odd numbers
thrust::device_vector<int> vec(5);
vec[0] = 0;
vec[1] = 1;
vec[2] = 2;
vec[3] = 3;
vec[4] = 4;

// count the odd elements in vec
int result = thrust::count_if(vec.begin(), vec.end(), is_odd());
// result == 2
```

Return The number of elements where pred is true.

Parameters

- first: The beginning of the sequence.

- last: The end of the sequence.
- pred: The predicate.

Template Parameters

- InputIterator: must be a model of [Input Iterator](#) and InputIterator's value_type must be convertible to Predicate's argument_type.
- Predicate: must be a model of [Predicate](#).

See <http://www.sgi.com/tech/stl/count.html>

Template Function thrust::device_delete

Function Documentation

```
template<typename T>
void thrust::device_delete (thrust::device_ptr<T> ptr, const size_t n = 1)
    device_delete deletes a device_ptr allocated with device_new.
```

See [device_ptr](#)

See [device_new](#)

Parameters

- ptr: The *device_ptr* to delete, assumed to have been allocated with device_new.
- n: The number of objects to destroy at ptr. Defaults to 1 similar to device_new.

Function thrust::device_free

Function Documentation

```
void thrust::device_free (thrust::device_ptr<void> ptr)
    device_free deallocates memory allocated by the function device_malloc.
```

The following code snippet demonstrates how to use device_free to deallocate memory allocated by device_malloc.

Parameters

- ptr: A *device_ptr* pointing to memory to be deallocated.

```
#include <thrust/device_malloc.h>
#include <thrust/device_free.h>
...
// allocate some integers with device_malloc
const int N = 100;
thrust::device_ptr<int> int_array = thrust::device_malloc<int>(N);

// manipulate integers
...

// deallocate with device_free
thrust::device_free(int_array);
```

See [*device_ptr*](#)

See [*device_malloc*](#)

Function `thrust::device_malloc`

Function Documentation

device_ptr<T> thrust::device_malloc (const std::size_t n)

This version of `device_malloc` allocates sequential device storage for bytes.

The following code snippet demonstrates how to use `device_malloc` to allocate a range of device memory.

Return A *device_ptr* to the newly allocated memory.

Parameters

- n: The number of bytes to allocate sequentially in device memory.

```
#include <thrust/device_malloc.h>
#include <thrust/device_free.h>
...
// allocate some memory with device_malloc
const int N = 100;
thrust::device_ptr<void> void_ptr = thrust::device_malloc(N);

// manipulate memory
...

// deallocate with device_free
thrust::device_free(void_ptr);
```

This version of `device_malloc` allocates sequential device storage for new objects of the given type.

See [*device_ptr*](#)

See [*device_free*](#)

The following code snippet demonstrates how to use `device_malloc` to allocate a range of device memory.

Return A *device_ptr* to the newly allocated memory.

Parameters

- n: The number of objects of type T to allocate sequentially in device memory.

```
#include <thrust/device_malloc.h>
#include <thrust/device_free.h>
...
// allocate some integers with device_malloc
const int N = 100;
thrust::device_ptr<int> int_array = thrust::device_malloc<int>(N);

// manipulate integers
...

// deallocate with device_free
thrust::device_free(int_array);
```

See [*device_ptr*](#)

See [device_free](#)

Template Function `thrust::device_new(device_ptr<void>, const size_t)`

- Defined in `file_thrust_device_new.h`

Function Documentation

template<typename T>

`device_ptr<T>` thrust::device_new(`device_ptr<void>` p, const size_t n = 1)

`device_new` implements the placement new operator for types resident in device memory. `device_new` calls T's null constructor on a array of objects in device memory. No memory is allocated by this function.

Return p, casted to T's type.

See [device_ptr](#)

Parameters

- p: A [device_ptr](#) to a region of device memory into which to construct one or many Ts.
- n: The number of objects to construct at p.

Template Function `thrust::device_new(device_ptr<void>, const T&, const size_t)`

Function Documentation

template<typename T>

`device_ptr<T>` thrust::device_new(`device_ptr<void>` p, const T &exemplar, const size_t n = 1)

`device_new` implements the placement new operator for types resident in device memory. `device_new` calls T's copy constructor on a array of objects in device memory. No memory is allocated by this function.

Return p, casted to T's type.

See [device_ptr](#)

See `fill`

Parameters

- p: A [device_ptr](#) to a region of device memory into which to construct one or many Ts.
- exemplar: The value from which to copy.
- n: The number of objects to construct at p.

Template Function `thrust::device_new(const size_t)`

Function Documentation

template<typename T>

`device_ptr<T>` thrust::device_new(const size_t n = 1)

`device_new` implements the new operator for types resident in device memory. It allocates device memory large enough to hold n new objects of type T.

Return A *device_ptr* to the newly allocated region of device memory.

Parameters

- *n*: The number of objects to allocate. Defaults to 1.

Template Function `thrust::device_pointer_cast(T *)`

- Defined in `file_thrust_device_ptr.h`

Function Documentation

template<typename T>__host__ __device__ device_ptr<T> thrust::device_pointer_cast(T * ptr)
`device_pointer_cast` creates a *device_ptr* from a raw pointer which is presumed to point to a location in device memory.

Return A *device_ptr* wrapping *ptr*.

Parameters

- *ptr*: A raw pointer, presumed to point to a location in device memory.

Template Function `thrust::device_pointer_cast(const device_ptr<T>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::device_pointer_cast`” with arguments `(const device_ptr<T>&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T>__host__ __device__ device_ptr<T> thrust::device_pointer_
  ↳ cast(T *)
- template<typename T>__host__ __device__ device_ptr<T> thrust::device_pointer_
  ↳ cast(const device_ptr < T > &)
```

Template Function `thrust::distance`

- Defined in `file_thrust_distance.h`

Function Documentation

template<typename InputIterator>__host__ __device__ thrust::iterator_traits<InputIterator>
`distance` finds the distance between *first* and *last*, i.e. the number of times that *first* must be incremented until it is equal to *last*.

The following code snippet demonstrates how to use `distance` to compute the distance to one iterator from another.

Return The distance between the beginning and end of the input range.

Pre If *InputIterator* meets the requirements of random access iterator, *last* shall be reachable from *first* or *first* shall be reachable from *last*; otherwise, *last* shall be reachable from *first*.

Parameters

- first: The beginning of an input range of interest.
- last: The end of an input range of interest.

Template Parameters

- InputIterator: is a model of [Input Iterator](#).

```
#include <thrust/distance.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> vec(13);
thrust::device_vector<int>::iterator iter1 = vec.begin();
thrust::device_vector<int>::iterator iter2 = iter1 + 7;

int d = thrust::distance(iter1, iter2);

// d is 7
```

See <http://www.sgi.com/tech/stl/distance.html>

Template Function `thrust::equal(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2)`

- Defined in file `_thrust_equal.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::equal`” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename BinaryPredicate>__host__ __device__ bool thrust::equal(const_
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2>
  ↳ __host__ __device__ bool thrust::equal(const thrust::detail::execution_policy_base
  ↳ < DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳ BinaryPredicate>
  ↳ bool thrust::equal(InputIterator1, InputIterator1, InputIterator2,
  ↳ BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2>
  ↳ bool thrust::equal(InputIterator1, InputIterator1, InputIterator2)
```

Template Function `thrust::equal(InputIterator1, InputIterator1, InputIterator2)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**>

`bool thrust::equal (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2)`
 equal returns true if the two ranges [first1, last1) and [first2, first2 + (last1 - first1)) are identical when compared element-by-element, and otherwise returns false.

This version of equal returns true if and only if for every iterator i in [first1, last1), $*i == *(first2 + (i - first1))$.

The following code snippet demonstrates how to use equal to test two ranges for equality.

Return true, if the sequences are equal; false, otherwise.

Parameters

- first1: The beginning of the first sequence.
- last1: The end of the first sequence.
- first2: The beginning of the second sequence.

Template Parameters

- InputIterator1: is a model of [Input Iterator](#), and InputIterator1's value_type is a model of [Equality Comparable](#), and InputIterator1's value_type can be compared for equality with InputIterator2's value_type.
- InputIterator2: is a model of [Input Iterator](#), and InputIterator2's value_type is a model of [Equality Comparable](#), and InputIterator2's value_type can be compared for equality with InputIterator1's value_type.

```
#include <thrust/equal.h>
...
int A1[7] = {3, 1, 4, 1, 5, 9, 3};
int A2[7] = {3, 1, 4, 2, 8, 5, 7};
...
bool result = thrust::equal(A1, A1 + 7, A2);

// result == false
```

See <http://www.sgi.com/tech/stl/equal.html>

Template Function `thrust::equal(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::equal” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, BinaryPredicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename BinaryPredicate>__host__ __device__ bool thrust::equal(const_
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2>
  ↳ __host__ __device__ bool thrust::equal(const thrust::detail::execution_policy_base
  ↳ < DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳ BinaryPredicate>
  ↳ bool thrust::equal(InputIterator1, InputIterator1, InputIterator2,
  ↳ BinaryPredicate)
```

102 `template<typename InputIterator1, typename InputIterator2>`
`bool thrust::equal(InputIterator1, InputIterator1, InputIterator2)`

Template Function `thrust::equal(InputIterator1, InputIterator1, InputIterator2, BinaryPredicate)`

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename BinaryPredicate>
bool thrust::equal (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, BinaryPredicate bi-
                    nary_pred)
```

`equal` returns true if the two ranges `[first1, last1)` and `[first2, first2 + (last1 - first1))` are identical when compared element-by-element, and otherwise returns false.

This version of `equal` returns true if and only if for every iterator `i` in `[first1, last1)`, `binary_pred(*i, *(first2 + (i - first1)))` is true.

The following code snippet demonstrates how to use `equal` to compare the elements in two ranges modulo 2.

Return true, if the sequences are equal; false, otherwise.

Parameters

- `first1`: The beginning of the first sequence.
- `last1`: The end of the first sequence.
- `first2`: The beginning of the second sequence.
- `binary_pred`: Binary predicate used to test element equality.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), and `InputIterator1`'s `value_type` is convertible to `BinaryPredicate`'s `first_argument_type`.
- `InputIterator2`: is a model of [Input Iterator](#), and `InputIterator2`'s `value_type` is convertible to `BinaryPredicate`'s `second_argument_type`.
- `BinaryPredicate`: is a model of [Binary Predicate](#).

```
#include <thrust/equal.h>

struct compare_modulo_two
{
    __host__ __device__
    bool operator()(int x, int y) const
    {
        return (x % 2) == (y % 2);
    }
};

...
int x[6] = {0, 2, 4, 6, 8, 10};
int y[6] = {1, 3, 5, 7, 9, 11};

bool result = thrust::equal(x, x + 5, y, compare_modulo_two());

// result is true
```

See <http://www.sgi.com/tech/stl/equal.html>

Template Function `thrust::equal_range(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const LessThanComparable&)`

- Defined in `file_thrust_binary_search.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::equal_range`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `ForwardIterator`, `ForwardIterator`, `const LessThanComparable&`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<class ForwardIterator, class LessThanComparable>
  thrust::pair<ForwardIterator, ForwardIterator> thrust::equal_
  range(ForwardIterator, ForwardIterator, const LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  thrust::pair<ForwardIterator, ForwardIterator> thrust::equal_
  range(ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename
  LessThanComparable>__host__ __device__ thrust::pair<ForwardIterator,
  ForwardIterator> thrust::equal_range(const thrust::detail::execution_policy_base<
  DerivedPolicy > &, ForwardIterator, ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
  StrictWeakOrdering>__host__ __device__ thrust::pair<ForwardIterator,
  ForwardIterator> thrust::equal_range(const thrust::detail::execution_policy_base<
  DerivedPolicy > &, ForwardIterator, ForwardIterator, const T &,
  StrictWeakOrdering)
```

Template Function `thrust::equal_range(ForwardIterator, ForwardIterator, const LessThanComparable&)`

Function Documentation

```
template<class ForwardIterator, class LessThanComparable>
thrust::pair<ForwardIterator, ForwardIterator> thrust::equal_range(ForwardIterator first, ForwardIterator last, const LessThanComparable &value)
```

`equal_range` is a version of binary search: it attempts to find the element value in an ordered range `[first, last)`. The value returned by `equal_range` is essentially a combination of the values returned by `lower_bound` and `upper_bound`: it returns a pair of iterators `i` and `j` such that `i` is the first position where `value` could be inserted without violating the ordering and `j` is the last position where `value` could be inserted without violating the ordering. It follows that every element in the range `[i, j)` is equivalent to `value`, and that `[i, j)` is the largest subrange of `[first, last)` that has this property.

This version of `equal_range` returns a pair of iterators `[i, j)`, where `i` is the furthestmost iterator in `[first, last)` such that, for every iterator `k` in `[first, i)`, `*k < value`. `j` is the furthestmost iterator in `[first, last)` such that, for every iterator `k` in `[first, j)`, `value < *k` is false. For every iterator `k` in `[i, j)`, neither `value < *k` nor `*k < value` is true.

The following code snippet demonstrates how to use `equal_range` to search for values in a ordered range.

Return A pair of iterators `[i, j)` that define the range of equivalent elements.

Parameters

- first: The beginning of the ordered sequence.
- last: The end of the ordered sequence.
- value: The value to be searched.

Template Parameters

- ForwardIterator: is a model of [Forward Iterator](#).
- LessThanComparable: is a model of [LessThanComparable](#).

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::equal_range(input.begin(), input.end(), 0); // returns [input.begin(),
↳ input.begin() + 1)
thrust::equal_range(input.begin(), input.end(), 1); // returns [input.begin() + 1,
↳ input.begin() + 1)
thrust::equal_range(input.begin(), input.end(), 2); // returns [input.begin() + 1,
↳ input.begin() + 2)
thrust::equal_range(input.begin(), input.end(), 3); // returns [input.begin() + 2,
↳ input.begin() + 2)
thrust::equal_range(input.begin(), input.end(), 8); // returns [input.begin() + 4,
↳ input.end)
thrust::equal_range(input.begin(), input.end(), 9); // returns [input.end(),
↳ input.end)
```

See http://www.sgi.com/tech/stl/equal_range.html

See [lower_bound](#)

See [upper_bound](#)

See [binary_search](#)

Template Function `thrust::equal_range(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::equal_range” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<class ForwardIterator, class LessThanComparable>
  thrust::pair<ForwardIterator, ForwardIterator> thrust::equal_
  ↳range(ForwardIterator, ForwardIterator, const LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  thrust::pair<ForwardIterator, ForwardIterator> thrust::equal_
  ↳range(ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename
  ↳LessThanComparable>__host__ __device__ thrust::pair<ForwardIterator,
  ↳ForwardIterator> thrust::equal_range(const thrust::detail::execution_policy_base<
  ↳DerivedPolicy > &, ForwardIterator, ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
  ↳StrictWeakOrdering>__host__ __device__ thrust::pair<ForwardIterator,
  ↳ForwardIterator> thrust::equal_range(const thrust::detail::execution_policy_base<
  ↳DerivedPolicy > &, ForwardIterator, ForwardIterator, const T &,
  ↳StrictWeakOrdering)

```

Template Function `thrust::equal_range(ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`

Function Documentation

```

template<class ForwardIterator, class T, class StrictWeakOrdering>
thrust::pair<ForwardIterator, ForwardIterator> thrust::equal_range(ForwardIterator first, ForwardIt-
                                                                erator last, const T &value,
                                                                StrictWeakOrdering comp)

```

`equal_range` is a version of binary search: it attempts to find the element value in an ordered range `[first, last)`. The value returned by `equal_range` is essentially a combination of the values returned by `lower_bound` and `upper_bound`: it returns a pair of iterators `i` and `j` such that `i` is the first position where value could be inserted without violating the ordering and `j` is the last position where value could be inserted without violating the ordering. It follows that every element in the range `[i, j)` is equivalent to value, and that `[i, j)` is the largest subrange of `[first, last)` that has this property.

This version of `equal_range` returns a pair of iterators `[i, j)`. `i` is the furthestmost iterator in `[first, last)` such that, for every iterator `k` in `[first, i)`, `comp(*k, value)` is true. `j` is the furthestmost iterator in `[first, last)` such that, for every iterator `k` in `[first, last)`, `comp(value, *k)` is false. For every iterator `k` in `[i, j)`, neither `comp(value, *k)` nor `comp(*k, value)` is true.

The following code snippet demonstrates how to use `equal_range` to search for values in an ordered range.

Return A pair of iterators `[i, j)` that define the range of equivalent elements.

Parameters

- `first`: The beginning of the ordered sequence.
- `last`: The end of the ordered sequence.
- `value`: The value to be searched.
- `comp`: The comparison operator.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#).
- `T`: is comparable to `ForwardIterator`'s `value_type`.
- `StrictWeakOrdering`: is a model of [Strict Weak Ordering](#).

```

#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
#include <thrust/functional.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::equal_range(input.begin(), input.end(), 0, thrust::less<int>()); //
↳returns [input.begin(), input.begin() + 1)
thrust::equal_range(input.begin(), input.end(), 1, thrust::less<int>()); //
↳returns [input.begin() + 1, input.begin() + 1)
thrust::equal_range(input.begin(), input.end(), 2, thrust::less<int>()); //
↳returns [input.begin() + 1, input.begin() + 2)
thrust::equal_range(input.begin(), input.end(), 3, thrust::less<int>()); //
↳returns [input.begin() + 2, input.begin() + 2)
thrust::equal_range(input.begin(), input.end(), 8, thrust::less<int>()); //
↳returns [input.begin() + 4, input.end)
thrust::equal_range(input.begin(), input.end(), 9, thrust::less<int>()); //
↳returns [input.end(), input.end)

```

See http://www.sgi.com/tech/stl/equal_range.html

See `lower_bound`

See `upper_bound`

See `binary_search`

Template Function `thrust::exclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator)`

- Defined in file `thrust_scan.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::exclusive_scan” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
↳typename T, typename AssociativeOperator>__host__ __device__ OutputIterator
↳thrust::exclusive_scan(const thrust::detail::execution_policy_base< DerivedPolicy
↳> &, InputIterator, InputIterator, OutputIterator, T, AssociativeOperator)
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
↳typename T>__host__ __device__ OutputIterator thrust::exclusive_scan(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
↳InputIterator, OutputIterator, T)
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator>_
↳__host__ __device__ OutputIterator thrust::exclusive_scan(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
↳InputIterator, OutputIterator)
- template<typename InputIterator, typename OutputIterator, typename T, typename
↳AssociativeOperator>
  OutputIterator thrust::exclusive_scan(InputIterator, InputIterator,
↳OutputIterator, T, AssociativeOperator)
- template<typename InputIterator, typename OutputIterator, typename T>
  OutputIterator thrust::exclusive_scan(InputIterator, InputIterator,
↳OutputIterator, T)
- template<typename InputIterator, typename OutputIterator>
  OutputIterator thrust::exclusive_scan(InputIterator, InputIterator,
↳OutputIterator)

```

Template Function `thrust::exclusive_scan(InputIterator, InputIterator, OutputIterator)`

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**>

OutputIterator thrust::exclusive_scan(*InputIterator first*, *InputIterator last*, *OutputIterator result*)

`exclusive_scan` computes an exclusive prefix sum operation. The term ‘exclusive’ means that each result does not include the corresponding input operand in the partial sum. More precisely, 0 is assigned to `*result` and the sum of 0 and `*first` is assigned to `*(result + 1)`, and so on. This version of `exclusive_scan` assumes plus as the associative operator and 0 as the initial value. When the input and output sequences are the same, the scan is performed in-place.

The following code snippet demonstrates how to use `exclusive_scan`

Return The end of the output sequence.

Pre `first` may equal `result` but the range `[first, last)` and the range `[result, result + (last - first))` shall not overlap otherwise.

Parameters

- `first`: The beginning of the input sequence.
- `last`: The end of the input sequence.
- `result`: The beginning of the output sequence.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#) and `InputIterator`'s `value_type` is convertible to `OutputIterator`'s `value_type`.
- `OutputIterator`: is a model of [Output Iterator](#), and if `x` and `y` are objects of `OutputIterator`'s `value_type`, then `x + y` is defined. If `T` is `OutputIterator`'s

value_type, then $T(0)$ is defined.

```
#include <thrust/scan.h>

int data[6] = {1, 0, 2, 2, 1, 3};

thrust::exclusive_scan(data, data + 6, data); // in-place scan

// data is now {0, 1, 1, 3, 5, 6}
```

See http://www.sgi.com/tech/stl/partial_sum.html

Template Function `thrust::exclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, T)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::exclusive_scan” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, T) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
  ↳typename T, typename AssociativeOperator>__host__ __device__ OutputIterator
  ↳thrust::exclusive_scan(const thrust::detail::execution_policy_base< DerivedPolicy
  ↳> &, InputIterator, InputIterator, OutputIterator, T, AssociativeOperator)
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
  ↳typename T>__host__ __device__ OutputIterator thrust::exclusive_scan(const
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳InputIterator, OutputIterator, T)
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator>_
  ↳__host__ __device__ OutputIterator thrust::exclusive_scan(const
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳InputIterator, OutputIterator)
- template<typename InputIterator, typename OutputIterator, typename T, typename
  ↳AssociativeOperator>
  ↳OutputIterator thrust::exclusive_scan(InputIterator, InputIterator,
  ↳OutputIterator, T, AssociativeOperator)
- template<typename InputIterator, typename OutputIterator, typename T>
  ↳OutputIterator thrust::exclusive_scan(InputIterator, InputIterator,
  ↳OutputIterator, T)
- template<typename InputIterator, typename OutputIterator>
  ↳OutputIterator thrust::exclusive_scan(InputIterator, InputIterator,
  ↳OutputIterator)
```

Template Function `thrust::exclusive_scan(InputIterator, InputIterator, OutputIterator, T)`

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**, typename **T**>

OutputIterator thrust::exclusive_scan(*InputIterator first*, *InputIterator last*, *OutputIterator result*, *T init*)

exclusive_scan computes an exclusive prefix sum operation. The term ‘exclusive’ means that each result

does not include the corresponding input operand in the partial sum. More precisely, `init` is assigned to `*result` and the sum of `init` and `*first` is assigned to `*(result + 1)`, and so on. This version of `exclusive_scan` assumes plus as the associative operator but requires an initial value `init`. When the input and output sequences are the same, the scan is performed in-place.

The following code snippet demonstrates how to use `exclusive_scan`

Return The end of the output sequence.

Pre `first` may equal `result` but the range `[first, last)` and the range `[result, result + (last - first))` shall not overlap otherwise.

Parameters

- `first`: The beginning of the input sequence.
- `last`: The end of the input sequence.
- `result`: The beginning of the output sequence.
- `init`: The initial value.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#) and `InputIterator`'s `value_type` is convertible to `OutputIterator`'s `value_type`.
- `OutputIterator`: is a model of [Output Iterator](#), and if `x` and `y` are objects of `OutputIterator`'s `value_type`, then `x + y` is defined.
- `T`: is convertible to `OutputIterator`'s `value_type`.

```
#include <thrust/scan.h>

int data[6] = {1, 0, 2, 2, 1, 3};

thrust::exclusive_scan(data, data + 6, data, 4); // in-place scan

// data is now {4, 5, 5, 7, 9, 10}
```

See http://www.sgi.com/tech/stl/partial_sum.html

Template Function `thrust::exclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, T, AssociativeOperator)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::exclusive_scan`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, T, AssociativeOperator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
↳typename T, typename AssociativeOperator>__host__ __device__ OutputIterator
↳thrust::exclusive_scan(const thrust::detail::execution_policy_base< DerivedPolicy
↳> &, InputIterator, InputIterator, OutputIterator, T, AssociativeOperator)
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
↳typename T>__host__ __device__ OutputIterator thrust::exclusive_scan(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
↳InputIterator, OutputIterator, T)
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator>_
↳__host__ __device__ OutputIterator thrust::exclusive_scan(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
↳InputIterator, OutputIterator)
- template<typename InputIterator, typename OutputIterator, typename T, typename
↳AssociativeOperator>
  OutputIterator thrust::exclusive_scan(InputIterator, InputIterator,
↳OutputIterator, T, AssociativeOperator)
- template<typename InputIterator, typename OutputIterator, typename T>
  OutputIterator thrust::exclusive_scan(InputIterator, InputIterator,
↳OutputIterator, T)
- template<typename InputIterator, typename OutputIterator>
  OutputIterator thrust::exclusive_scan(InputIterator, InputIterator,
↳OutputIterator)

```

Template Function `thrust::exclusive_scan(InputIterator, InputIterator, OutputIterator, T, Associative-Operator)`

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**, typename **T**, typename **AssociativeOperator**>
OutputIterator thrust::**exclusive_scan** (*InputIterator first, InputIterator last, OutputIterator result, T*
init, AssociativeOperator binary_op)

`exclusive_scan` computes an exclusive prefix sum operation. The term ‘exclusive’ means that each result does not include the corresponding input operand in the partial sum. More precisely, `init` is assigned to `*result` and the value `binary_op(init, *first)` is assigned to `*(result + 1)`, and so on. This version of the function requires both an associative operator and an initial value `init`. When the input and output sequences are the same, the scan is performed in-place.

The following code snippet demonstrates how to use `exclusive_scan`

Return The end of the output sequence.

Pre `first` may equal `result` but the range `[first, last)` and the range `[result, result + (last - first))` shall not overlap otherwise.

Parameters

- `first`: The beginning of the input sequence.
- `last`: The end of the input sequence.
- `result`: The beginning of the output sequence.
- `init`: The initial value.
- `binary_op`: The associative operator used to ‘sum’ values.

Template Parameters

- InputIterator: is a model of [Input Iterator](#) and InputIterator's value_type is convertible to OutputIterator's value_type.
- OutputIterator: is a model of [Output Iterator](#) and OutputIterator's value_type is convertible to both AssociativeOperator's first_argument_type and second_argument_type.
- T: is convertible to OutputIterator's value_type.
- AssociativeOperator: is a model of [Binary Function](#) and AssociativeOperator's result_type is convertible to OutputIterator's value_type.

```
#include <thrust/scan.h>
#include <thrust/functional.h>

int data[10] = {-5, 0, 2, -3, 2, 4, 0, -1, 2, 8};

thrust::maximum<int> binary_op;

thrust::exclusive_scan(data, data + 10, data, 1, binary_op); // in-place scan

// data is now {1, 1, 1, 2, 2, 2, 4, 4, 4, 4 }
```

See http://www.sgi.com/tech/stl/partial_sum.html

Template Function `thrust::exclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator)`

- Defined in file `thrust_scan.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::exclusive_scan_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename T, typename BinaryPredicate, typename
  ↳ AssociativeOperator>__host__ __device__ OutputIterator thrust::exclusive_scan_by_
  ↳ key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, OutputIterator, T,
  ↳ BinaryPredicate, AssociativeOperator)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename T, typename BinaryPredicate>__host__ __device__
  ↳ OutputIterator thrust::exclusive_scan_by_key(const thrust::detail::execution_
  ↳ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputIterator, T, BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename T>__host__ __device__ OutputIterator
  ↳ thrust::exclusive_scan_by_key(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2, OutputIterator,
  ↳ T)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::exclusive_
  ↳ scan_by_key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename T, typename BinaryPredicate, typename
  ↳ AssociativeOperator>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator, T, BinaryPredicate, AssociativeOperator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename T, typename BinaryPredicate>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator, T, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename T>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator, T)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator)

```

Template Function `thrust::exclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**>
OutputIterator thrust::exclusive_scan_by_key(*InputIterator1 first1*, *InputIterator1 last1*, *InputIt-*
erator2 first2, *OutputIterator result*)

exclusive_scan_by_key computes an exclusive segmented prefix

This version of `exclusive_scan_by_key` uses the value 0 to initialize the exclusive scan operation.

This version of `exclusive_scan_by_key` assumes plus as the associative operator used to perform the prefix sum. When the input and output sequences are the same, the scan is performed in-place.

This version of `exclusive_scan_by_key` assumes `equal_to` as the binary predicate used to compare adjacent keys. Specifically, consecutive iterators *i* and *i*+1 in the range [*first1*, *last1*] belong to the

same segment if `*i == *(i+1)`, and belong to different segments otherwise.

Refer to the most general form of `exclusive_scan_by_key` for additional details.

The following code snippet demonstrates how to use `exclusive_scan_by_key`.

Pre `first1` may equal `result` but the range `[first1, last1)` and the range `[result, result + (last1 - first1))` shall not overlap otherwise.

Pre `first2` may equal `result` but the range `[first2, first2 + (last1 - first1)` and range `[result, result + (last1 - first1))` shall not overlap otherwise.

Parameters

- `first1`: The beginning of the key sequence.
- `last1`: The end of the key sequence.
- `first2`: The beginning of the input value sequence.
- `result`: The beginning of the output value sequence.

```
#include <thrust/scan.h>

int keys[10] = {0, 0, 0, 1, 1, 2, 3, 3, 3, 3};
int vals[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};

thrust::exclusive_scan_by_key(key, key + 10, vals, vals); // in-place scan

// vals is now {0, 1, 2, 0, 1, 0, 0, 1, 2, 3};
```

See `exclusive_scan`

Template Function `thrust::exclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, T)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::exclusive_scan_by_key`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, T)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename T, typename BinaryPredicate, typename
  ↳ AssociativeOperator>__host__ __device__ OutputIterator thrust::exclusive_scan_by_
  ↳ key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, OutputIterator, T,
  ↳ BinaryPredicate, AssociativeOperator)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename T, typename BinaryPredicate>__host__ __device__
  ↳ OutputIterator thrust::exclusive_scan_by_key(const thrust::detail::execution_
  ↳ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputIterator, T, BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename T>__host__ __device__ OutputIterator
  ↳ thrust::exclusive_scan_by_key(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2, OutputIterator,
  ↳ T)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::exclusive_
  ↳ scan_by_key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename T, typename BinaryPredicate, typename
  ↳ AssociativeOperator>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator, T, BinaryPredicate, AssociativeOperator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename T, typename BinaryPredicate>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator, T, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename T>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator, T)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator)

```

Template Function `thrust::exclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator, T)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**, typename **T**>
OutputIterator thrust::exclusive_scan_by_key(*InputIterator1 first1, InputIterator1 last1, InputIt-*
erator2 first2, OutputIterator result, T init)

`exclusive_scan_by_key` computes an exclusive key-value or ‘segmented’ prefix sum operation. The term ‘exclusive’ means that each result does not include the corresponding input operand in the partial sum. The term ‘segmented’ means that the partial sums are broken into distinct segments. In other words, within each segment a separate exclusive scan operation is computed. Refer to the code sample below for example usage.

This version of `exclusive_scan_by_key` uses the value `init` to initialize the exclusive scan operation.

The following code snippet demonstrates how to use `exclusive_scan_by_key`

Return The end of the output sequence.

Pre `first1` may equal `result` but the range `[first1, last1)` and the range `[result, result + (last1 - first1))` shall not overlap otherwise.

Pre `first2` may equal `result` but the range `[first2, first2 + (last1 - first1)` and range `[result, result + (last1 - first1))` shall not overlap otherwise.

Parameters

- `first1`: The beginning of the key sequence.
- `last1`: The end of the key sequence.
- `first2`: The beginning of the input value sequence.
- `result`: The beginning of the output value sequence.
- `init`: The initial of the exclusive sum value.

```
#include <thrust/scan.h>
#include <thrust/functional.h>

int keys[10] = {0, 0, 0, 1, 1, 2, 3, 3, 3, 3};
int vals[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};

int init = 5;

thrust::exclusive_scan_by_key(key, key + 10, vals, vals, init); // in-place scan

// vals is now {5, 6, 7, 5, 6, 5, 5, 6, 7, 8};
```

See `exclusive_scan`

See `inclusive_scan_by_key`

Template Function `thrust::exclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, T, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::exclusive_scan_by_key`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, T, BinaryPredicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:


```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename T, typename BinaryPredicate, typename_
  ↳ AssociativeOperator>__host__ __device__ OutputIterator thrust::exclusive_scan_by_
  ↳ key(const thrust::detail::execution_policy_base< DerivedPolicy > &,_
  ↳ InputIterator1, InputIterator1, InputIterator2, OutputIterator, T,_
  ↳ BinaryPredicate, AssociativeOperator)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename T, typename BinaryPredicate>__host__ __device__
  ↳ OutputIterator thrust::exclusive_scan_by_key(const thrust::detail::execution_
  ↳ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,_
  ↳ OutputIterator, T, BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename T>__host__ __device__ OutputIterator_
  ↳ thrust::exclusive_scan_by_key(const thrust::detail::execution_policy_base<_
  ↳ DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2, OutputIterator,
  ↳ T)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::exclusive_
  ↳ scan_by_key(const thrust::detail::execution_policy_base< DerivedPolicy > &,_
  ↳ InputIterator1, InputIterator1, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳ OutputIterator, typename T, typename BinaryPredicate, typename_
  ↳ AssociativeOperator>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,_
  ↳ InputIterator2, OutputIterator, T, BinaryPredicate, AssociativeOperator)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳ OutputIterator, typename T, typename BinaryPredicate>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,_
  ↳ InputIterator2, OutputIterator, T, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳ OutputIterator, typename T>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,_
  ↳ InputIterator2, OutputIterator, T)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳ OutputIterator>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,_
  ↳ InputIterator2, OutputIterator)

```

Template Function `thrust::exclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator, T, BinaryPredicate)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**, typename **T**, typename **BinaryPredicate**> **OutputIterator** thrust::exclusive_scan_by_key(*InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, OutputIterator result, T init, BinaryPredicate binary_pred*)

`exclusive_scan_by_key` computes an exclusive key-value or ‘segmented’ prefix sum operation. The term ‘exclusive’ means that each result does not include the corresponding input operand in the partial sum. The term ‘segmented’ means that the partial sums are broken into distinct segments. In other words, within each segment a separate exclusive scan operation is computed. Refer to the code sample below for example usage.

This version of `exclusive_scan_by_key` uses the value `init` to initialize the exclusive scan operation.

This version of `exclusive_scan_by_key` uses the binary predicate `binary_pred` to compare adjacent

keys. Specifically, consecutive iterators `i` and `i+1` in the range `[first1, last1)` belong to the same segment if `binary_pred(*i, *(i+1))` is true, and belong to different segments otherwise.

The following code snippet demonstrates how to use `exclusive_scan_by_key`

Return The end of the output sequence.

Pre `first1` may equal `result` but the range `[first1, last1)` and the range `[result, result + (last1 - first1))` shall not overlap otherwise.

Pre `first2` may equal `result` but the range `[first2, first2 + (last1 - first1)` and range `[result, result + (last1 - first1))` shall not overlap otherwise.

Parameters

- `first1`: The beginning of the key sequence.
- `last1`: The end of the key sequence.
- `first2`: The beginning of the input value sequence.
- `result`: The beginning of the output value sequence.
- `init`: The initial of the exclusive sum value.
- `binary_pred`: The binary predicate used to determine equality of keys.

```
#include <thrust/scan.h>
#include <thrust/functional.h>

int keys[10] = {0, 0, 0, 1, 1, 2, 3, 3, 3, 3};
int vals[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};

int init = 5;

thrust::equal_to<int> binary_pred;

thrust::exclusive_scan_by_key(key, key + 10, vals, vals, init, binary_pred); // in-place scan

// vals is now {5, 6, 7, 5, 6, 5, 5, 6, 7, 8};
```

See `exclusive_scan`

See `inclusive_scan_by_key`

Template Function `thrust::exclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, T, BinaryPredicate, AssociativeOperator)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::exclusive_scan_by_key`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, T, BinaryPredicate, AssociativeOperator)` in doxygen xml output for project “roc-Thrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename T, typename BinaryPredicate, typename
  ↳ AssociativeOperator>__host__ __device__ OutputIterator thrust::exclusive_scan_by_
  ↳ key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, OutputIterator, T,
  ↳ BinaryPredicate, AssociativeOperator)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename T, typename BinaryPredicate>__host__ __device__
  ↳ OutputIterator thrust::exclusive_scan_by_key(const thrust::detail::execution_
  ↳ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputIterator, T, BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename T>__host__ __device__ OutputIterator
  ↳ thrust::exclusive_scan_by_key(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2, OutputIterator,
  ↳ T)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::exclusive_
  ↳ scan_by_key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename T, typename BinaryPredicate, typename
  ↳ AssociativeOperator>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator, T, BinaryPredicate, AssociativeOperator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename T, typename BinaryPredicate>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator, T, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename T>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator, T)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator>
  ↳ OutputIterator thrust::exclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator)

```

Template Function `thrust::exclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator, T, BinaryPredicate, AssociativeOperator)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**, typename **T**, typename **BinaryPredicate**, typename **AssociativeOperator**> **OutputIterator** thrust::exclusive_scan_by_key(*InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, OutputIterator result, T init, BinaryPredicate binary_pred, AssociativeOperator binary_op*)

`exclusive_scan_by_key` computes an exclusive key-value or ‘segmented’ prefix sum operation. The term ‘exclusive’ means that each result does not include the corresponding input operand in the partial sum. The term ‘segmented’ means that the partial sums are broken into distinct segments. In other words, within each segment a separate exclusive scan operation is computed. Refer to the code sample below for example usage.

This version of `exclusive_scan_by_key` uses the value `init` to initialize the exclusive scan operation.

This version of `exclusive_scan_by_key` uses the binary predicate `binary_pred` to compare adjacent keys. Specifically, consecutive iterators `i` and `i+1` in the range `[first1, last1)` belong to the same segment if `binary_pred(*i, *(i+1))` is true, and belong to different segments otherwise.

This version of `exclusive_scan_by_key` uses the associative operator `binary_op` to perform the prefix sum. When the input and output sequences are the same, the scan is performed in-place.

The following code snippet demonstrates how to use `exclusive_scan_by_key`

Return The end of the output sequence.

Pre `first1` may equal `result` but the range `[first1, last1)` and the range `[result, result + (last1 - first1))` shall not overlap otherwise.

Pre `first2` may equal `result` but the range `[first2, first2 + (last1 - first1)` and range `[result, result + (last1 - first1))` shall not overlap otherwise.

Parameters

- `first1`: The beginning of the key sequence.
- `last1`: The end of the key sequence.
- `first2`: The beginning of the input value sequence.
- `result`: The beginning of the output value sequence.
- `init`: The initial of the exclusive sum value.
- `binary_pred`: The binary predicate used to determine equality of keys.
- `binary_op`: The associative operator used to 'sum' values.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#)
- `InputIterator2`: is a model of [Input Iterator](#) and `InputIterator2`'s `value_type` is convertible to `OutputIterator`'s `value_type`.
- `OutputIterator`: is a model of [Output Iterator](#), and if `x` and `y` are objects of `OutputIterator`'s `value_type`, then `binary_op(x, y)` is defined.
- `T`: is convertible to `OutputIterator`'s `value_type`.
- `BinaryPredicate`: is a model of [Binary Predicate](#).
- `AssociativeOperator`: is a model of [Binary Function](#) and `AssociativeOperator`'s `result_type` is convertible to `OutputIterator`'s `value_type`.

```
#include <thrust/scan.h>
#include <thrust/functional.h>

int keys[10] = {0, 0, 0, 1, 1, 2, 3, 3, 3, 3};
int vals[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};

int init = 5;

thrust::equal_to<int> binary_pred;
thrust::plus<int>      binary_op;

thrust::exclusive_scan_by_key(key, key + 10, vals, vals, init, binary_pred,
    ↪binary_op); // in-place scan

// vals is now {5, 6, 7, 5, 6, 5, 5, 6, 7, 8};
```

See `exclusive_scan`

See `inclusive_scan_by_key`

Template Function `thrust::exp`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::exp(const complex < T > & z)
Returns the complex exponential of a complex number.

Parameters

- `z`: The complex argument.

Template Function `thrust::fill(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&)`

- Defined in `file_thrust_fill.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::fill`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `ForwardIterator`, `ForwardIterator`, `const T&`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename T>__host__ __
  ↳device__ void thrust::fill(const thrust::detail::execution_policy_base<
  ↳DerivedPolicy > &, ForwardIterator, ForwardIterator, const T &)
- template<typename ForwardIterator, typename T>__host__ __device__ void
  ↳thrust::fill(ForwardIterator, ForwardIterator, const T &)
```

Template Function `thrust::fill(ForwardIterator, ForwardIterator, const T&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::fill`” with arguments (`ForwardIterator`, `ForwardIterator`, `const T&`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename T>__host__ __
  ↳device__ void thrust::fill(const thrust::detail::execution_policy_base<
  ↳DerivedPolicy > &, ForwardIterator, ForwardIterator, const T &)
- template<typename ForwardIterator, typename T>__host__ __device__ void
  ↳thrust::fill(ForwardIterator, ForwardIterator, const T &)
```

Template Function thrust::fill_n(const thrust::detail::execution_policy_base<DerivedPolicy>&, OutputIterator, Size, const T&)

- Defined in file_thrust_fill.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::fill_n” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, OutputIterator, Size, const T&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename OutputIterator, typename Size, typename
↳T>__host__ __device__ OutputIterator thrust::fill_n(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, OutputIterator, Size,
↳const T &)
- template<typename OutputIterator, typename Size, typename T>__host__ __device__
↳OutputIterator thrust::fill_n(OutputIterator, Size, const T &)
```

Template Function thrust::fill_n(OutputIterator, Size, const T&)

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::fill_n” with arguments (OutputIterator, Size, const T&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename OutputIterator, typename Size, typename
↳T>__host__ __device__ OutputIterator thrust::fill_n(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, OutputIterator, Size,
↳const T &)
- template<typename OutputIterator, typename Size, typename T>__host__ __device__
↳OutputIterator thrust::fill_n(OutputIterator, Size, const T &)
```

Template Function thrust::find(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, const T&)

- Defined in file_thrust_find.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::find” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, const T&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename T>__host__ __
↳device__ InputIterator thrust::find(const thrust::detail::execution_policy_base<
↳DerivedPolicy > &, InputIterator, InputIterator, const T &)
- template<typename InputIterator, typename T>
↳InputIterator thrust::find(InputIterator, InputIterator, const T &)
```

Template Function `thrust::find(InputIterator, InputIterator, const T&)`

Function Documentation

template<typename **InputIterator**, typename **T**>

InputIterator thrust::find(*InputIterator* first, *InputIterator* last, const *T* &value)

find returns the first iterator *i* in the range [*first*, *last*) such that **i* == *value* or *last* if no such iterator exists.

```
#include <thrust/find.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> input(4);

input[0] = 0;
input[1] = 5;
input[2] = 3;
input[3] = 7;

thrust::device_vector<int>::iterator iter;

iter = thrust::find(input.begin(), input.end(), 3); // returns input.first() + 2
iter = thrust::find(input.begin(), input.end(), 5); // returns input.first() + 1
iter = thrust::find(input.begin(), input.end(), 9); // returns input.end()
```

Return The first iterator *i* such that **i* == *value* or *last*.

Parameters

- *first*: Beginning of the sequence to search.
- *last*: End of the sequence to search.
- *value*: The value to find.

Template Parameters

- *InputIterator*: is a model of [Input Iterator](#) and *InputIterator*'s `value_type` is equality comparable to type *T*.
- *T*: is a model of [EqualityComparable](#).

See `find_if`

See `mismatch`

Template Function `thrust::find_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`

- Defined in `file_thrust_find.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::find_if” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename Predicate>__
↪host__ __device__ InputIterator thrust::find_if(const thrust::detail::execution_
↪policy_base< DerivedPolicy > &, InputIterator, InputIterator, Predicate)
- template<typename InputIterator, typename Predicate>
InputIterator thrust::find_if(InputIterator, InputIterator, Predicate)
```

Template Function thrust::find_if(InputIterator, InputIterator, Predicate)

Function Documentation

template<typename **InputIterator**, typename **Predicate**>

InputIterator thrust::find_if(*InputIterator* first, *InputIterator* last, *Predicate* pred)

find_if returns the first iterator *i* in the range [*first*, *last*) such that *pred*(**i*) is true or last if no such iterator exists.

```
#include <thrust/find.h>
#include <thrust/device_vector.h>

struct greater_than_four
{
    __host__ __device__
    bool operator() (int x)
    {
        return x > 4;
    }
};

struct greater_than_ten
{
    __host__ __device__
    bool operator() (int x)
    {
        return x > 10;
    }
};

...
thrust::device_vector<int> input(4);

input[0] = 0;
input[1] = 5;
input[2] = 3;
input[3] = 7;

thrust::device_vector<int>::iterator iter;

iter = thrust::find_if(input.begin(), input.end(), greater_than_four()); // ↪
↪returns input.first() + 1
```

(continues on next page)

(continued from previous page)

```
iter = thrust::find_if(input.begin(), input.end(), greater_than_ten()); // 
↳ returns input.end()
```

Return The first iterator *i* such that `pred(*i)` is true, or last.

Parameters

- `first`: Beginning of the sequence to search.
- `last`: End of the sequence to search.
- `pred`: A predicate used to test range elements.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#).
- `Predicate`: is a model of [Predicate](#).

See `find`

See `find_if_not`

See `mismatch`

Template Function `thrust::find_if_not(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`

- Defined in file `thrust_find.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::find_if_not`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename Predicate>__
↳host__ __device__ InputIterator thrust::find_if_not(const_
↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
↳InputIterator, Predicate)
- template<typename InputIterator, typename Predicate>
  InputIterator thrust::find_if_not(InputIterator, InputIterator, Predicate)
```

Template Function `thrust::find_if_not(InputIterator, InputIterator, Predicate)`

Function Documentation

template<typename **InputIterator**, typename **Predicate**>

InputIterator thrust::find_if_not(*InputIterator* first, *InputIterator* last, *Predicate* pred)

`find_if_not` returns the first iterator *i* in the range `[first, last)` such that `pred(*i)` is false or last if no such iterator exists.

```
#include <thrust/find.h>
#include <thrust/device_vector.h>

struct greater_than_four
{
    __host__ __device__
    bool operator() (int x)
    {
        return x > 4;
    }
};

struct greater_than_ten
{
    __host__ __device__
    bool operator() (int x)
    {
        return x > 10;
    }
};

...
thrust::device_vector<int> input(4);

input[0] = 0;
input[1] = 5;
input[2] = 3;
input[3] = 7;

thrust::device_vector<int>::iterator iter;

iter = thrust::find_if_not(input.begin(), input.end(), greater_than_four()); //␣
↪returns input.first()

iter = thrust::find_if_not(input.begin(), input.end(), greater_than_ten()); //␣
↪returns input.first()
```

Return The first iterator `i` such that `pred(*i)` is false, or last.

Parameters

- `first`: Beginning of the sequence to search.
- `last`: End of the sequence to search.
- `pred`: A predicate used to test range elements.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#).
- `Predicate`: is a model of [Predicate](#).

See `find`

See `find_if`

See `mismatch`

Template Function `thrust::for_each(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, UnaryFunction)`

- Defined in `file_thrust_for_each.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::for_each`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, UnaryFunction)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename UnaryFunction>__
  ↳host__ __device__ InputIterator thrust::for_each(const thrust::detail::execution_
  ↳policy_base< DerivedPolicy > &, InputIterator, InputIterator, UnaryFunction)
- template<typename InputIterator, typename UnaryFunction>
  InputIterator thrust::for_each(InputIterator, InputIterator, UnaryFunction)
```

Template Function `thrust::for_each(InputIterator, InputIterator, UnaryFunction)`

Function Documentation

template<typename **InputIterator**, typename **UnaryFunction**>

InputIterator thrust::for_each(*InputIterator* first, *InputIterator* last, *UnaryFunction* f)

`for_each` applies the function object `f` to each element in the range `[first, last)`; `f`'s return value, if any, is ignored. Unlike the C++ Standard Template Library function `std::for_each`, this version offers no guarantee on order of execution. For this reason, this version of `for_each` does not return a copy of the function object.

The following code snippet demonstrates how to use `for_each` to print the elements of a *device_vector*.

Return last

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.
- `f`: The function object to apply to the range `[first, last)`.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `UnaryFunction`'s `argument_type`.
- `UnaryFunction`: is a model of [Unary Function](#), and `UnaryFunction` does not apply any non-constant operation through its argument.

```
#include <thrust/for_each.h>
#include <thrust/device_vector.h>
#include <stdio.h>

struct printf_funcutor
{
    __host__ __device__
```

(continues on next page)

(continued from previous page)

```

void operator() (int x)
{
    // note that using printf in a __device__ function requires
    // code compiled for a GPU with compute capability 2.0 or
    // higher (nvcc --arch=sm_20)
    printf("%d\n", x);
}
};
...
thrust::device_vector<int> d_vec(3);
d_vec[0] = 0; d_vec[1] = 1; d_vec[2] = 2;

thrust::for_each(d_vec.begin(), d_vec.end(), printf_function());

// 0 1 2 is printed to standard output in some unspecified order

```

See `for_each_n`

See http://www.sgi.com/tech/stl/for_each.html

Template Function `thrust::for_each_n(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, Size, UnaryFunction)`

- Defined in file `_thrust_for_each.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::for_each_n`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, Size, UnaryFunction)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator, typename Size, typename
  ↳UnaryFunction>__host__ __device__ InputIterator thrust::for_each_n(const
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator, Size,
  ↳UnaryFunction)
- template<typename InputIterator, typename Size, typename UnaryFunction>
  InputIterator thrust::for_each_n(InputIterator, Size, UnaryFunction)

```

Template Function `thrust::for_each_n(InputIterator, Size, UnaryFunction)`

Function Documentation

template<typename **InputIterator**, typename **Size**, typename **UnaryFunction**>
InputIterator thrust::for_each_n(*InputIterator first*, *Size n*, *UnaryFunction f*)

`for_each_n` applies the function object `f` to each element in the range `[first, first + n)`; `f`'s return value, if any, is ignored. Unlike the C++ Standard Template Library function `std::for_each`, this version offers no guarantee on order of execution.

The following code snippet demonstrates how to use `for_each_n` to print the elements of a `device_vector`.

Return `first + n`

Parameters

- `first`: The beginning of the sequence.
- `n`: The size of the input sequence.
- `f`: The function object to apply to the range `[first, first + n)`.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `UnaryFunction`'s `argument_type`.
- `Size`: is an integral type.
- `UnaryFunction`: is a model of [Unary Function](#), and `UnaryFunction` does not apply any non-constant operation through its argument.

```
#include <thrust/for_each.h>
#include <thrust/device_vector.h>
#include <stdio.h>

struct printf_funcutor
{
    __host__ __device__
    void operator()(int x)
    {
        // note that using printf in a __device__ function requires
        // code compiled for a GPU with compute capability 2.0 or
        // higher (nvcc --arch=sm_20)
        printf("%d\n", x);
    }
};

...
thrust::device_vector<int> d_vec(3);
d_vec[0] = 0; d_vec[1] = 1; d_vec[2] = 2;

thrust::for_each_n(d_vec.begin(), d_vec.size(), printf_funcutor());

// 0 1 2 is printed to standard output in some unspecified order
```

See `for_each`

See http://www.sgi.com/tech/stl/for_each.html

Template Function `thrust::free`

- Defined in `file_thrust_memory.h`

Function Documentation

template<typename DerivedPolicy, typename Pointer> __host__ __device__ void thrust::free(const Pointer& p)
`free` deallocates the storage previously allocated by `thrust::malloc`.

The following code snippet demonstrates how to use `free` to deallocate a range of memory previously allocated with `thrust::malloc`.

Pre `ptr` shall have been returned by a previous call to `thrust::malloc(system, n)` or `thrust::malloc<T>(system, n)` for some type `T`.

Parameters

- `system`: The Thrust system with which the storage is associated.
- `ptr`: A pointer previously returned by `thrust::malloc`. If `ptr` is null, `free` does nothing.

Template Parameters

- `DerivedPolicy`: The name of the derived execution policy.

```
#include <thrust/memory.h>
...
// allocate storage for 100 ints with thrust::malloc
const int N = 100;
thrust::device_system_tag device_sys;
thrust::pointer<int, thrust::device_system_tag> ptr = thrust::malloc<int>(device_
→sys, N);

// manipulate memory
...

// deallocate ptr with thrust::free
thrust::free(device_sys, ptr);
```

Template Function `thrust::gather(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, RandomAccessIterator, OutputIterator)`

- Defined in file `_thrust_gather.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::gather`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, RandomAccessIterator, OutputIterator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename
→RandomAccessIterator, typename OutputIterator>__host__ __device__ OutputIterator
→thrust::gather(const thrust::detail::execution_policy_base< DerivedPolicy > &,
→InputIterator, InputIterator, RandomAccessIterator, OutputIterator)
- template<typename InputIterator, typename RandomAccessIterator, typename
→OutputIterator>
  OutputIterator thrust::gather(InputIterator, InputIterator, RandomAccessIterator,
→OutputIterator)
```

Template Function `thrust::gather(InputIterator, InputIterator, RandomAccessIterator, OutputIterator)`

Function Documentation

template<typename **InputIterator**, typename **RandomAccessIterator**, typename **OutputIterator**>

OutputIterator thrust::gather(*InputIterator* map_first, *InputIterator* map_last, *RandomAccessIterator* input_first, *OutputIterator* result)

gather copies elements from a source array into a destination range according to a map. For each input iterator *i* in the range [map_first, map_last), the value input_first[*i] is assigned to *(result + (i - map_first)). RandomAccessIterator must permit random access.

The following code snippet demonstrates how to use gather to reorder a range.

Pre The range [map_first, map_last) shall not overlap the range [result, result + (map_last - map_first)).

Remark gather is the inverse of thrust::scatter.

Parameters

- map_first: Beginning of the range of gather locations.
- map_last: End of the range of gather locations.
- input_first: Beginning of the source range.
- result: Beginning of the destination range.

Template Parameters

- InputIterator: must be a model of [Input Iterator](#) and InputIterator's value_type must be convertible to RandomAccessIterator's difference_type.
- RandomAccessIterator: must be a model of [Random Access Iterator](#) and RandomAccessIterator's value_type must be convertible to OutputIterator's value_type.
- OutputIterator: must be a model of [Output Iterator](#).

```
#include <thrust/gather.h>
#include <thrust/device_vector.h>
...
// mark even indices with a 1; odd indices with a 0
int values[10] = {1, 0, 1, 0, 1, 0, 1, 0, 1, 0};
thrust::device_vector<int> d_values(values, values + 10);

// gather all even indices into the first half of the range
// and odd indices to the last half of the range
int map[10] = {0, 2, 4, 6, 8, 1, 3, 5, 7, 9};
thrust::device_vector<int> d_map(map, map + 10);

thrust::device_vector<int> d_output(10);
thrust::gather(d_map.begin(), d_map.end(),
              d_values.begin(),
              d_output.begin());
// d_output is now {1, 1, 1, 1, 1, 0, 0, 0, 0, 0}
```

Template Function thrust::gather_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator, OutputIterator)

- Defined in file_thrust_gather.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::gather_if” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator, OutputIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename RandomAccessIterator, typename OutputIterator, typename Predicate>__
  ↳ host__ __device__ OutputIterator thrust::gather_if(const_
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, RandomAccessIterator, OutputIterator, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename RandomAccessIterator, typename OutputIterator>__host__ __device__
  ↳ OutputIterator thrust::gather_if(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ RandomAccessIterator, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ RandomAccessIterator, typename OutputIterator, typename Predicate>
  ↳ OutputIterator thrust::gather_if(InputIterator1, InputIterator1, InputIterator2,
  ↳ RandomAccessIterator, OutputIterator, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ RandomAccessIterator, typename OutputIterator>
  ↳ OutputIterator thrust::gather_if(InputIterator1, InputIterator1, InputIterator2,
  ↳ RandomAccessIterator, OutputIterator)
```

Template Function thrust::gather_if(InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator, OutputIterator)

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename RandomAccessIterator, typename OutputIterator>
OutputIterator thrust::gather_if(InputIterator1 map_first, InputIterator1 map_last, InputIterator2
                                stencil, RandomAccessIterator input_first, OutputIterator result)
```

gather_if conditionally copies elements from a source array into a destination range according to a map. For each input iterator *i* in the range `[map_first, map_last)`, such that the value of `*(stencil + (i - map_first))` is true, the value `input_first[*i]` is assigned to `*(result + (i - map_first))`. RandomAccessIterator must permit random access.

The following code snippet demonstrates how to use gather_if to gather selected values from an input range.

Pre The range `[map_first, map_last)` shall not overlap the range `[result, result + (map_last - map_first))`.

Pre The range `[stencil, stencil + (map_last - map_first))` shall not overlap the range `[result, result + (map_last - map_first))`.

Remark gather_if is the inverse of scatter_if.

Parameters

- map_first: Beginning of the range of gather locations.
- map_last: End of the range of gather locations.
- stencil: Beginning of the range of predicate values.

- `input_first`: Beginning of the source range.
- `result`: Beginning of the destination range.

Template Parameters

- `InputIterator1`: must be a model of [Input Iterator](#) and `InputIterator1`'s `value_type` must be convertible to `RandomAccessIterator`'s `difference_type`.
- `InputIterator2`: must be a model of [Input Iterator](#) and `InputIterator2`'s `value_type` must be convertible to `bool`.
- `RandomAccessIterator`: must be a model of [Random Access iterator](#) and `RandomAccessIterator`'s `value_type` must be convertible to `OutputIterator`'s `value_type`.
- `OutputIterator`: must be a model of [Output Iterator](#).

```
#include <thrust/gather.h>
#include <thrust/device_vector.h>
...

int values[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
thrust::device_vector<int> d_values(values, values + 10);

// select elements at even-indexed locations
int stencil[10] = {1, 0, 1, 0, 1, 0, 1, 0, 1, 0};
thrust::device_vector<int> d_stencil(stencil, stencil + 10);

// map all even indices into the first half of the range
// and odd indices to the last half of the range
int map[10] = {0, 2, 4, 6, 8, 1, 3, 5, 7, 9};
thrust::device_vector<int> d_map(map, map + 10);

thrust::device_vector<int> d_output(10, 7);
thrust::gather_if(d_map.begin(), d_map.end(),
                 d_stencil.begin(),
                 d_values.begin(),
                 d_output.begin());
// d_output is now {0, 7, 4, 7, 8, 7, 3, 7, 7, 7}
```

Template Function `thrust::gather_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator, OutputIterator, Predicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::gather_if`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator, OutputIterator, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename RandomAccessIterator, typename OutputIterator, typename Predicate>__
  ↳ host__ __device__ OutputIterator thrust::gather_if(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, RandomAccessIterator, OutputIterator, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename RandomAccessIterator, typename OutputIterator>__host__ __device__
  ↳ OutputIterator thrust::gather_if(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ RandomAccessIterator, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ RandomAccessIterator, typename OutputIterator, typename Predicate>
  ↳ OutputIterator thrust::gather_if(InputIterator1, InputIterator1, InputIterator2,
  ↳ RandomAccessIterator, OutputIterator, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ RandomAccessIterator, typename OutputIterator>
  ↳ OutputIterator thrust::gather_if(InputIterator1, InputIterator1, InputIterator2,
  ↳ RandomAccessIterator, OutputIterator)

```

Template Function `thrust::gather_if(InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator, OutputIterator, Predicate)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **RandomAccessIterator**, typename **OutputIterator**
OutputIterator thrust::gather_if(*InputIterator1* map_first, *InputIterator1* map_last, *InputIterator2*
stencil, *RandomAccessIterator* input_first, *OutputIterator* result,
Predicate pred)

`gather_if` conditionally copies elements from a source array into a destination range according to a map. For each input iterator *i* in the range `[map_first, map_last)` such that the value of `pred(*(stencil + (i - map_first)))` is true, the value `input_first[*i]` is assigned to `*(result + (i - map_first))`. `RandomAccessIterator` must permit random access.

The following code snippet demonstrates how to use `gather_if` to gather selected values from an input range based on an arbitrary selection function.

Pre The range `[map_first, map_last)` shall not overlap the range `[result, result + (map_last - map_first))`.

Pre The range `[stencil, stencil + (map_last - map_first))` shall not overlap the range `[result, result + (map_last - map_first))`.

Remark `gather_if` is the inverse of `scatter_if`.

Parameters

- `map_first`: Beginning of the range of gather locations.
- `map_last`: End of the range of gather locations.
- `stencil`: Beginning of the range of predicate values.
- `input_first`: Beginning of the source range.
- `result`: Beginning of the destination range.
- `pred`: Predicate to apply to the stencil values.

Template Parameters

- InputIterator1: must be a model of [Input Iterator](#) and InputIterator1's value_type must be convertible to RandomAccessIterator's difference_type.
- InputIterator2: must be a model of [Input Iterator](#) and InputIterator2's value_type must be convertible to Predicate's argument_type.
- RandomAccessIterator: must be a model of [Random Access iterator](#) and RandomAccessIterator's value_type must be convertible to OutputIterator's value_type.
- OutputIterator: must be a model of [Output Iterator](#).
- Predicate: must be a model of [Predicate](#).

```
#include <thrust/gather.h>
#include <thrust/device_vector.h>

struct is_even
{
    __host__ __device__
    bool operator()(const int x)
    {
        return (x % 2) == 0;
    }
};

...

int values[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
thrust::device_vector<int> d_values(values, values + 10);

// we will select an element when our stencil is even
int stencil[10] = {0, 3, 4, 1, 4, 1, 2, 7, 8, 9};
thrust::device_vector<int> d_stencil(stencil, stencil + 10);

// map all even indices into the first half of the range
// and odd indices to the last half of the range
int map[10] = {0, 2, 4, 6, 8, 1, 3, 5, 7, 9};
thrust::device_vector<int> d_map(map, map + 10);

thrust::device_vector<int> d_output(10, 7);
thrust::gather_if(d_map.begin(), d_map.end(),
                 d_stencil.begin(),
                 d_values.begin(),
                 d_output.begin(),
                 is_even());
// d_output is now {0, 7, 4, 7, 8, 7, 3, 7, 7, 7}
```

Template Function `thrust::generate(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Generator)`

- Defined in file `thrust_generate.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::generate” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Generator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename Generator>__
  ↳host__ __device__ void thrust::generate(const thrust::detail::execution_policy_
  ↳base< DerivedPolicy > &, ForwardIterator, ForwardIterator, Generator)
- template<typename ForwardIterator, typename Generator>
  void thrust::generate(ForwardIterator, ForwardIterator, Generator)
```

Template Function thrust::generate(ForwardIterator, ForwardIterator, Generator)

Function Documentation

template<typename **ForwardIterator**, typename **Generator**>

void thrust::generate (ForwardIterator first, ForwardIterator last, Generator gen)

generate assigns the result of invoking gen, a function object that takes no arguments, to each element in the range [first, last).

The following code snippet demonstrates how to fill a *host_vector* with random numbers, using the standard C library function rand.

Parameters

- first: The first element in the range of interest.
- last: The last element in the range of interest.
- gen: A function argument, taking no parameters, used to generate values to assign to elements in the range [first, last).

Template Parameters

- ForwardIterator: is a model of **Forward Iterator**, and ForwardIterator is mutable.
- Generator: is a model of **Generator**, and Generator's result_type is convertible to ForwardIterator's value_type.

```
#include <thrust/generate.h>
#include <thrust/host_vector.h>
#include <thrust/execution_policy.h>
#include <cstdlib>
...
thrust::host_vector<int> v(10);
srand(13);
thrust::generate(v.begin(), v.end(), rand);

// the elements of v are now pseudo-random numbers
```

See generate_n

See <http://www.sgi.com/tech/stl/generate.html>

Template Function `thrust::generate_n(const thrust::detail::execution_policy_base<DerivedPolicy>&, OutputIterator, Size, Generator)`

- Defined in file `thrust_generate.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::generate_n`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, OutputIterator, Size, Generator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename OutputIterator, typename Size, typename
↳Generator>__host__ __device__ OutputIterator thrust::generate_n(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, OutputIterator, Size,
↳Generator)
- template<typename OutputIterator, typename Size, typename Generator>
  OutputIterator thrust::generate_n(OutputIterator, Size, Generator)
```

Template Function `thrust::generate_n(OutputIterator, Size, Generator)`

Function Documentation

template<typename **OutputIterator**, typename **Size**, typename **Generator**>

OutputIterator thrust::generate_n(*OutputIterator first*, *Size n*, *Generator gen*)

`generate_n` assigns the result of invoking `gen`, a function object that takes no arguments, to each element in the range `[first, first + n)`. The return value is `first + n`.

The following code snippet demonstrates how to fill a `host_vector` with random numbers, using the standard C library function `rand`.

Parameters

- `first`: The first element in the range of interest.
- `n`: The size of the range of interest.
- `gen`: A function argument, taking no parameters, used to generate values to assign to elements in the range `[first, first + n)`.

Template Parameters

- `OutputIterator`: is a model of [Output Iterator](#).
- `Size`: is an integral type (either signed or unsigned).
- `Generator`: is a model of [Generator](#), and `Generator`'s `result_type` is convertible to a type in `OutputIterator`'s set of `value_types`.

```
#include <thrust/generate.h>
#include <thrust/host_vector.h>
#include <stdlib.h>
...
thrust::host_vector<int> v(10);
srand(13);
thrust::generate_n(v.begin(), 10, rand);
```

(continues on next page)

(continued from previous page)

```
// the elements of v are now pseudo-random numbers
```

See `generate`

See <http://www.sgi.com/tech/stl/generate.html>

Template Function `thrust::get(detail::cons<HT, TT>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::get`” with arguments (`detail::cons<HT, TT>&`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<int N, class HT, class TT>__host__ __device__ access_traits< typename_
↳tuple_element<N, detail::cons<HT, TT> >::type >::const_type thrust::get(const_
↳detail::cons< HT, TT > &)
- template<int N, class HT, class TT>__host__ __device__ access_traits< typename_
↳tuple_element<N, detail::cons<HT, TT> >::type >::non_const_type_
↳thrust::get(detail::cons< HT, TT > &)
```

Template Function `thrust::get(const detail::cons<HT, TT>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::get`” with arguments (`const detail::cons<HT, TT>&`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<int N, class HT, class TT>__host__ __device__ access_traits< typename_
↳tuple_element<N, detail::cons<HT, TT> >::type >::const_type thrust::get(const_
↳detail::cons< HT, TT > &)
- template<int N, class HT, class TT>__host__ __device__ access_traits< typename_
↳tuple_element<N, detail::cons<HT, TT> >::type >::non_const_type_
↳thrust::get(detail::cons< HT, TT > &)
```

Template Function `thrust::get_temporary_buffer`

- Defined in `file_thrust_memory.h`

Function Documentation

template<typename T, typename DerivedPolicy> __host__ __device__ thrust::pair<thrust::point

`get_temporary_buffer` returns a pointer to storage associated with a given Thrust system sufficient to store up to `n` objects of type `T`. If not enough storage is available to accomodate `n` objects, an implementation may return a smaller buffer. The number of objects the returned buffer can accomodate is also returned.

Thrust uses `get_temporary_buffer` internally when allocating temporary storage required by algorithm implementations.

The storage allocated with `get_temporary_buffer` must be returned to the system with `return_temporary_buffer`.

The following code snippet demonstrates how to use `get_temporary_buffer` to allocate a range of memory to accomodate integers associated with Thrust's device system.

Return A pair `p` such that `p.first` is a pointer to the allocated storage and `p.second` is the number of contiguous objects of type `T` that the storage can accomodate. If no storage can be allocated, `p.first` if no storage can be obtained. The storage must be returned to the system using `return_temporary_buffer`.

Pre `DerivedPolicy` must be publically derived from `thrust::execution_policy<DerivedPolicy>`.

Parameters

- `system`: The Thrust system with which to associate the storage.
- `n`: The requested number of objects of type `T` the storage should accomodate.

Template Parameters

- `DerivedPolicy`: The name of the derived execution policy.

```
#include <thrust/memory.h>
...
// allocate storage for 100 ints with thrust::get_temporary_buffer
const int N = 100;

typedef thrust::pair<
    thrust::pointer<int, thrust::device_system_tag>,
    std::ptrdiff_t
> ptr_and_size_t;

thrust::device_system_tag device_sys;
ptr_and_size_t ptr_and_size = thrust::get_temporary_buffer<int>(device_sys, N);

// manipulate up to 100 ints
for(int i = 0; i < ptr_and_size.second; ++i)
{
    *ptr_and_size.first = i;
}

// deallocate storage with thrust::return_temporary_buffer
thrust::return_temporary_buffer(device_sys, ptr_and_size.first);
```

See `malloc`

See `return_temporary_buffer`

Template Function `thrust::inclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator)`

- Defined in `file_thrust_scan.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::inclusive_scan” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
↳typename AssociativeOperator>__host__ __device__ OutputIterator thrust::inclusive_
↳scan(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳InputIterator, InputIterator, OutputIterator, AssociativeOperator)
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator>_
↳__host__ __device__ OutputIterator thrust::inclusive_scan(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
↳InputIterator, OutputIterator)
- template<typename InputIterator, typename OutputIterator, typename
↳AssociativeOperator>
  OutputIterator thrust::inclusive_scan(InputIterator, InputIterator,
↳OutputIterator, AssociativeOperator)
- template<typename InputIterator, typename OutputIterator>
  OutputIterator thrust::inclusive_scan(InputIterator, InputIterator,
↳OutputIterator)
```

Template Function thrust::inclusive_scan(InputIterator, InputIterator, OutputIterator)

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**>

OutputIterator thrust::inclusive_scan(*InputIterator first*, *InputIterator last*, *OutputIterator result*)

inclusive_scan computes an inclusive prefix sum operation. The term ‘inclusive’ means that each result includes the corresponding input operand in the partial sum. More precisely, **first* is assigned to **result* and the sum of **first* and **(first + 1)* is assigned to **(result + 1)*, and so on. This version of inclusive_scan assumes plus as the associative operator. When the input and output sequences are the same, the scan is performed in-place.

inclusive_scan is similar to std::partial_sum in the STL. The primary difference between the two functions is that std::partial_sum guarantees a serial summation order, while inclusive_scan requires associativity of the binary operation to parallelize the prefix sum.

The following code snippet demonstrates how to use inclusive_scan

Return The end of the output sequence.

Pre *first* may equal *result* but the range [*first*, *last*) and the range [*result*, *result* + (*last* - *first*)) shall not overlap otherwise.

Parameters

- *first*: The beginning of the input sequence.
- *last*: The end of the input sequence.
- *result*: The beginning of the output sequence.

Template Parameters

- **InputIterator**: is a model of [Input Iterator](#) and InputIterator's value_type is convertible to OutputIterator's value_type.

- `OutputIterator`: is a model of `Output Iterator`, and if `x` and `y` are objects of `OutputIterator`'s `value_type`, then `x + y` is defined. If `T` is `OutputIterator`'s `value_type`, then `T(0)` is defined.

```
#include <thrust/scan.h>

int data[6] = {1, 0, 2, 2, 1, 3};

thrust::inclusive_scan(data, data + 6, data); // in-place scan

// data is now {1, 1, 3, 5, 6, 9}
```

See http://www.sgi.com/tech/stl/partial_sum.html

Template Function `thrust::inclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, AssociativeOperator)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::inclusive_scan” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, AssociativeOperator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
  ↳typename AssociativeOperator>__host__ __device__ OutputIterator thrust::inclusive_
  ↳scan(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳InputIterator, InputIterator, OutputIterator, AssociativeOperator)
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator>_
  ↳__host__ __device__ OutputIterator thrust::inclusive_scan(const
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳InputIterator, OutputIterator)
- template<typename InputIterator, typename OutputIterator, typename
  ↳AssociativeOperator>
  ↳OutputIterator thrust::inclusive_scan(InputIterator, InputIterator,
  ↳OutputIterator, AssociativeOperator)
- template<typename InputIterator, typename OutputIterator>
  ↳OutputIterator thrust::inclusive_scan(InputIterator, InputIterator,
  ↳OutputIterator)
```

Template Function `thrust::inclusive_scan(InputIterator, InputIterator, OutputIterator, AssociativeOperator)`

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**, typename **AssociativeOperator**>
OutputIterator thrust::inclusive_scan(*InputIterator* first, *InputIterator* last, *OutputIterator* result,
AssociativeOperator binary_op)

`inclusive_scan` computes an inclusive prefix sum operation. The term ‘inclusive’ means that each result includes the corresponding input operand in the partial sum. When the input and output sequences are the same, the scan is performed in-place.

`inclusive_scan` is similar to `std::partial_sum` in the STL. The primary difference between the two functions is that `std::partial_sum` guarantees a serial summation order, while `inclusive_scan` requires associativity of the binary operation to parallelize the prefix sum.

The following code snippet demonstrates how to use `inclusive_scan`

Return The end of the output sequence.

Pre `first` may equal `result` but the range `[first, last)` and the range `[result, result + (last - first))` shall not overlap otherwise.

Parameters

- `first`: The beginning of the input sequence.
- `last`: The end of the input sequence.
- `result`: The beginning of the output sequence.
- `binary_op`: The associative operator used to ‘sum’ values.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#) and `InputIterator`'s `value_type` is convertible to `OutputIterator`'s `value_type`.
- `OutputIterator`: is a model of [Output Iterator](#) and `OutputIterator`'s `value_type` is convertible to both `AssociativeOperator`'s `first_argument_type` and `second_argument_type`.
- `AssociativeOperator`: is a model of [Binary Function](#) and `AssociativeOperator`'s `result_type` is convertible to `OutputIterator`'s `value_type`.

```
int data[10] = {-5, 0, 2, -3, 2, 4, 0, -1, 2, 8};

thrust::maximum<int> binary_op;

thrust::inclusive_scan(data, data + 10, data, binary_op); // in-place scan

// data is now {-5, 0, 2, 2, 2, 4, 4, 4, 4, 8}
```

See http://www.sgi.com/tech/stl/partial_sum.html

Template Function `thrust::inclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator)`

- Defined in `file_thrust_scan.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::inclusive_scan_by_key`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename BinaryPredicate, typename AssociativeOperator>_
  ↳ _host__ __device__ OutputIterator thrust::inclusive_scan_by_key(const_
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, OutputIterator, BinaryPredicate,
  ↳ AssociativeOperator)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename BinaryPredicate>__host__ __device__
  ↳ OutputIterator thrust::inclusive_scan_by_key(const thrust::detail::execution_
  ↳ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::inclusive_
  ↳ scan_by_key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳ OutputIterator, typename BinaryPredicate, typename AssociativeOperator>
  ↳ OutputIterator thrust::inclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator, BinaryPredicate, AssociativeOperator)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳ OutputIterator, typename BinaryPredicate>
  ↳ OutputIterator thrust::inclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳ OutputIterator>
  ↳ OutputIterator thrust::inclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳ InputIterator2, OutputIterator)

```

Template Function `thrust::inclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**>
OutputIterator thrust::inclusive_scan_by_key(*InputIterator1 first1, InputIterator1 last1, InputIt-*
erator2 first2, OutputIterator result)

`inclusive_scan_by_key` computes an inclusive key-value or ‘segmented’ prefix sum operation. The term ‘inclusive’ means that each result includes the corresponding input operand in the partial sum. The term ‘segmented’ means that the partial sums are broken into distinct segments. In other words, within each segment a separate inclusive scan operation is computed. Refer to the code sample below for example usage.

This version of `inclusive_scan_by_key` assumes `equal_to` as the binary predicate used to compare adjacent keys. Specifically, consecutive iterators `i` and `i+1` in the range `[first1, last1)` belong to the same segment if `*i == *(i+1)`, and belong to different segments otherwise.

This version of `inclusive_scan_by_key` assumes `plus` as the associative operator used to perform the prefix sum. When the input and output sequences are the same, the scan is performed in-place.

The following code snippet demonstrates how to use `inclusive_scan_by_key`

Return The end of the output sequence.

Pre `first1` may equal `result` but the range `[first1, last1)` and the range `[result, result + (last1 - first1))` shall not overlap otherwise.

Pre `first2` may equal `result` but the range `[first2, first2 + (last1 - first1))` and range `[result, result + (last1 - first1))` shall not overlap otherwise.

Parameters

- `first1`: The beginning of the key sequence.
- `last1`: The end of the key sequence.
- `first2`: The beginning of the input value sequence.
- `result`: The beginning of the output value sequence.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#)
- `InputIterator2`: is a model of [Input Iterator](#) and `InputIterator2`'s `value_type` is convertible to `OutputIterator`'s `value_type`.
- `OutputIterator`: is a model of [Output Iterator](#), and if `x` and `y` are objects of `OutputIterator`'s `value_type`, then `binary_op(x, y)` is defined.

```
#include <thrust/scan.h>

int data[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
int keys[10] = {0, 0, 0, 1, 1, 2, 3, 3, 3, 3};

thrust::inclusive_scan_by_key(keys, keys + 10, vals, vals); // in-place scan

// data is now {1, 2, 3, 1, 2, 1, 1, 2, 3, 4};
```

See `inclusive_scan`

See `exclusive_scan_by_key`

Template Function `thrust::inclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::inclusive_scan_by_key`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryPredicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename BinaryPredicate, typename AssociativeOperator>_
  ↳_host__ __device__ OutputIterator thrust::inclusive_scan_by_key(const_
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳InputIterator1, InputIterator2, OutputIterator, BinaryPredicate,
  ↳AssociativeOperator)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename BinaryPredicate>_host__ __device__
  ↳OutputIterator thrust::inclusive_scan_by_key(const thrust::detail::execution_
  ↳policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳OutputIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>_host__ __device__ OutputIterator thrust::inclusive_
  ↳scan_by_key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳InputIterator1, InputIterator1, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳OutputIterator, typename BinaryPredicate, typename AssociativeOperator>
  ↳OutputIterator thrust::inclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳InputIterator2, OutputIterator, BinaryPredicate, AssociativeOperator)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳OutputIterator, typename BinaryPredicate>
  ↳OutputIterator thrust::inclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳InputIterator2, OutputIterator, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳OutputIterator>
  ↳OutputIterator thrust::inclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳InputIterator2, OutputIterator)

```

Template Function `thrust::inclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryPredicate)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**, typename **BinaryPredicate**, typename **AssociativeOperator**>
OutputIterator thrust::inclusive_scan_by_key(*InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, OutputIterator result, BinaryPredicate binary_pred*)

`inclusive_scan_by_key` computes an inclusive key-value or ‘segmented’ prefix sum operation. The term ‘inclusive’ means that each result includes the corresponding input operand in the partial sum. The term ‘segmented’ means that the partial sums are broken into distinct segments. In other words, within each segment a separate inclusive scan operation is computed. Refer to the code sample below for example usage.

This version of `inclusive_scan_by_key` uses the binary predicate `pred` to compare adjacent keys. Specifically, consecutive iterators `i` and `i+1` in the range `[first1, last1)` belong to the same segment if `binary_pred(*i, *(i+1))` is true, and belong to different segments otherwise.

This version of `inclusive_scan_by_key` assumes plus as the associative operator used to perform the prefix sum. When the input and output sequences are the same, the scan is performed in-place.

The following code snippet demonstrates how to use `inclusive_scan_by_key`

Return The end of the output sequence.

Pre `first1` may equal `result` but the range `[first1, last1)` and the range `[result, result + (last1 - first1))` shall not overlap otherwise.

Pre first2 may equal result but the range [first2, first2 + (last1 - first1) and range [result, result + (last1 - first1)) shall not overlap otherwise.

Parameters

- first1: The beginning of the key sequence.
- last1: The end of the key sequence.
- first2: The beginning of the input value sequence.
- result: The beginning of the output value sequence.
- binary_pred: The binary predicate used to determine equality of keys.

Template Parameters

- InputIterator1: is a model of [Input Iterator](#)
- InputIterator2: is a model of [Input Iterator](#) and InputIterator2's value_type is convertible to OutputIterator's value_type.
- OutputIterator: is a model of [Output Iterator](#), and if x and y are objects of OutputIterator's value_type, then binary_op(x, y) is defined.
- BinaryPredicate: is a model of [Binary Predicate](#).

```
#include <thrust/scan.h>
#include <thrust/functional.h>

int data[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
int keys[10] = {0, 0, 0, 1, 1, 2, 3, 3, 3, 3};

thrust::equal_to<int> binary_pred;

thrust::inclusive_scan_by_key(keys, keys + 10, vals, vals, binary_pred); // in-
→place scan

// data is now {1, 2, 3, 1, 2, 1, 1, 2, 3, 4};
```

See `inclusive_scan`

See `exclusive_scan_by_key`

Template Function `thrust::inclusive_scan_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryPredicate, AssociativeOperator)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::inclusive_scan_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryPredicate, AssociativeOperator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename BinaryPredicate, typename AssociativeOperator>_
  ↳_host__ __device__ OutputIterator thrust::inclusive_scan_by_key(const_
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳InputIterator1, InputIterator2, OutputIterator, BinaryPredicate,
  ↳AssociativeOperator)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename BinaryPredicate>_host__ __device__
  ↳OutputIterator thrust::inclusive_scan_by_key(const thrust::detail::execution_
  ↳policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳OutputIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>_host__ __device__ OutputIterator thrust::inclusive_
  ↳scan_by_key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳InputIterator1, InputIterator1, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳OutputIterator, typename BinaryPredicate, typename AssociativeOperator>
  ↳OutputIterator thrust::inclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳InputIterator2, OutputIterator, BinaryPredicate, AssociativeOperator)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳OutputIterator, typename BinaryPredicate>
  ↳OutputIterator thrust::inclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳InputIterator2, OutputIterator, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳OutputIterator>
  ↳OutputIterator thrust::inclusive_scan_by_key(InputIterator1, InputIterator1,
  ↳InputIterator2, OutputIterator)

```

Template Function `thrust::inclusive_scan_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryPredicate, AssociativeOperator)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**, typename **BinaryPredicate**, typename **AssociativeOperator**>
OutputIterator thrust::inclusive_scan_by_key(*InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, OutputIterator result, BinaryPredicate binary_pred, AssociativeOperator binary_op*)

`inclusive_scan_by_key` computes an inclusive key-value or ‘segmented’ prefix sum operation. The term ‘inclusive’ means that each result includes the corresponding input operand in the partial sum. The term ‘segmented’ means that the partial sums are broken into distinct segments. In other words, within each segment a separate inclusive scan operation is computed. Refer to the code sample below for example usage.

This version of `inclusive_scan_by_key` uses the binary predicate `pred` to compare adjacent keys. Specifically, consecutive iterators `i` and `i+1` in the range `[first1, last1)` belong to the same segment if `binary_pred(*i, *(i+1))` is true, and belong to different segments otherwise.

This version of `inclusive_scan_by_key` uses the associative operator `binary_op` to perform the prefix sum. When the input and output sequences are the same, the scan is performed in-place.

The following code snippet demonstrates how to use `inclusive_scan_by_key`

Return The end of the output sequence.

Pre `first1` may equal `result` but the range `[first1, last1)` and the range `[result, result + (last1 - first1))` shall not overlap otherwise.

Pre first2 may equal result but the range [first2, first2 + (last1 - first1) and range [result, result + (last1 - first1)) shall not overlap otherwise.

Parameters

- first1: The beginning of the key sequence.
- last1: The end of the key sequence.
- first2: The beginning of the input value sequence.
- result: The beginning of the output value sequence.
- binary_pred: The binary predicate used to determine equality of keys.
- binary_op: The associative operator used to 'sum' values.

Template Parameters

- InputIterator1: is a model of [Input Iterator](#)
- InputIterator2: is a model of [Input Iterator](#) and InputIterator2's value_type is convertible to OutputIterator's value_type.
- OutputIterator: is a model of [Output Iterator](#), and if x and y are objects of OutputIterator's value_type, then binary_op(x, y) is defined.
- BinaryPredicate: is a model of [Binary Predicate](#).
- AssociativeOperator: is a model of [Binary Function](#) and AssociativeOperator's result_type is convertible to OutputIterator's value_type.

```
#include <thrust/scan.h>
#include <thrust/functional.h>

int data[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
int keys[10] = {0, 0, 0, 1, 1, 2, 3, 3, 3, 3};

thrust::equal_to<int> binary_pred;
thrust::plus<int>      binary_op;

thrust::inclusive_scan_by_key(keys, keys + 10, vals, vals, binary_pred, binary_
→op); // in-place scan

// data is now {1, 2, 3, 1, 2, 1, 1, 2, 3, 4};
```

See `inclusive_scan`

See `exclusive_scan_by_key`

Template Function `thrust::inner_product(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputType)`

- Defined in file `_thrust_inner_product.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::inner_product” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputType) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputType, typename BinaryFunction1, typename BinaryFunction2>__host__
  ↳ __device__ OutputType thrust::inner_product(const thrust::detail::execution_
  ↳ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputType, BinaryFunction1, BinaryFunction2)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputType>__host__ __device__ OutputType thrust::inner_product(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, OutputType)
- template<typename InputIterator1, typename InputIterator2, typename OutputType,
  ↳ typename BinaryFunction1, typename BinaryFunction2>
  OutputType thrust::inner_product(InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputType, BinaryFunction1, BinaryFunction2)
- template<typename InputIterator1, typename InputIterator2, typename OutputType>
  OutputType thrust::inner_product(InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputType)
```

Template Function thrust::inner_product(InputIterator1, InputIterator1, InputIterator2, OutputType)

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputType**>
OutputType thrust::inner_product(*InputIterator1 first1*, *InputIterator1 last1*, *InputIterator2 first2*,
OutputType init)

inner_product calculates an inner product of the ranges [first1, last1) and [first2, first2 + (last1 - first1)).

Specifically, this version of inner_product computes the sum $\text{init} + (*\text{first1} * *\text{first2}) + (*(\text{first1}+1) * *(\text{first2}+1)) + \dots$

Unlike the C++ Standard Template Library function `std::inner_product`, this version offers no guarantee on order of execution.

The following code demonstrates how to use inner_product to compute the dot product of two vectors.

Return The inner product of sequences [first1, last1) and [first2, last2) plus init.

Parameters

- first1: The beginning of the first sequence.
- last1: The end of the first sequence.
- first2: The beginning of the second sequence.
- init: Initial value of the result.

Template Parameters

- InputIterator1: is a model of [Input Iterator](#),
- InputIterator2: is a model of [Input Iterator](#),

- `OutputType`: is a model of `Assignable`, and if `x` is an object of type `OutputType`, and `y` is an object of `InputIterator1`'s `value_type`, and `z` is an object of `InputIterator2`'s `value_type`, then `x + y * z` is defined and is convertible to `OutputType`.

```
#include <thrust/inner_product.h>
...
float vec1[3] = {1.0f, 2.0f, 5.0f};
float vec2[3] = {4.0f, 1.0f, 5.0f};

float result = thrust::inner_product(vec1, vec1 + 3, vec2, 0.0f);

// result == 31.0f
```

See http://www.sgi.com/tech/stl/inner_product.html

Template Function `thrust::inner_product(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputType, BinaryFunction1, BinaryFunction2)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::inner_product`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `InputIterator1`, `InputIterator1`, `InputIterator2`, `OutputType`, `BinaryFunction1`, `BinaryFunction2`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputType, typename BinaryFunction1, typename BinaryFunction2>__host__
  ↳ __device__ OutputType thrust::inner_product(const thrust::detail::execution_
  ↳ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputType, BinaryFunction1, BinaryFunction2)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputType>__host__ __device__ OutputType thrust::inner_product(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, OutputType)
- template<typename InputIterator1, typename InputIterator2, typename OutputType,
  ↳ typename BinaryFunction1, typename BinaryFunction2>
  ↳ OutputType thrust::inner_product(InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputType, BinaryFunction1, BinaryFunction2)
- template<typename InputIterator1, typename InputIterator2, typename OutputType>
  ↳ OutputType thrust::inner_product(InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputType)
```

Template Function `thrust::inner_product(InputIterator1, InputIterator1, InputIterator2, OutputType, BinaryFunction1, BinaryFunction2)`

Function Documentation

`template<typename InputIterator1, typename InputIterator2, typename OutputType, typename BinaryFunction1, OutputType thrust::inner_product (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, OutputType init, BinaryFunction1 binary_op1, BinaryFunction2 binary_op2)`
`inner_product` calculates an inner product of the ranges `[first1, last1)` and `[first2, first2`

```
+ (last1 - first1)).
```

This version of `inner_product` is identical to the first, except that it uses two user-supplied function objects instead of `operator+` and `operator*`.

Specifically, this version of `inner_product` computes the sum `binary_op1(init, binary_op2(*first1, *first2)), ...`

Unlike the C++ Standard Template Library function `std::inner_product`, this version offers no guarantee on order of execution.

```
#include <thrust/inner_product.h>
...
float vec1[3] = {1.0f, 2.0f, 5.0f};
float vec2[3] = {4.0f, 1.0f, 5.0f};

float init = 0.0f;
thrust::plus<float>      binary_op1;
thrust::multiplies<float> binary_op2;

float result = thrust::inner_product(vec1, vec1 + 3, vec2, init, binary_op1,
↪binary_op2);

// result == 31.0f
```

Return The inner product of sequences `[first1, last1)` and `[first2, last2)`.

Parameters

- `first1`: The beginning of the first sequence.
- `last1`: The end of the first sequence.
- `first2`: The beginning of the second sequence.
- `init`: Initial value of the result.
- `binary_op1`: Generalized addition operation.
- `binary_op2`: Generalized multiplication operation.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), and `InputIterator1`'s `value_type` is convertible to `BinaryFunction2`'s `first_argument_type`.
- `InputIterator2`: is a model of [Input Iterator](#). and `InputIterator2`'s `value_type` is convertible to `BinaryFunction2`'s `second_argument_type`.
- `OutputType`: is a model of [Assignable](#), and `OutputType` is convertible to `BinaryFunction1`'s `first_argument_type`.
- `BinaryFunction1`: is a model of [Binary Function](#), and `BinaryFunction1`'s `return_type` is convertible to `OutputType`.
- `BinaryFunction2`: is a model of [Binary Function](#), and `BinaryFunction2`'s `return_type` is convertible to `BinaryFunction1`'s `second_argument_type`.

See http://www.sgi.com/tech/stl/inner_product.html

Template Function `thrust::is_partitioned(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`

- Defined in `file_thrust_partition.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::is_partitioned`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename Predicate>__
  ↪host__ __device__ bool thrust::is_partitioned(const thrust::detail::execution_
  ↪policy_base< DerivedPolicy > &, InputIterator, InputIterator, Predicate)
- template<typename InputIterator, typename Predicate>
  bool thrust::is_partitioned(InputIterator, InputIterator, Predicate)
```

Template Function `thrust::is_partitioned(InputIterator, InputIterator, Predicate)`**Function Documentation**

```
template<typename InputIterator, typename Predicate>
```

```
bool thrust::is_partitioned(InputIterator first, InputIterator last, Predicate pred)
```

`is_partitioned` returns true if the given range is partitioned with respect to a predicate, and false otherwise.

Specifically, `is_partitioned` returns true if `[first, last)` is empty or if `[first, last)` is partitioned by `pred`, i.e. if all elements that satisfy `pred` appear before those that do not.

```
#include <thrust/partition.h>

struct is_even
{
    __host__ __device__
    bool operator()(const int &x)
    {
        return (x % 2) == 0;
    }
};

...

int A[] = {2, 4, 6, 8, 10, 1, 3, 5, 7, 9};
int B[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

thrust::is_partitioned(A, A + 10, is_even()); // returns true
thrust::is_partitioned(B, B + 10, is_even()); // returns false
```

Return true if the range `[first, last)` is partitioned with respect to `pred`, or if `[first, last)` is empty. false, otherwise.

Parameters

- `first`: The beginning of the range to consider.
- `last`: The end of the range to consider.
- `pred`: A function object which decides to which partition each element of the range `[first, last)` belongs.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `Predicate`: is a model of [Predicate](#).

See `partition`

Template Function `thrust::is_sorted(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`

- Defined in file `thrust_sort.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::is_sorted`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename Compare>__
  ↳host__ __device__ bool thrust::is_sorted(const thrust::detail::execution_policy_
  ↳base< DerivedPolicy > &, ForwardIterator, ForwardIterator, Compare)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
  ↳bool thrust::is_sorted(const thrust::detail::execution_policy_base< DerivedPolicy_
  ↳> &, ForwardIterator, ForwardIterator)
- template<typename ForwardIterator, typename Compare>
  bool thrust::is_sorted(ForwardIterator, ForwardIterator, Compare)
- template<typename ForwardIterator>
  bool thrust::is_sorted(ForwardIterator, ForwardIterator)
```

Template Function `thrust::is_sorted(ForwardIterator, ForwardIterator)`

Function Documentation

template<typename **ForwardIterator**>

bool thrust::is_sorted(*ForwardIterator first, ForwardIterator last*)

`is_sorted` returns true if the range `[first, last)` is sorted in ascending order, and false otherwise.

Specifically, this version of `is_sorted` returns false if for some iterator `i` in the range `[first, last - 1)` the expression `*(i + 1) < *i` is true.

The following code demonstrates how to use `is_sorted` to test whether the contents of a `device_vector` are stored in ascending order.

Return true, if the sequence is sorted; false, otherwise.

Parameters

- first: The beginning of the sequence.
- last: The end of the sequence.

Template Parameters

- ForwardIterator: is a model of [Forward Iterator](#), ForwardIterator's value_type is a model of [LessThan Comparable](#), and the ordering on objects of ForwardIterator's value_type is a *strict weak ordering*, as defined in the [LessThan Comparable](#) requirements.

```
#include <thrust/sort.h>
#include <thrust/device_vector.h>
#include <thrust/sort.h>
...
thrust::device_vector<int> v(6);
v[0] = 1;
v[1] = 4;
v[2] = 2;
v[3] = 8;
v[4] = 5;
v[5] = 7;

bool result = thrust::is_sorted(v.begin(), v.end());

// result == false

thrust::sort(v.begin(), v.end());
result = thrust::is_sorted(v.begin(), v.end());

// result == true
```

See http://www.sgi.com/tech/stl/is_sorted.html

See `is_sorted_until`

See `sort`

See `stable_sort`

See `less<T>`

Template Function `thrust::is_sorted(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Compare)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::is_sorted” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Compare) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename Compare>__
  ↳host__ __device__ bool thrust::is_sorted(const thrust::detail::execution_policy_
  ↳base< DerivedPolicy > &, ForwardIterator, ForwardIterator, Compare)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
  ↳bool thrust::is_sorted(const thrust::detail::execution_policy_base< DerivedPolicy_
  ↳> &, ForwardIterator, ForwardIterator)
- template<typename ForwardIterator, typename Compare>
  ↳bool thrust::is_sorted(ForwardIterator, ForwardIterator, Compare)
- template<typename ForwardIterator>
  ↳bool thrust::is_sorted(ForwardIterator, ForwardIterator)
```

Template Function `thrust::is_sorted(ForwardIterator, ForwardIterator, Compare)`

Function Documentation

template<typename **ForwardIterator**, typename **Compare**>

bool thrust::is_sorted(*ForwardIterator first, ForwardIterator last, Compare comp*)

`is_sorted` returns true if the range `[first, last)` is sorted in ascending order according to a user-defined comparison operation, and false otherwise.

Specifically, this version of `is_sorted` returns false if for some iterator `i` in the range `[first, last - 1)` the expression `comp(*(i + 1), *i)` is true.

The following code snippet demonstrates how to use `is_sorted` to test whether the contents of a `device_vector` are stored in descending order.

Return true, if the sequence is sorted according to `comp`; false, otherwise.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.
- `comp`: Comparison operator.

Template Parameters

- `ForwardIterator`: is a model of `Forward Iterator`, and `ForwardIterator`'s `value_type` is convertible to both `StrictWeakOrdering`'s `first_argument_type` and `second_argument_type`.
- `Compare`: is a model of `Strict Weak Ordering`.

```
#include <thrust/sort.h>
#include <thrust/functional.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> v(6);
v[0] = 1;
v[1] = 4;
v[2] = 2;
v[3] = 8;
v[4] = 5;
v[5] = 7;

thrust::greater<int> comp;
bool result = thrust::is_sorted(v.begin(), v.end(), comp);

// result == false

thrust::sort(v.begin(), v.end(), comp);
result = thrust::is_sorted(v.begin(), v.end(), comp);

// result == true
```

See http://www.sgi.com/tech/stl/is_sorted.html

See `sort`

See `stable_sort`

See `less<T>`

Template Function `thrust::is_sorted_until(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`

- Defined in file `thrust_sort.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::is_sorted_until`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename Compare>__
  ↳host__ __device__ ForwardIterator thrust::is_sorted_until(const_
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ForwardIterator, Compare)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
  ↳ForwardIterator thrust::is_sorted_until(const thrust::detail::execution_policy_
  ↳base< DerivedPolicy > &, ForwardIterator, ForwardIterator)
- template<typename ForwardIterator, typename Compare>
  ForwardIterator thrust::is_sorted_until(ForwardIterator, ForwardIterator, Compare)
- template<typename ForwardIterator>
  ForwardIterator thrust::is_sorted_until(ForwardIterator, ForwardIterator)
```

Template Function `thrust::is_sorted_until(ForwardIterator, ForwardIterator)`

Function Documentation

template<typename **ForwardIterator**>

ForwardIterator `thrust::is_sorted_until (ForwardIterator first, ForwardIterator last)`

This version of `is_sorted_until` returns the last iterator `i` in `[first, last]` for which the range `[first, last)` is sorted using operator`<`. If `distance(first, last) < 2`, `is_sorted_until` simply returns `last`.

The following code snippet demonstrates how to use `is_sorted_until` to find the first position in an array where the data becomes unsorted:

Return The last iterator in the input range for which it is sorted.

Parameters

- `first`: The beginning of the range of interest.
- `last`: The end of the range of interest.

Template Parameters

- `ForwardIterator`: is a model of `Forward Iterator` and `ForwardIterator`'s `value_type` is a model of `LessThan Comparable`.


```
#include <thrust/sort.h>

...

int A[8] = {0, 1, 2, 3, 0, 1, 2, 3};

int * B = thrust::is_sorted_until(A, A + 8);

// B - A is 4
// [A, B) is sorted
```

See `is_sorted`

See `sort`

See `sort_by_key`

See `stable_sort`

See `stable_sort_by_key`

Template Function `thrust::is_sorted_until(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Compare)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::is_sorted_until” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Compare) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename Compare>__
  ↳host__ __device__ ForwardIterator thrust::is_sorted_until(const_
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ForwardIterator, Compare)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
  ↳ForwardIterator thrust::is_sorted_until(const thrust::detail::execution_policy_
  ↳base< DerivedPolicy > &, ForwardIterator, ForwardIterator)
- template<typename ForwardIterator, typename Compare>
  ↳ForwardIterator thrust::is_sorted_until(ForwardIterator, ForwardIterator, Compare)
- template<typename ForwardIterator>
  ↳ForwardIterator thrust::is_sorted_until(ForwardIterator, ForwardIterator)
```

Template Function `thrust::is_sorted_until(ForwardIterator, ForwardIterator, Compare)`

Function Documentation

template<typename **ForwardIterator**, typename **Compare**>

ForwardIterator thrust::is_sorted_until(*ForwardIterator* first, *ForwardIterator* last, *Compare* comp)

This version of `is_sorted_until` returns the last iterator `i` in `[first,last]` for which the range `[first,last)` is sorted using the function object `comp`. If `distance(first,last) < 2`, `is_sorted_until` simply returns `last`.

The following code snippet demonstrates how to use `is_sorted_until` to find the first position in an array where the data becomes unsorted in descending order:

Return The last iterator in the input range for which it is sorted.

Parameters

- `first`: The beginning of the range of interest.
- `last`: The end of the range of interest.
- `comp`: The function object to use for comparison.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#) and `ForwardIterator`'s `value_type` is convertible to `Compare`'s `argument_type`.
- `Compare`: is a model of [Strict Weak Ordering](#).

```
#include <thrust/sort.h>
#include <thrust/functional.h>

...

int A[8] = {3, 2, 1, 0, 3, 2, 1, 0};

thrust::greater<int> comp;
int * B = thrust::is_sorted_until(A, A + 8, comp);

// B - A is 4
// [A, B) is sorted in descending order
```

See `is_sorted`

See `sort`

See `sort_by_key`

See `stable_sort`

See `stable_sort_by_key`

Template Function `thrust::log`

- Defined in `file_thrust_complex.h`

Function Documentation

`template<typename T> __host__ __device__ complex<T> thrust::log(const complex < T > & z)`
Returns the complex natural logarithm of a complex number.

Parameters

- `z`: The complex argument.

Template Function `thrust::log10`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::log10(const complex < T > & z)
Returns the complex base 10 logarithm of a complex number.

Parameters

- z: The complex argument.

Template Function thrust::lower_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const LessThanComparable&)

- Defined in file_thrust_binary_search.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::lower_bound” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const LessThanComparable&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<class ForwardIterator, class InputIterator, class OutputIterator, class
↳ StrictWeakOrdering>
  OutputIterator thrust::lower_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)
- template<class ForwardIterator, class InputIterator, class OutputIterator>
  OutputIterator thrust::lower_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator)
- template<class ForwardIterator, class LessThanComparable>
  ForwardIterator thrust::lower_bound(ForwardIterator, ForwardIterator, const
↳ LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  ForwardIterator thrust::lower_bound(ForwardIterator, ForwardIterator, const T&,
↳ StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator, typename StrictWeakOrdering>__host__ __device__
↳ OutputIterator thrust::lower_bound(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, ForwardIterator, ForwardIterator, InputIterator, InputIterator,
↳ OutputIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::lower_
↳ bound(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳ LessThanComparable>__host__ __device__ ForwardIterator thrust::lower_bound(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
↳ StrictWeakOrdering>__host__ __device__ ForwardIterator thrust::lower_bound(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const T &, StrictWeakOrdering)
```

Template Function `thrust::lower_bound(ForwardIterator, ForwardIterator, const LessThanComparable&)`

Function Documentation

```
template<class ForwardIterator, class LessThanComparable>
```

```
ForwardIterator thrust::lower_bound(ForwardIterator first, ForwardIterator last, const  
LessThanComparable &value)
```

`lower_bound` is a version of binary search: it attempts to find the element value in an ordered range `[first, last)`. Specifically, it returns the first position where value could be inserted without violating the ordering. This version of `lower_bound` uses `operator<` for comparison and returns the furthestmost iterator `i` in `[first, last)` such that, for every iterator `j` in `[first, i)`, `*j < value`.

The following code snippet demonstrates how to use `lower_bound` to search for values in an ordered range.

Return The furthestmost iterator `i`, such that `*i < value`.

Parameters

- `first`: The beginning of the ordered sequence.
- `last`: The end of the ordered sequence.
- `value`: The value to be searched.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#).
- `LessThanComparable`: is a model of [LessThanComparable](#).

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::lower_bound(input.begin(), input.end(), 0); // returns input.begin()
thrust::lower_bound(input.begin(), input.end(), 1); // returns input.begin() + 1
thrust::lower_bound(input.begin(), input.end(), 2); // returns input.begin() + 1
thrust::lower_bound(input.begin(), input.end(), 3); // returns input.begin() + 2
thrust::lower_bound(input.begin(), input.end(), 8); // returns input.begin() + 4
thrust::lower_bound(input.begin(), input.end(), 9); // returns input.end()
```

See http://www.sgi.com/tech/stl/lower_bound.html

See `upper_bound`

See `equal_range`

See `binary_search`

Template Function `thrust::lower_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::lower_bound” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<class ForwardIterator, class InputIterator, class OutputIterator, class
↳ StrictWeakOrdering>
  OutputIterator thrust::lower_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)
- template<class ForwardIterator, class InputIterator, class OutputIterator>
  OutputIterator thrust::lower_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator)
- template<class ForwardIterator, class LessThanComparable>
  ForwardIterator thrust::lower_bound(ForwardIterator, ForwardIterator, const
↳ LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  ForwardIterator thrust::lower_bound(ForwardIterator, ForwardIterator, const T&,
↳ StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator, typename StrictWeakOrdering>__host__ __device__
↳ OutputIterator thrust::lower_bound(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, ForwardIterator, ForwardIterator, InputIterator, InputIterator,
↳ OutputIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::lower_
↳ bound(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳ LessThanComparable>__host__ __device__ ForwardIterator thrust::lower_bound(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
↳ StrictWeakOrdering>__host__ __device__ ForwardIterator thrust::lower_bound(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const T &, StrictWeakOrdering)
```

Template Function thrust::lower_bound(ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)

Function Documentation

```
template<class ForwardIterator, class T, class StrictWeakOrdering>
```

```
ForwardIterator thrust::lower_bound(ForwardIterator first, ForwardIterator last, const T &value,
↳ StrictWeakOrdering comp)
```

lower_bound is a version of binary search: it attempts to find the element value in an ordered range [first, last). Specifically, it returns the first position where value could be inserted without violating the ordering. This version of lower_bound uses function object comp for comparison and returns the furthestmost iterator i in [first, last) such that, for every iterator j in [first, i), comp(*j, value) is true.

The following code snippet demonstrates how to use lower_bound to search for values in an ordered range.

Return The furthestmost iterator i, such that comp(*i, value) is true.

Parameters

- first: The beginning of the ordered sequence.
- last: The end of the ordered sequence.
- value: The value to be searched.
- comp: The comparison operator.

Template Parameters

- ForwardIterator: is a model of [Forward Iterator](#).
- T: is comparable to ForwardIterator's value_type.
- StrictWeakOrdering: is a model of [Strict Weak Ordering](#).

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
#include <thrust/functional.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::lower_bound(input.begin(), input.end(), 0, thrust::less<int>()); //
↳ returns input.begin()
thrust::lower_bound(input.begin(), input.end(), 1, thrust::less<int>()); //
↳ returns input.begin() + 1
thrust::lower_bound(input.begin(), input.end(), 2, thrust::less<int>()); //
↳ returns input.begin() + 1
thrust::lower_bound(input.begin(), input.end(), 3, thrust::less<int>()); //
↳ returns input.begin() + 2
thrust::lower_bound(input.begin(), input.end(), 8, thrust::less<int>()); //
↳ returns input.begin() + 4
thrust::lower_bound(input.begin(), input.end(), 9, thrust::less<int>()); //
↳ returns input.end()
```

See http://www.sgi.com/tech/stl/lower_bound.html

See `upper_bound`

See `equal_range`

See `binary_search`

Template Function `thrust::lower_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::lower_bound” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<class ForwardIterator, class InputIterator, class OutputIterator, class
↳ StrictWeakOrdering>
  OutputIterator thrust::lower_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)
- template<class ForwardIterator, class InputIterator, class OutputIterator>
  OutputIterator thrust::lower_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator)
- template<class ForwardIterator, class LessThanComparable>
  ForwardIterator thrust::lower_bound(ForwardIterator, ForwardIterator, const
↳ LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  ForwardIterator thrust::lower_bound(ForwardIterator, ForwardIterator, const T&,
↳ StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator, typename StrictWeakOrdering>__host__ __device__
↳ OutputIterator thrust::lower_bound(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, ForwardIterator, ForwardIterator, InputIterator, InputIterator,
↳ OutputIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::lower_
↳ bound(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳ LessThanComparable>__host__ __device__ ForwardIterator thrust::lower_bound(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
↳ StrictWeakOrdering>__host__ __device__ ForwardIterator thrust::lower_bound(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const T &, StrictWeakOrdering)
```

Template Function thrust::lower_bound(ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)

Function Documentation

```
template<class ForwardIterator, class InputIterator, class OutputIterator>
```

```
OutputIterator thrust::lower_bound(ForwardIterator first, ForwardIterator last, InputIterator values_first, InputIterator values_last, OutputIterator result)
```

lower_bound is a vectorized version of binary search: for each iterator v in [values_first, values_last) it attempts to find the value *v in an ordered range [first, last). Specifically, it returns the index of first position where value could be inserted without violating the ordering.

The following code snippet demonstrates how to use lower_bound to search for multiple values in a ordered range.

Pre The ranges [first, last) and [result, result + (last - first)) shall not overlap.

Parameters

- first: The beginning of the ordered sequence.

- last: The end of the ordered sequence.
- values_first: The beginning of the search values sequence.
- values_last: The end of the search values sequence.
- result: The beginning of the output sequence.

Template Parameters

- ForwardIterator: is a model of [Forward Iterator](#).
- InputIterator: is a model of [Input Iterator](#). and InputIterator's value_type is [LessThanComparable](#).
- OutputIterator: is a model of [Output Iterator](#). and ForwardIterator's difference_type is convertible to OutputIterator's value_type.

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::device_vector<int> values(6);
values[0] = 0;
values[1] = 1;
values[2] = 2;
values[3] = 3;
values[4] = 8;
values[5] = 9;

thrust::device_vector<unsigned int> output(6);

thrust::lower_bound(input.begin(), input.end(),
                    values.begin(), values.end(),
                    output.begin());

// output is now [0, 1, 1, 2, 4, 5]
```

See http://www.sgi.com/tech/stl/lower_bound.html

See `upper_bound`

See `equal_range`

See `binary_search`

Template Function `thrust::lower_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::lower_bound” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<class ForwardIterator, class InputIterator, class OutputIterator, class
↳ StrictWeakOrdering>
  OutputIterator thrust::lower_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)
- template<class ForwardIterator, class InputIterator, class OutputIterator>
  OutputIterator thrust::lower_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator)
- template<class ForwardIterator, class LessThanComparable>
  ForwardIterator thrust::lower_bound(ForwardIterator, ForwardIterator, const
↳ LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  ForwardIterator thrust::lower_bound(ForwardIterator, ForwardIterator, const T&,
↳ StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator, typename StrictWeakOrdering>__host__ __device__
↳ OutputIterator thrust::lower_bound(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, ForwardIterator, ForwardIterator, InputIterator, InputIterator,
↳ OutputIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::lower_
↳ bound(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳ LessThanComparable>__host__ __device__ ForwardIterator thrust::lower_bound(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
↳ StrictWeakOrdering>__host__ __device__ ForwardIterator thrust::lower_bound(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const T &, StrictWeakOrdering)
```

Template Function thrust::lower_bound(ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)

Function Documentation

```
template<class ForwardIterator, class InputIterator, class OutputIterator, class StrictWeakOrdering>
OutputIterator thrust::lower_bound(ForwardIterator first, ForwardIterator last, InputIterator val-
ues_first, InputIterator values_last, OutputIterator result,
StrictWeakOrdering comp)
```

lower_bound is a vectorized version of binary search: for each iterator v in [values_first, values_last) it attempts to find the value *v in an ordered range [first, last). Specifically, it returns the index of first position where value could be inserted without violating the ordering. This version of lower_bound uses function object comp for comparison.

The following code snippet demonstrates how to use lower_bound to search for multiple values in a ordered range.

Pre The ranges `[first, last)` and `[result, result + (last - first))` shall not overlap.

Parameters

- `first`: The beginning of the ordered sequence.
- `last`: The end of the ordered sequence.
- `values_first`: The beginning of the search values sequence.
- `values_last`: The end of the search values sequence.
- `result`: The beginning of the output sequence.
- `comp`: The comparison operator.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#).
- `InputIterator`: is a model of [Input Iterator](#). and `InputIterator`'s `value_type` is comparable to `ForwardIterator`'s `value_type`.
- `OutputIterator`: is a model of [Output Iterator](#). and `ForwardIterator`'s `difference_type` is convertible to `OutputIterator`'s `value_type`.
- `StrictWeakOrdering`: is a model of [Strict Weak Ordering](#).

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
#include <thrust/functional.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::device_vector<int> values(6);
values[0] = 0;
values[1] = 1;
values[2] = 2;
values[3] = 3;
values[4] = 8;
values[5] = 9;

thrust::device_vector<unsigned int> output(6);

thrust::lower_bound(input.begin(), input.end(),
                    values.begin(), values.end(),
                    output.begin(),
                    thrust::less<int>());

// output is now [0, 1, 1, 2, 4, 5]
```

See http://www.sgi.com/tech/stl/lower_bound.html

See `upper_bound`

See `equal_range`

See `binary_search`

Template Function `thrust::make_pair`

- Defined in `file_thrust_pair.h`

Function Documentation

template<typename T1, typename T2> __host__ __device__ pair<T1,T2> thrust::make_pair(T1 x, T2 y)

This convenience function creates a `pair` from two objects.

Return A newly-constructed `pair` copied from `a` and `b`.

Parameters

- `x`: The first object to copy from.
- `y`: The second object to copy from.

Template Parameters

- `T1`: There are no requirements on the type of `T1`.
- `T2`: There are no requirements on the type of `T2`.

Template Function `thrust::make_tuple(const T0&)`

- Defined in `file_thrust_tuple.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::make_tuple`” with arguments `(const T0&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<class T0, class T1>__host__ __device__ detail::make_tuple_mapper<T0, T1>
  ↳::type thrust::make_tuple(const T0 &, const T1 &)
- template<class T0>__host__ __device__ detail::make_tuple_mapper<T0>::type_
  ↳thrust::make_tuple(const T0 &)
```

Template Function `thrust::make_tuple(const T0&, const T1&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::make_tuple`” with arguments `(const T0&, const T1&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<class T0, class T1>__host__ __device__ detail::make_tuple_mapper<T0, T1>
  ↳::type thrust::make_tuple(const T0 &, const T1 &)
- template<class T0>__host__ __device__ detail::make_tuple_mapper<T0>::type_
  ↳thrust::make_tuple(const T0 &)
```

Template Function `thrust::malloc(const thrust::detail::execution_policy_base<DerivedPolicy>&, std::size_t)`

- Defined in `file_thrust_memory.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::malloc`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, std::size_t)` in doxygen xml output for project “roc-Thrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy>__host__ __device__ pointer<void,DerivedPolicy>_  
  ↳thrust::malloc(const thrust::detail::execution_policy_base< DerivedPolicy > &,_  
  ↳std::size_t)  
- template<typename T, typename DerivedPolicy>__host__ __device__ pointer<T,  
  ↳DerivedPolicy> thrust::malloc(const thrust::detail::execution_policy_base<_  
  ↳DerivedPolicy > &, std::size_t)
```

Template Function `thrust::malloc(const thrust::detail::execution_policy_base<DerivedPolicy>&, std::size_t)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::malloc`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, std::size_t)` in doxygen xml output for project “roc-Thrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy>__host__ __device__ pointer<void,DerivedPolicy>_  
  ↳thrust::malloc(const thrust::detail::execution_policy_base< DerivedPolicy > &,_  
  ↳std::size_t)  
- template<typename T, typename DerivedPolicy>__host__ __device__ pointer<T,  
  ↳DerivedPolicy> thrust::malloc(const thrust::detail::execution_policy_base<_  
  ↳DerivedPolicy > &, std::size_t)
```

Template Function `thrust::max_element(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`

- Defined in `file_thrust_extrema.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::max_element” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳ BinaryPredicate>__host__ __device__ ForwardIterator thrust::max_element(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
↳ ForwardIterator thrust::max_element(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, ForwardIterator, ForwardIterator)
- template<typename ForwardIterator, typename BinaryPredicate>
ForwardIterator thrust::max_element(ForwardIterator, ForwardIterator,
↳ BinaryPredicate)
- template<typename ForwardIterator>
ForwardIterator thrust::max_element(ForwardIterator, ForwardIterator)
```

Template Function thrust::max_element(ForwardIterator, ForwardIterator)

Function Documentation

template<typename **ForwardIterator**>

ForwardIterator thrust::max_element (*ForwardIterator* first, *ForwardIterator* last)

max_element finds the largest element in the range [first, last). It returns the first iterator *i* in [first, last) such that no other iterator in [first, last) points to a value larger than **i*. The return value is last if and only if [first, last) is an empty range.

The two versions of max_element differ in how they define whether one element is greater than another. This version compares objects using operator<. Specifically, this version of max_element returns the first iterator *i* in [first, last) such that, for every iterator *j* in [first, last), **i* < **j* is false.

```
#include <thrust/extrema.h>
...
int data[6] = {1, 0, 2, 2, 1, 3};
int *result = thrust::max_element(data, data + 6);

// *result == 3
```

Return An iterator pointing to the largest element of the range [first, last), if it is not an empty range; last, otherwise.

Parameters

- first: The beginning of the sequence.
- last: The end of the sequence.

Template Parameters

- ForwardIterator: is a model of [Forward Iterator](#), and ForwardIterator's value_type is a model of [LessThan Comparable](#).

See http://www.sgi.com/tech/stl/max_element.html

Template Function `thrust::max_element(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::max_element” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, BinaryPredicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename
  ↳ BinaryPredicate>__host__ __device__ ForwardIterator thrust::max_element(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
  ↳ ForwardIterator thrust::max_element(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, ForwardIterator, ForwardIterator)
- template<typename ForwardIterator, typename BinaryPredicate>
  ↳ ForwardIterator thrust::max_element(ForwardIterator, ForwardIterator,
  ↳ BinaryPredicate)
- template<typename ForwardIterator>
  ↳ ForwardIterator thrust::max_element(ForwardIterator, ForwardIterator)
```

Template Function `thrust::max_element(ForwardIterator, ForwardIterator, BinaryPredicate)`

Function Documentation

template<typename **ForwardIterator**, typename **BinaryPredicate**>

ForwardIterator thrust::max_element(*ForwardIterator first*, *ForwardIterator last*, *BinaryPredicate comp*)

`max_element` finds the largest element in the range `[first, last)`. It returns the first iterator `i` in `[first, last)` such that no other iterator in `[first, last)` points to a value larger than `*i`. The return value is `last` if and only if `[first, last)` is an empty range.

The two versions of `max_element` differ in how they define whether one element is less than another. This version compares objects using a function object `comp`. Specifically, this version of `max_element` returns the first iterator `i` in `[first, last)` such that, for every iterator `j` in `[first, last)`, `comp(*i, *j)` is false.

The following code snippet demonstrates how to use `max_element` to find the largest element of a collection of key-value pairs.

Return An iterator pointing to the largest element of the range `[first, last)`, if it is not an empty range; `last`, otherwise.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.
- `comp`: A binary predicate used for comparison.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator`'s `value_type` is convertible to both `comp`'s `first_argument_type` and `second_argument_type`.

- BinaryPredicate: is a model of Binary Predicate.

```
#include <thrust/extrema.h>

struct key_value
{
    int key;
    int value;
};

struct compare_key_value
{
    __host__ __device__
    bool operator()(key_value lhs, key_value rhs)
    {
        return lhs.key < rhs.key;
    }
};

...
key_value data[4] = { {4,5}, {0,7}, {2,3}, {6,1} };

key_value *largest = thrust::max_element(data, data + 4, compare_key_value());

// largest == data + 3
// *largest == {6,1}
```

See http://www.sgi.com/tech/stl/max_element.html

Template Function `thrust::merge(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`

- Defined in file `thrust_merge.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::merge” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename StrictWeakCompare>__host__ __device__
  ↳ OutputIterator thrust::merge(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2, InputIterator2,
  ↳ OutputIterator, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::merge(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename StrictWeakCompare>
  ↳ OutputIterator thrust::merge(InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator>
  ↳ OutputIterator thrust::merge(InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator2, OutputIterator)
```

Template Function `thrust::merge(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename OutputIterator>
OutputIterator thrust::merge (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2, OutputIterator result)
```

`merge` combines two sorted ranges `[first1, last1)` and `[first2, last2)` into a single sorted range. That is, it copies from `[first1, last1)` and `[first2, last2)` into `[result, result + (last1 - first1) + (last2 - first2))` such that the resulting range is in ascending order. `merge` is stable, meaning both that the relative order of elements within each input range is preserved, and that for equivalent elements in both input ranges the element from the first range precedes the element from the second. The return value is `result + (last1 - first1) + (last2 - first2)`.

This version of `merge` compares elements using `operator<`.

The following code snippet demonstrates how to use `merge` to compute the merger of two sorted sets of integers.

Return The end of the output range.

Pre The ranges `[first1, last1)` and `[first2, last2)` shall be sorted with respect to `operator<`.

Pre The resulting range shall not overlap with either input range.

Parameters

- `first1`: The beginning of the first input range.
- `last1`: The end of the first input range.
- `first2`: The beginning of the second input range.
- `last2`: The end of the second input range.
- `result`: The beginning of the merged output.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `OutputIterator`: is a model of [Output Iterator](#).

```
#include <thrust/merge.h>
...
int A1[6] = {1, 3, 5, 7, 9, 11};
int A2[7] = {1, 1, 2, 3, 5, 8, 13};
```

(continues on next page)

(continued from previous page)

```
int result[13];

int *result_end = thrust::merge(A1, A1 + 6, A2, A2 + 7, result);
// result = {1, 1, 1, 2, 3, 3, 5, 5, 7, 8, 9, 11, 13}
```

See <http://www.sgi.com/tech/stl/merge.html>

See `set_union`

See `sort`

See `is_sorted`

Template Function `thrust::merge(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::merge” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename StrictWeakCompare>__host__ __device__
  ↳OutputIterator thrust::merge(const thrust::detail::execution_policy_base<
  ↳DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2, InputIterator2,
  ↳ OutputIterator, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::merge(const
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳InputIterator1, InputIterator2, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳OutputIterator, typename StrictWeakCompare>
  ↳OutputIterator thrust::merge(InputIterator1, InputIterator1, InputIterator2,
  ↳InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳OutputIterator>
  ↳OutputIterator thrust::merge(InputIterator1, InputIterator1, InputIterator2,
  ↳InputIterator2, OutputIterator)
```

Template Function `thrust::merge(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**, typename **StrictWeakCompare**>
OutputIterator thrust::merge(*InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2, OutputIterator result, StrictWeakCompare comp*)

merge combines two sorted ranges [first1, last1) and [first2, last2) into a single sorted range. That is, it copies from [first1, last1) and [first2, last2) into [result, result + (last1 - first1) + (last2 - first2)) such that the resulting range is in ascending order.

`merge` is stable, meaning both that the relative order of elements within each input range is preserved, and that for equivalent elements in both input ranges the element from the first range precedes the element from the second. The return value is `result + (last1 - first1) + (last2 - first2)`.

This version of `merge` compares elements using a function object `comp`.

The following code snippet demonstrates how to use `merge` to compute the merger of two sets of integers sorted in descending order.

Return The end of the output range.

Pre The ranges `[first1, last1)` and `[first2, last2)` shall be sorted with respect to `comp`.

Pre The resulting range shall not overlap with either input range.

Parameters

- `first1`: The beginning of the first input range.
- `last1`: The end of the first input range.
- `first2`: The beginning of the second input range.
- `last2`: The end of the second input range.
- `result`: The beginning of the merged output.
- `comp`: Comparison operator.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1`'s `value_type` is convertible to `StrictWeakCompare`'s `first_argument_type`. and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2`'s `value_type` is convertible to `StrictWeakCompare`'s `second_argument_type`. and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `OutputIterator`: is a model of [Output Iterator](#).
- `StrictWeakCompare`: is a model of [Strict Weak Ordering](#).

```
#include <thrust/merge.h>
#include <thrust/functional.h>
...
int A1[6] = {11, 9, 7, 5, 3, 1};
int A2[7] = {13, 8, 5, 3, 2, 1, 1};

int result[13];

int *result_end = thrust::merge(A1, A1 + 6, A2, A2 + 7, result, thrust::greater
↪<int>());
// result = {13, 11, 9, 8, 7, 5, 5, 3, 3, 2, 1, 1, 1}
```

See <http://www.sgi.com/tech/stl/merge.html>

See `sort`

See `is_sorted`

Template Function `thrust::merge_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)`

- Defined in `file_thrust_merge.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::merge_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
    ↪ typename OutputIterator2, typename Compare>__host__ __device__ thrust::pair
    ↪<OutputIterator1,OutputIterator2> thrust::merge_by_key(const
    ↪thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
    ↪InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4,
    ↪OutputIterator1, OutputIterator2, Compare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
    ↪ typename OutputIterator2>__host__ __device__ thrust::pair<OutputIterator1,
    ↪OutputIterator2> thrust::merge_by_key(const thrust::detail::execution_policy_base
    ↪< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
    ↪InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪InputIterator3, typename InputIterator4, typename OutputIterator1, typename
    ↪OutputIterator2, typename StrictWeakCompare>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::merge_by_
    ↪key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪InputIterator3, InputIterator4, OutputIterator1, OutputIterator2,
    ↪StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪InputIterator3, typename InputIterator4, typename OutputIterator1, typename
    ↪OutputIterator2>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::merge_by_
    ↪key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)
```

Template Function `thrust::merge_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **InputIterator3**, typename **InputIterator4**, typename **OutputIterator1**, typename **OutputIterator2**>

```
thrust::pair<OutputIterator1, OutputIterator2> thrust::merge_by_key (InputIterator1 keys_first1, InputIterator1 keys_last1, InputIterator2 keys_first2, InputIterator2 keys_last2, InputIterator3 values_first1, InputIterator4 values_first2, OutputIterator1 keys_result, OutputIterator2 values_result)
```

`merge_by_key` performs a key-value merge. That is, `merge_by_key` copies elements from $[keys_first1, keys_last1)$ and $[keys_first2, keys_last2)$ into a single range, $[keys_result, keys_result + (keys_last1 - keys_first1) + (keys_last2 - keys_first2))$ such that the resulting range is in ascending key order.

At the same time, `merge_by_key` copies elements from the two associated ranges $[values_first1 + (keys_last1 - keys_first1))$ and $[values_first2 + (keys_last2 - keys_first2))$ into a single range, $[values_result, values_result + (keys_last1 - keys_first1) + (keys_last2 - keys_first2))$ such that the resulting range is in ascending order implied by each input element's associated key.

`merge_by_key` is stable, meaning both that the relative order of elements within each input range is preserved, and that for equivalent elements in all input key ranges the element from the first range precedes the element from the second.

The return value is $(keys_result + (keys_last1 - keys_first1) + (keys_last2 - keys_first2))$ and $(values_result + (keys_last1 - keys_first1) + (keys_last2 - keys_first2))$.

The following code snippet demonstrates how to use `merge_by_key` to compute the merger of two sets of integers sorted in ascending order.

Return A pair `p` such that `p.first` is the end of the output range of keys, and such that `p.second` is the end of the output range of values.

Pre The ranges $[keys_first1, keys_last1)$ and $[keys_first2, keys_last2)$ shall be sorted with respect to `operator<`.

Pre The resulting ranges shall not overlap with any input range.

Parameters

- `keys_first1`: The beginning of the first input range of keys.
- `keys_last1`: The end of the first input range of keys.
- `keys_first2`: The beginning of the second input range of keys.
- `keys_last2`: The end of the second input range of keys.
- `values_first1`: The beginning of the first input range of values.
- `values_first2`: The beginning of the first input range of values.
- `keys_result`: The beginning of the merged output range of keys.
- `values_result`: The beginning of the merged output range of values.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.

- InputIterator2: is a model of [Input Iterator](#), InputIterator2 and InputIterator1 have the same value_type, InputIterator2's value_type is a model of [LessThan Comparable](#), the ordering on InputIterator2's value_type is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and InputIterator2's value_type is convertible to a type in OutputIterator's set of value_types.
- InputIterator3: is a model of [Input Iterator](#), and InputIterator3's value_type is convertible to a type in OutputIterator2's set of value_types.
- InputIterator4: is a model of [Input Iterator](#), and InputIterator4's value_type is convertible to a type in OutputIterator2's set of value_types.
- OutputIterator1: is a model of [Output Iterator](#).
- OutputIterator2: is a model of [Output Iterator](#).

```
#include <thrust/merge.h>
#include <thrust/functional.h>
...
int A_keys[6] = {1, 3, 5, 7, 9, 11};
int A_vals[6] = {0, 0, 0, 0, 0, 0};

int B_keys[7] = {1, 1, 2, 3, 5, 8, 13};
int B_vals[7] = {1, 1, 1, 1, 1, 1, 1};

int keys_result[13];
int vals_result[13];

thrust::pair<int*,int*> end = thrust::merge_by_key(A_keys, A_keys + 6, B_keys, B_
↪keys + 7, A_vals, B_vals, keys_result, vals_result);

// keys_result = {1, 1, 1, 2, 3, 3, 5, 5, 7, 8, 9, 11, 13}
// vals_result = {0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1}
```

See [merge](#)

See [sort_by_key](#)

See [is_sorted](#)

Template Function `thrust::merge_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, Compare)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::merge_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, Compare) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
  ↳ typename OutputIterator2, typename Compare>__host__ __device__ thrust::pair
  ↳<OutputIterator1,OutputIterator2> thrust::merge_by_key(const
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4,
  ↳OutputIterator1, OutputIterator2, Compare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
  ↳ typename OutputIterator2>__host__ __device__ thrust::pair<OutputIterator1,
  ↳OutputIterator2> thrust::merge_by_key(const thrust::detail::execution_policy_base
  ↳< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳InputIterator3, typename InputIterator4, typename OutputIterator1, typename
  ↳OutputIterator2, typename StrictWeakCompare>
  thrust::pair<OutputIterator1, OutputIterator2> thrust::merge_by_
  ↳key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
  ↳InputIterator3, InputIterator4, OutputIterator1, OutputIterator2,
  ↳StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳InputIterator3, typename InputIterator4, typename OutputIterator1, typename
  ↳OutputIterator2>
  thrust::pair<OutputIterator1, OutputIterator2> thrust::merge_by_
  ↳key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
  ↳InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)

```

Template Function `thrust::merge_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)`

Function Documentation

`template<typename InputIterator1, typename InputIterator2, typename InputIterator3, typename InputIterator4, typename OutputIterator1, OutputIterator2> thrust::pair<OutputIterator1, OutputIterator2> thrust::merge_by_key` (*InputIterator1 keys_first1, InputIterator1 keys_last1, InputIterator2 keys_first2, InputIterator2 keys_last2, InputIterator3 values_first1, InputIterator4 values_first2, OutputIterator1 keys_result, OutputIterator2 values_result, StrictWeakCompare comp*)

`merge_by_key` performs a key-value merge. That is, `merge_by_key` copies elements from `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)` into a single range, `[keys_result, keys_result + (keys_last1 - keys_first1) + (keys_last2 - keys_first2))` such that the resulting range is in ascending key order.

At the same time, `merge_by_key` copies elements from the two associated ranges `[values_first1 + (keys_last1 - keys_first1))` and `[values_first2 + (keys_last2 - keys_first2))` into a single range, `[values_result, values_result + (keys_last1 - keys_first1) + (keys_last2 - keys_first2))` such that the resulting range is in ascending order implied by each input element's associated key.

`merge_by_key` is stable, meaning both that the relative order of elements within each input range is preserved,

and that for equivalent elements in all input key ranges the element from the first range precedes the element from the second.

The return value is $(\text{keys_result} + (\text{keys_last1} - \text{keys_first1}) + (\text{keys_last2} - \text{keys_first2}))$ and $(\text{values_result} + (\text{keys_last1} - \text{keys_first1}) + (\text{keys_last2} - \text{keys_first2}))$.

This version of `merge_by_key` compares key elements using a function object `comp`.

The following code snippet demonstrates how to use `merge_by_key` to compute the merger of two sets of integers sorted in descending order.

Return A pair `p` such that `p.first` is the end of the output range of keys, and such that `p.second` is the end of the output range of values.

Pre The ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)` shall be sorted with respect to `comp`.

Pre The resulting ranges shall not overlap with any input range.

Parameters

- `keys_first1`: The beginning of the first input range of keys.
- `keys_last1`: The end of the first input range of keys.
- `keys_first2`: The beginning of the second input range of keys.
- `keys_last2`: The end of the second input range of keys.
- `values_first1`: The beginning of the first input range of values.
- `values_first2`: The beginning of the first input range of values.
- `keys_result`: The beginning of the merged output range of keys.
- `values_result`: The beginning of the merged output range of values.
- `comp`: Comparison operator.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1`'s `value_type` is convertible to `StrictWeakCompare`'s `first_argument_type`. and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator1`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2`'s `value_type` is convertible to `StrictWeakCompare`'s `second_argument_type`. and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator1`'s set of `value_types`.
- `InputIterator3`: is a model of [Input Iterator](#), and `InputIterator3`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `InputIterator4`: is a model of [Input Iterator](#), and `InputIterator4`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `OutputIterator1`: is a model of [Output Iterator](#).
- `OutputIterator2`: is a model of [Output Iterator](#).
- `StrictWeakCompare`: is a model of [Strict Weak Ordering](#).

```
#include <thrust/merge.h>
#include <thrust/functional.h>
...
int A_keys[6] = {11, 9, 7, 5, 3, 1};
```

(continues on next page)

(continued from previous page)

```

int A_vals[6] = { 0, 0, 0, 0, 0, 0};

int B_keys[7] = {13, 8, 5, 3, 2, 1, 1};
int B_vals[7] = { 1, 1, 1, 1, 1, 1, 1};

int keys_result[13];
int vals_result[13];

thrust::pair<int*,int*> end = thrust::merge_by_key(A_keys, A_keys + 6, B_keys, B_
↳keys + 7, A_vals, B_vals, keys_result, vals_result, thrust::greater<int>());

// keys_result = {13, 11, 9, 8, 7, 5, 5, 3, 3, 2, 1, 1, 1}
// vals_result = { 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1}

```

See `merge`

See `sort_by_key`

See `is_sorted`

Template Function `thrust::min_element(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`

- Defined in file `thrust_extrema.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::min_element`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename ForwardIterator, typename
↳BinaryPredicate>__host__ __device__ ForwardIterator thrust::min_element(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ForwardIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
↳ForwardIterator thrust::min_element(const thrust::detail::execution_policy_base<
↳DerivedPolicy > &, ForwardIterator, ForwardIterator)
- template<typename ForwardIterator, typename BinaryPredicate>
  ForwardIterator thrust::min_element(ForwardIterator, ForwardIterator,
↳BinaryPredicate)
- template<typename ForwardIterator>
  ForwardIterator thrust::min_element(ForwardIterator, ForwardIterator)

```

Template Function `thrust::min_element(ForwardIterator, ForwardIterator)`

Function Documentation

template<typename **ForwardIterator**>

ForwardIterator thrust::min_element (*ForwardIterator* first, *ForwardIterator* last)

min_element finds the smallest element in the range [first, last). It returns the first iterator i in [first, last) such that no other iterator in [first, last) points to a value smaller than *i. The return value is last if and only if [first, last) is an empty range.

The two versions of min_element differ in how they define whether one element is less than another. This version compares objects using operator<. Specifically, this version of min_element returns the first iterator i in [first, last) such that, for every iterator j in [first, last), *j < *i is false.

```
#include <thrust/extrema.h>
...
int data[6] = {1, 0, 2, 2, 1, 3};
int *result = thrust::min_element(data, data + 6);

// result is data + 1
// *result is 0
```

Return An iterator pointing to the smallest element of the range [first, last), if it is not an empty range; last, otherwise.

Parameters

- first: The beginning of the sequence.
- last: The end of the sequence.

Template Parameters

- ForwardIterator: is a model of [Forward Iterator](#), and ForwardIterator's value_type is a model of [LessThan Comparable](#).

See http://www.sgi.com/tech/stl/min_element.html

Template Function thrust::min_element(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, BinaryPredicate)

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::min_element” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, BinaryPredicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳ BinaryPredicate>__host__ __device__ ForwardIterator thrust::min_element(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
↳ ForwardIterator thrust::min_element(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, ForwardIterator, ForwardIterator)
- template<typename ForwardIterator, typename BinaryPredicate>
  ForwardIterator thrust::min_element(ForwardIterator, ForwardIterator,
↳ BinaryPredicate)
- template<typename ForwardIterator>
  ForwardIterator thrust::min_element(ForwardIterator, ForwardIterator)
```

Template Function `thrust::min_element(ForwardIterator, ForwardIterator, BinaryPredicate)`

Function Documentation

template<typename **ForwardIterator**, typename **BinaryPredicate**>

ForwardIterator thrust::**min_element**(*ForwardIterator first*, *ForwardIterator last*, *BinaryPredicate comp*)

`min_element` finds the smallest element in the range `[first, last)`. It returns the first iterator `i` in `[first, last)` such that no other iterator in `[first, last)` points to a value smaller than `*i`. The return value is `last` if and only if `[first, last)` is an empty range.

The two versions of `min_element` differ in how they define whether one element is less than another. This version compares objects using a function object `comp`. Specifically, this version of `min_element` returns the first iterator `i` in `[first, last)` such that, for every iterator `j` in `[first, last)`, `comp(*j, *i)` is `false`.

The following code snippet demonstrates how to use `min_element` to find the smallest element of a collection of key-value pairs.

Return An iterator pointing to the smallest element of the range `[first, last)`, if it is not an empty range; `last`, otherwise.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.
- `comp`: A binary predicate used for comparison.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator`'s `value_type` is convertible to both `comp`'s `first_argument_type` and `second_argument_type`.
- `BinaryPredicate`: is a model of [Binary Predicate](#).

```
#include <thrust/extrema.h>

struct key_value
{
    int key;
    int value;
};

struct compare_key_value
{
    __host__ __device__
    bool operator()(key_value lhs, key_value rhs)
    {
        return lhs.key < rhs.key;
    }
};

...
key_value data[4] = { {4,5}, {0,7}, {2,3}, {6,1} };

key_value *smallest = thrust::min_element(data, data + 4, compare_key_value());
```

(continues on next page)

(continued from previous page)

```
// smallest == data + 1
// *smallest == {0,7}
```

See http://www.sgi.com/tech/stl/min_element.html

Template Function `thrust::minmax_element(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`

- Defined in file `thrust_extrema.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::minmax_element`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename
  ↳ BinaryPredicate>__host__ __device__ thrust::pair<ForwardIterator,ForwardIterator>
  ↳ thrust::minmax_element(const thrust::detail::execution_policy_base< DerivedPolicy
  ↳ > &, ForwardIterator, ForwardIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
  ↳ thrust::pair<ForwardIterator,ForwardIterator> thrust::minmax_element(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator)
- template<typename ForwardIterator, typename BinaryPredicate>
  thrust::pair<ForwardIterator, ForwardIterator> thrust::minmax_
  ↳ element(ForwardIterator, ForwardIterator, BinaryPredicate)
- template<typename ForwardIterator>
  thrust::pair<ForwardIterator, ForwardIterator> thrust::minmax_
  ↳ element(ForwardIterator, ForwardIterator)
```

Template Function `thrust::minmax_element(ForwardIterator, ForwardIterator)`

Function Documentation

template<typename **ForwardIterator**>

`thrust::pair<ForwardIterator, ForwardIterator> thrust::minmax_element (ForwardIterator first, ForwardIterator last)`

`minmax_element` finds the smallest and largest elements in the range `[first, last)`. It returns a pair of iterators (`imin`, `imax`) where `imin` is the same iterator returned by `min_element` and `imax` is the same iterator returned by `max_element`. This function is potentially more efficient than separate calls to `min_element` and `max_element`.

```
#include <thrust/extrema.h>
...
int data[6] = {1, 0, 2, 2, 1, 3};
thrust::pair<int *, int *> result = thrust::minmax_element(data, data + 6);

// result.first is data + 1
```

(continues on next page)

(continued from previous page)

```
// result.second is data + 5
// *result.first is 0
// *result.second is 3
```

Return A pair of iterator pointing to the smallest and largest elements of the range `[first, last)`, if it is not an empty range; `last`, otherwise.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator`'s `value_type` is a model of [LessThan Comparable](#).

See `min_element`

See `max_element`

See <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1840.pdf>

Template Function `thrust::minmax_element(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::minmax_element`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, BinaryPredicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename
  ↳ BinaryPredicate>__host__ __device__ thrust::pair<ForwardIterator,ForwardIterator>
  ↳ thrust::minmax_element(const thrust::detail::execution_policy_base< DerivedPolicy
  ↳ &, ForwardIterator, ForwardIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
  ↳ thrust::pair<ForwardIterator,ForwardIterator> thrust::minmax_element(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator)
- template<typename ForwardIterator, typename BinaryPredicate>
  ↳ thrust::pair<ForwardIterator, ForwardIterator> thrust::minmax_
  ↳ element(ForwardIterator, ForwardIterator, BinaryPredicate)
- template<typename ForwardIterator>
  ↳ thrust::pair<ForwardIterator, ForwardIterator> thrust::minmax_
  ↳ element(ForwardIterator, ForwardIterator)
```

Template Function `thrust::minmax_element(ForwardIterator, ForwardIterator, BinaryPredicate)`

Function Documentation

template<typename **ForwardIterator**, typename **BinaryPredicate**>

```
thrust::pair<ForwardIterator, ForwardIterator> thrust::minmax_element (ForwardIterator    first,
                                                                    ForwardIterator    last,
                                                                    BinaryPredicate comp)
```

`minmax_element` finds the smallest and largest elements in the range `[first, last)`. It returns a pair of iterators (`imin`, `imax`) where `imin` is the same iterator returned by `min_element` and `imax` is the same iterator returned by `max_element`. This function is potentially more efficient than separate calls to `min_element` and `max_element`.

The following code snippet demonstrates how to use `minmax_element` to find the smallest and largest elements of a collection of key-value pairs.

Return A pair of iterator pointing to the smallest and largest elements of the range `[first, last)`, if it is not an empty range; `last`, otherwise.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.
- `comp`: A binary predicate used for comparison.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator`'s `value_type` is convertible to both `comp`'s `first_argument_type` and `second_argument_type`.
- `BinaryPredicate`: is a model of [Binary Predicate](#).

```
#include <thrust/extrema.h>
#include <thrust/pair.h>

struct key_value
{
    int key;
    int value;
};

struct compare_key_value
{
    __host__ __device__
    bool operator() (key_value lhs, key_value rhs)
    {
        return lhs.key < rhs.key;
    }
};

...
key_value data[4] = { {4,5}, {0,7}, {2,3}, {6,1} };

thrust::pair<key_value*,key_value*> extrema = thrust::minmax_element(data, data + 4,
↪compare_key_value());

// extrema.first == data + 1
// *extrema.first == {0,7}
// extrema.second == data + 3
// *extrema.second == {6,1}
```

See `min_element`

See `max_element`

See <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1840.pdf>

Template Function `thrust::mismatch(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2)`

- Defined in file `_thrust_mismatch.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::mismatch`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename BinaryPredicate>__host__ __device__ thrust::pair<InputIterator1,
  ↳InputIterator2> thrust::mismatch(const thrust::detail::execution_policy_base<
  ↳DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2>
  ↳__host__ __device__ thrust::pair<InputIterator1, InputIterator2>
  ↳thrust::mismatch(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳InputIterator1, InputIterator1, InputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳BinaryPredicate>
  ↳thrust::pair<InputIterator1, InputIterator2> thrust::mismatch(InputIterator1,
  ↳InputIterator1, InputIterator2, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2>
  ↳thrust::pair<InputIterator1, InputIterator2> thrust::mismatch(InputIterator1,
  ↳InputIterator1, InputIterator2)
```

Template Function `thrust::mismatch(InputIterator1, InputIterator1, InputIterator2)`

Function Documentation

`template<typename InputIterator1, typename InputIterator2>`

`thrust::pair<InputIterator1, InputIterator2> thrust::mismatch (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2)`

`mismatch` finds the first position where the two ranges `[first1, last1)` and `[first2, first2 + (last1 - first1))` differ. The two versions of `mismatch` use different tests for whether elements differ.

This version of `mismatch` finds the first iterator `i` in `[first1, last1)` such that `*i == *(first2 + (i - first1))` is false. The return value is a pair whose first element is `i` and whose second element is `*(first2 + (i - first1))`. If no such iterator `i` exists, the return value is a pair whose first element is `last1` and whose second element is `*(first2 + (last1 - first1))`.

```
#include <thrust/mismatch.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> vec1(4);
thrust::device_vector<int> vec2(4);
```

(continues on next page)

(continued from previous page)

```

vec1[0] = 0;  vec2[0] = 0;
vec1[1] = 5;  vec2[1] = 5;
vec1[2] = 3;  vec2[2] = 8;
vec1[3] = 7;  vec2[3] = 7;

typedef thrust::device_vector<int>::iterator Iterator;
thrust::pair<Iterator,Iterator> result;

result = thrust::mismatch(vec1.begin(), vec1.end(), vec2.begin());

// result.first  is vec1.begin() + 2
// result.second is vec2.begin() + 2

```

Return The first position where the sequences differ.

Parameters

- first1: The beginning of the first sequence.
- last1: The end of the first sequence.
- first2: The beginning of the second sequence.

Template Parameters

- InputIterator1: is a model of [Input Iterator](#) and InputIterator1's value_type is equality comparable to InputIterator2's value_type.
- InputIterator2: is a model of [Input Iterator](#).

See [find](#)

See [find_if](#)

Template Function `thrust::mismatch(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::mismatch” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, BinaryPredicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename BinaryPredicate>__host__ __device__ thrust::pair<InputIterator1,
  ↳InputIterator2> thrust::mismatch(const thrust::detail::execution_policy_base<
  ↳DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2>
  ↳__host__ __device__ thrust::pair<InputIterator1, InputIterator2>
  ↳thrust::mismatch(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳InputIterator1, InputIterator1, InputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳BinaryPredicate>
  ↳thrust::pair<InputIterator1, InputIterator2> thrust::mismatch(InputIterator1,
  ↳InputIterator1, InputIterator2, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2>
  ↳thrust::pair<InputIterator1, InputIterator2> thrust::mismatch(InputIterator1,
  ↳InputIterator1, InputIterator2)

```

Template Function `thrust::mismatch(InputIterator1, InputIterator1, InputIterator2, BinaryPredicate)`

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename BinaryPredicate>
thrust::pair<InputIterator1, InputIterator2> thrust::mismatch(InputIterator1 first1, InputIterator1 last1,
                                                         InputIterator2 first2, BinaryPredicate
                                                         pred)
```

`mismatch` finds the first position where the two ranges `[first1, last1)` and `[first2, first2 + (last1 - first1))` differ. The two versions of `mismatch` use different tests for whether elements differ.

This version of `mismatch` finds the first iterator `i` in `[first1, last1)` such that `pred(*i, *(first2 + (i - first1)))` is false. The return value is a pair whose first element is `i` and whose second element is `*(first2 + (i - first1))`. If no such iterator `i` exists, the return value is a pair whose first element is `last1` and whose second element is `*(first2 + (last1 - first1))`.

```
#include <thrust/mismatch.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> vec1(4);
thrust::device_vector<int> vec2(4);

vec1[0] = 0;  vec2[0] = 0;
vec1[1] = 5;  vec2[1] = 5;
vec1[2] = 3;  vec2[2] = 8;
vec1[3] = 7;  vec2[3] = 7;

typedef thrust::device_vector<int>::iterator Iterator;
thrust::pair<Iterator, Iterator> result;

result = thrust::mismatch(vec1.begin(), vec1.end(), vec2.begin(), thrust::equal_to
    ↪<int>());

// result.first  is vec1.begin() + 2
// result.second is vec2.begin() + 2
```

Return The first position where the sequences differ.

Parameters

- `first1`: The beginning of the first sequence.
- `last1`: The end of the first sequence.
- `first2`: The beginning of the second sequence.
- `pred`: The binary predicate to compare elements.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#).
- `InputIterator2`: is a model of [Input Iterator](#).
- `Predicate`: is a model of [Input Iterator](#).

See `find`

See `find_if`

Template Function `thrust::none_of(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)`

- Defined in file `thrust_logical.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::none_of`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename Predicate>__
  ↳host__ __device__ bool thrust::none_of(const thrust::detail::execution_policy_base
  ↳< DerivedPolicy > &, InputIterator, InputIterator, Predicate)
- template<typename InputIterator, typename Predicate>
  bool thrust::none_of(InputIterator, InputIterator, Predicate)
```

Template Function `thrust::none_of(InputIterator, InputIterator, Predicate)`

Function Documentation

template<typename **InputIterator**, typename **Predicate**>

bool thrust::none_of(*InputIterator first, InputIterator last, Predicate pred*)

`none_of` determines whether no element in a range satisfies a predicate. Specifically, `none_of` returns true if there is no iterator `i` in the range `[first, last)` such that `pred(*i)` is true, and false otherwise.

```
#include <thrust/logical.h>
#include <thrust/functional.h>
...
bool A[3] = {true, true, false};

thrust::none_of(A, A + 2, thrust::identity<bool>()); // returns false
thrust::none_of(A, A + 3, thrust::identity<bool>()); // returns false

thrust::none_of(A + 2, A + 3, thrust::identity<bool>()); // returns true

// empty range
thrust::none_of(A, A, thrust::identity<bool>()); // returns true
```

Return true, if no element satisfies the predicate; false, otherwise.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.
- `pred`: A predicate used to test range elements.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#),
- `Predicate`: must be a model of [Predicate](#).

See `all_of`

See `any_of`

See `transform_reduce`

Template Function `thrust::norm`

- Defined in `file_thrust_complex.h`

Function Documentation

```
template<typename T>__host__ __device__ T thrust::norm(const complex < T > & z)
```

Returns the square of the magnitude of a `complex`.

Parameters

- `z`: The `complex` from which to calculate the norm.

Template Function `thrust::not1`

- Defined in `file_thrust_functional.h`

Function Documentation

```
template<typename Predicate>__host__ __device__ unary_negate<Predicate> thrust::not1(const
```

`not1` is a helper function to simplify the creation of Adaptable Predicates: it takes an Adaptable Predicate `pred` as an argument and returns a new Adaptable Predicate that represents the negation of `pred`. That is: if `pred` is an object of a type which models Adaptable Predicate, then the the type of the result `npred` of `not1(pred)` is also a model of Adaptable Predicate and `npred(x)` always returns the same value as `!pred(x)`.

Return A new object, `npred` such that `npred(x)` always returns the same value as `!pred(x)`.

See [*unary_negate*](#)

See [*not2*](#)

Parameters

- `pred`: The Adaptable Predicate to negate.

Template Parameters

- `Predicate`: is a model of [Adaptable Predicate](#).

Template Function `thrust::not2`

- Defined in `file_thrust_functional.h`

Function Documentation

template<typename BinaryPredicate>__host__ __device__ binary_negate<BinaryPredicate> thrust::

`not2` is a helper function to simplify the creation of Adaptable Binary Predicates: it takes an Adaptable Binary Predicate `pred` as an argument and returns a new Adaptable Binary Predicate that represents the negation of `pred`. That is: if `pred` is an object of a type which models Adaptable Binary Predicate, then the the type of the result `npred` of `not2(pred)` is also a model of Adaptable Binary Predicate and `npred(x,y)` always returns the same value as `!pred(x,y)`.

Return A new object, `npred` such that `npred(x,y)` always returns the same value as `!pred(x,y)`.

See [*binary_negate*](#)

See [*not1*](#)

Parameters

- `pred`: The Adaptable Binary Predicate to negate.

Template Parameters

- `Binary::Predicate` is a model of [Adaptable Binary Predicate](#).

Template Function `thrust::operator!=(const complex<T0>&, const complex<T1>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::operator!=`” with arguments `(const complex<T0>&, const complex<T1>&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
  ↳ thrust::operator!=(const complex < T0 > &, const std::complex< T1 > &)
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
  ↳ thrust::operator!=(const std::complex< T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
  ↳ =(const complex < T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
  ↳ =(const complex < T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
  ↳ =(const T0 &, const complex < T1 > &)
- template<typename T1, typename T2>__host__ __device__ bool thrust::operator!
  ↳ =(const pair < T1, T2 > &, const pair < T1, T2 > &)
```

Template Function `thrust::operator!=(const complex<T0>&, const std::complex<T1>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::operator!=`” with arguments `(const complex<T0>&, const std::complex<T1>&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
↳thrust::operator!=(const complex < T0 > &, const std::complex< T1 > &)
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
↳thrust::operator!=(const std::complex< T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const complex < T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const complex < T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const T0 &, const complex < T1 > &)
- template<typename T1, typename T2>__host__ __device__ bool thrust::operator!
↳=(const pair < T1, T2 > &, const pair < T1, T2 > &)

```

Template Function `thrust::operator!=(const std::complex<T0>&, const complex<T1>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator!=” with arguments (const std::complex<T0>&, const complex<T1>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
↳thrust::operator!=(const complex < T0 > &, const std::complex< T1 > &)
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
↳thrust::operator!=(const std::complex< T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const complex < T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const complex < T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const T0 &, const complex < T1 > &)
- template<typename T1, typename T2>__host__ __device__ bool thrust::operator!
↳=(const pair < T1, T2 > &, const pair < T1, T2 > &)

```

Template Function `thrust::operator!=(const T0&, const complex<T1>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator!=” with arguments (const T0&, const complex<T1>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
↳thrust::operator!=(const complex < T0 > &, const std::complex< T1 > &)
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
↳thrust::operator!=(const std::complex< T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const complex < T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const complex < T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const T0 &, const complex < T1 > &)
- template<typename T1, typename T2>__host__ __device__ bool thrust::operator!
↳=(const pair < T1, T2 > &, const pair < T1, T2 > &)

```

Template Function `thrust::operator!=(const complex<T0>&, const T1&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::operator!=`” with arguments `(const complex<T0>&, const T1&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
↳thrust::operator!=(const complex < T0 > &, const std::complex< T1 > &)
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
↳thrust::operator!=(const std::complex< T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const complex < T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const complex < T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const T0 &, const complex < T1 > &)
- template<typename T1, typename T2>__host__ __device__ bool thrust::operator!
↳=(const pair < T1, T2 > &, const pair < T1, T2 > &)
```

Template Function `thrust::operator!=(const pair<T1, T2>&, const pair<T1, T2>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::operator!=`” with arguments `(const pair<T1, T2>&, const pair<T1, T2>&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
↳thrust::operator!=(const complex < T0 > &, const std::complex< T1 > &)
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
↳thrust::operator!=(const std::complex< T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const complex < T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const complex < T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ bool thrust::operator!
↳=(const T0 &, const complex < T1 > &)
- template<typename T1, typename T2>__host__ __device__ bool thrust::operator!
↳=(const pair < T1, T2 > &, const pair < T1, T2 > &)
```

Template Function `thrust::operator*(const complex<T0>&, const complex<T1>&)`

- Defined in file `_thrust_complex.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator*” with arguments (const complex<T0>&, const complex<T1>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator*(const complex <_
↳T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator*(const complex <_
↳T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator*(const T0 &,_
↳const complex < T1 > &)
```

Template Function thrust::operator*(const complex<T0>&, const T1&)

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator*” with arguments (const complex<T0>&, const T1&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator*(const complex <_
↳T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator*(const complex <_
↳T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator*(const T0 &,_
↳const complex < T1 > &)
```

Template Function thrust::operator*(const T0&, const complex<T1>&)

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator*” with arguments (const T0&, const complex<T1>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator*(const complex <_
↳T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator*(const complex <_
↳T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator*(const T0 &,_
↳const complex < T1 > &)
```

Template Function `thrust::operator+(const complex<T0>&, const complex<T1>&)`

- Defined in file `_thrust_complex.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::operator+`” with arguments `(const complex<T0>&, const complex<T1>&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator+(const complex <_
↳T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator+(const complex <_
↳T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator+(const T0 &,_
↳const complex < T1 > &)
- template<typename T>__host__ __device__ complex<T> thrust::operator+(const _
↳complex < T > &)
```

Template Function `thrust::operator+(const complex<T0>&, const T1&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::operator+`” with arguments `(const complex<T0>&, const T1&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator+(const complex <_
↳T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator+(const complex <_
↳T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator+(const T0 &,_
↳const complex < T1 > &)
- template<typename T>__host__ __device__ complex<T> thrust::operator+(const _
↳complex < T > &)
```

Template Function `thrust::operator+(const T0&, const complex<T1>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator+” with arguments (const T0&, const complex<T1>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator+(const complex <_
↳T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator+(const complex <_
↳T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator+(const T0 &,_
↳const complex < T1 > &)
- template<typename T>__host__ __device__ complex<T> thrust::operator+(const _
↳complex < T > &)
```

Template Function thrust::operator+(const complex<T>&)

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator+” with arguments (const complex<T>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator+(const complex <_
↳T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator+(const complex <_
↳T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator+(const T0 &,_
↳const complex < T1 > &)
- template<typename T>__host__ __device__ complex<T> thrust::operator+(const _
↳complex < T > &)
```

Template Function thrust::operator-(const complex<T0>&, const complex<T1>&)

- Defined in file_thrust_complex.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator-” with arguments (const complex<T0>&, const complex<T1>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:


```

- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator-(const complex <_
↳T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator-(const complex <_
↳T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator-(const T0 &,_
↳const complex < T1 > &)
- template<typename T>__host__ __device__ complex<T> thrust::operator-(const _
↳complex < T > &)

```

Template Function thrust::operator-(const complex<T0>&, const T1&)

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator-” with arguments (const complex<T0>&, const T1&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator-(const complex <_
↳T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator-(const complex <_
↳T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator-(const T0 &,_
↳const complex < T1 > &)
- template<typename T>__host__ __device__ complex<T> thrust::operator-(const _
↳complex < T > &)

```

Template Function thrust::operator-(const T0&, const complex<T1>&)

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator-” with arguments (const T0&, const complex<T1>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator-(const complex <_
↳T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator-(const complex <_
↳T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator-(const T0 &,_
↳const complex < T1 > &)
- template<typename T>__host__ __device__ complex<T> thrust::operator-(const _
↳complex < T > &)

```

Template Function `thrust::operator-(const complex<T>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::operator-`” with arguments `(const complex<T>&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_  
  ↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator-(const complex <_  
  ↳T0 > &, const complex < T1 > &)  
- template<typename T0, typename T1>__host__ __device__ complex<typename_  
  ↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator-(const complex <_  
  ↳T0 > &, const T1 &)  
- template<typename T0, typename T1>__host__ __device__ complex<typename_  
  ↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator-(const T0 &,  
  ↳const complex < T1 > &)  
- template<typename T>__host__ __device__ complex<T> thrust::operator-(const  
  ↳complex < T > &)
```

Template Function `thrust::operator/(const complex<T0>&, const complex<T1>&)`

- Defined in file `_thrust_complex.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::operator/`” with arguments `(const complex<T0>&, const complex<T1>&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_  
  ↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator/(const complex <_  
  ↳T0 > &, const complex < T1 > &)  
- template<typename T0, typename T1>__host__ __device__ complex<typename_  
  ↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator/(const complex <_  
  ↳T0 > &, const T1 &)  
- template<typename T0, typename T1>__host__ __device__ complex<typename_  
  ↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator/(const T0 &,  
  ↳const complex < T1 > &)
```

Template Function `thrust::operator/(const complex<T0>&, const T1&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator/” with arguments (const complex<T0>&, const T1&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator/(const complex <_
↳T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator/(const complex <_
↳T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator/(const T0 &,_
↳const complex < T1 > &)
```

Template Function thrust::operator/(const T0&, const complex<T1>&)

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator/” with arguments (const T0&, const complex<T1>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator/(const complex <_
↳T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator/(const complex <_
↳T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::operator/(const T0 &,_
↳const complex < T1 > &)
```

Template Function thrust::operator<

- Defined in file_thrust_tuple.h

Function Documentation

```
template<typename T1, typename T2>__host__ __device__ bool thrust::operator<(const pair <
```

This operator tests two pairs for ascending ordering.

Return true if and only if `x.first < y.first || (!(y.first < x.first) && x.second < y.second)`.

Parameters

- `x`: The first pair to compare.
- `y`: The second pair to compare.

Template Parameters

- T1: is a model of [LessThan Comparable](#).
- T2: is a model of [LessThan Comparable](#).

Template Function `thrust::operator<<`

Function Documentation

```
template<typename T, typename CharT, typename Traits>
std::basic_ostream<CharT, Traits> &thrust::operator<< (std::basic_ostream<CharT, Traits> &os,
                                                    const complex<T> &z)
```

Writes to an output stream a complex number in the form (real, imaginary).

Parameters

- os: The output stream.
- z: The complex number to output.

Template Function `thrust::operator<=`

- Defined in `file_thrust_tuple.h`

Function Documentation

```
template<typename T1, typename T2>__host__ __device__ bool thrust::operator<=(const pair<T1, T2> &x,
```

This operator tests two pairs for ascending ordering or equivalence.

Return `true` if and only if `!(y < x)`.

Parameters

- x: The first pair to compare.
- y: The second pair to compare.

Template Parameters

- T1: is a model of [LessThan Comparable](#).
- T2: is a model of [LessThan Comparable](#).

Template Function `thrust::operator==(const complex<T0>&, const complex<T1>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::operator==`” with arguments `(const complex<T0>&, const complex<T1>&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
→thrust::operator==(const complex < T0 > &, const std::complex< T1 > &)
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
→thrust::operator==(const std::complex< T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const complex < T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const complex < T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const T0 &, const complex < T1 > &)
- template<typename T1, typename T2>__host__ __device__ bool_
→thrust::operator==(const pair < T1, T2 > &, const pair < T1, T2 > &)

```

Template Function thrust::operator==(const complex<T0>&, const std::complex<T1>&)

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator==” with arguments (const complex<T0>&, const std::complex<T1>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
→thrust::operator==(const complex < T0 > &, const std::complex< T1 > &)
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
→thrust::operator==(const std::complex< T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const complex < T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const complex < T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const T0 &, const complex < T1 > &)
- template<typename T1, typename T2>__host__ __device__ bool_
→thrust::operator==(const pair < T1, T2 > &, const pair < T1, T2 > &)

```

Template Function thrust::operator==(const std::complex<T0>&, const complex<T1>&)

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator==” with arguments (const std::complex<T0>&, const complex<T1>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
→thrust::operator==(const complex < T0 > &, const std::complex< T1 > &)
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
→thrust::operator==(const std::complex< T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const complex < T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const complex < T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const T0 &, const complex < T1 > &)
- template<typename T1, typename T2>__host__ __device__ bool_
→thrust::operator==(const pair < T1, T2 > &, const pair < T1, T2 > &)

```

Template Function `thrust::operator==(const T0&, const complex<T1>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::operator==`” with arguments `(const T0&, const complex<T1>&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
→thrust::operator==(const complex < T0 > &, const std::complex< T1 > &)
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
→thrust::operator==(const std::complex< T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const complex < T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const complex < T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const T0 &, const complex < T1 > &)
- template<typename T1, typename T2>__host__ __device__ bool_
→thrust::operator==(const pair < T1, T2 > &, const pair < T1, T2 > &)
```

Template Function `thrust::operator==(const complex<T0>&, const T1&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::operator==`” with arguments `(const complex<T0>&, const T1&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
→thrust::operator==(const complex < T0 > &, const std::complex< T1 > &)
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
→thrust::operator==(const std::complex< T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const complex < T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const complex < T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const T0 &, const complex < T1 > &)
- template<typename T1, typename T2>__host__ __device__ bool_
→thrust::operator==(const pair < T1, T2 > &, const pair < T1, T2 > &)
```

Template Function `thrust::operator==(const pair<T1, T2>&, const pair<T1, T2>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::operator==” with arguments (const pair<T1, T2>&, const pair<T1, T2>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
→thrust::operator==(const complex < T0 > &, const std::complex< T1 > &)
- template<typename T0, typename T1>__host__ THRUST_STD_COMPLEX_DEVICE bool_
→thrust::operator==(const std::complex< T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const complex < T0 > &, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const complex < T0 > &, const T1 &)
- template<typename T0, typename T1>__host__ __device__ bool_
→thrust::operator==(const T0 &, const complex < T1 > &)
- template<typename T1, typename T2>__host__ __device__ bool_
→thrust::operator==(const pair < T1, T2 > &, const pair < T1, T2 > &)
```

Template Function thrust::operator<

Function Documentation

```
template<typename T1, typename T2>__host__ __device__ bool thrust::operator<(const pair <
```

This operator tests two pairs for descending ordering.

Return true if and only if $y < x$.

Parameters

- x : The first pair to compare.
- y : The second pair to compare.

Template Parameters

- T1: is a model of [LessThan Comparable](#).
- T2: is a model of [LessThan Comparable](#).

Template Function thrust::operator>=

Function Documentation

```
template<typename T1, typename T2>__host__ __device__ bool thrust::operator>=(const pair <
```

This operator tests two pairs for descending ordering or equivalence.

Return true if and only if $!(x < y)$.

Parameters

- x : The first pair to compare.
- y : The second pair to compare.

Template Parameters

- T1: is a model of [LessThan Comparable](#).
- T2: is a model of [LessThan Comparable](#).

Template Function `thrust::operator>>`

- Defined in file `_thrust_complex.h`

Function Documentation

```
template<typename T, typename CharT, typename Traits>
std::basic_istream<CharT, Traits> &thrust::operator>> (std::basic_istream<CharT, Traits> &is, com-
                                                    plex<T> &z)
```

Reads a complex number from an input stream.

The recognized formats are:

- real
- (real)
- (real, imaginary)

The values read must be convertible to the complex's `value_type`

Parameters

- `is`: The input stream.
- `z`: The complex number to set.

Template Function `thrust::partition(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate)`

- Defined in file `_thrust_partition.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::partition`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
  ↳ typename Predicate>__host__ __device__ ForwardIterator thrust::partition(const_
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator, _
  ↳ ForwardIterator, InputIterator, Predicate)
- template<typename DerivedPolicy, typename ForwardIterator, typename Predicate>__
  ↳ host__ __device__ ForwardIterator thrust::partition(const_
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator, _
  ↳ ForwardIterator, Predicate)
- template<typename ForwardIterator, typename InputIterator, typename Predicate>
  ↳ ForwardIterator thrust::partition(ForwardIterator, ForwardIterator, InputIterator,
  ↳ Predicate)
- template<typename ForwardIterator, typename Predicate>
  ↳ ForwardIterator thrust::partition(ForwardIterator, ForwardIterator, Predicate)
```


Template Function `thrust::partition(ForwardIterator, ForwardIterator, Predicate)`

Function Documentation

template<typename **ForwardIterator**, typename **Predicate**>

ForwardIterator thrust::partition(*ForwardIterator* first, *ForwardIterator* last, *Predicate* pred)

partition reorders the elements [first, last) based on the function object pred, such that all of the elements that satisfy pred precede the elements that fail to satisfy it. The postcondition is that, for some iterator middle in the range [first, last), pred(*i) is true for every iterator i in the range [first, middle) and false for every iterator i in the range [middle, last). The return value of partition is middle.

Note that the relative order of elements in the two reordered sequences is not necessarily the same as it was in the original sequence. A different algorithm, stable_partition, does guarantee to preserve the relative order.

The following code snippet demonstrates how to use partition to reorder a sequence so that even numbers precede odd numbers.

Return An iterator referring to the first element of the second partition, that is, the sequence of the elements which do not satisfy pred.

Parameters

- first: The beginning of the sequence to reorder.
- last: The end of the sequence to reorder.
- pred: A function object which decides to which partition each element of the sequence [first, last) belongs.

Template Parameters

- ForwardIterator: is a model of [Forward Iterator](#), and ForwardIterator's value_type is convertible to Predicate's argument_type, and ForwardIterator is mutable.
- Predicate: is a model of [Predicate](#).

```
#include <thrust/partition.h>
...
struct is_even
{
    __host__ __device__
    bool operator() (const int &x)
    {
        return (x % 2) == 0;
    }
};
...
int A[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
const int N = sizeof(A)/sizeof(int);
thrust::partition(A, A + N,
                  is_even());
// A is now {2, 4, 6, 8, 10, 1, 3, 5, 7, 9}
```

See <http://www.sgi.com/tech/stl/partition.html>

See `stable_partition`

See `partition_copy`

Template Function `thrust::partition(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, Predicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::partition” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, Predicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
  ↳ typename Predicate>__host__ __device__ ForwardIterator thrust::partition(const_
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator, InputIterator, Predicate)
- template<typename DerivedPolicy, typename ForwardIterator, typename Predicate>__
  ↳ host__ __device__ ForwardIterator thrust::partition(const_
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator, Predicate)
- template<typename ForwardIterator, typename InputIterator, typename Predicate>
  ForwardIterator thrust::partition(ForwardIterator, ForwardIterator, InputIterator,
  ↳ Predicate)
- template<typename ForwardIterator, typename Predicate>
  ForwardIterator thrust::partition(ForwardIterator, ForwardIterator, Predicate)
```

Template Function `thrust::partition(ForwardIterator, ForwardIterator, InputIterator, Predicate)`

Function Documentation

`template<typename ForwardIterator, typename InputIterator, typename Predicate>`

`ForwardIterator thrust::partition(ForwardIterator first, ForwardIterator last, InputIterator stencil, Predicate pred)`

`partition` reorders the elements `[first, last)` based on the function object `pred` applied to a stencil range `[stencil, stencil + (last - first))`, such that all of the elements whose corresponding stencil element satisfies `pred` precede all of the elements whose corresponding stencil element fails to satisfy it. The postcondition is that, for some iterator `middle` in the range `[first, last)`, `pred(*stencil_i)` is true for every iterator `stencil_i` in the range `[stencil, stencil + (middle - first))` and false for every iterator `stencil_i` in the range `[stencil + (middle - first), stencil + (last - first))`. The return value of `stable_partition` is `middle`.

Note that the relative order of elements in the two reordered sequences is not necessarily the same as it was in the original sequence. A different algorithm, `stable_partition`, does guarantee to preserve the relative order.

The following code snippet demonstrates how to use `partition` to reorder a sequence so that even numbers precede odd numbers.

Return An iterator referring to the first element of the second partition, that is, the sequence of the elements whose stencil elements do not satisfy `pred`.

Pre The ranges `[first, last)` and `[stencil, stencil + (last - first))` shall not overlap.

Parameters

- `first`: The beginning of the sequence to reorder.

- `last`: The end of the sequence to reorder.
- `stencil`: The beginning of the stencil sequence.
- `pred`: A function object which decides to which partition each element of the sequence `[first, last)` belongs.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator` is mutable.
- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/partition.h>
...
struct is_even
{
    __host__ __device__
    bool operator()(const int &x)
    {
        return (x % 2) == 0;
    }
};
...
int A[] = {0, 1, 0, 1, 0, 1, 0, 1, 0, 1};
int S[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
const int N = sizeof(A)/sizeof(int);
thrust::partition(A, A + N, S, is_even());
// A is now {1, 1, 1, 1, 1, 0, 0, 0, 0, 0}
// S is unmodified
```

See <http://www.sgi.com/tech/stl/partition.html>

See `stable_partition`

See `partition_copy`

Template Function `thrust::partition_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)`

- Defined in `file_thrust_partition.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::partition_copy`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator1,
  ↳ typename OutputIterator2, typename Predicate>__host__ __device__ thrust::pair
  ↳<OutputIterator1,OutputIterator2> thrust::partition_copy(const_
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳InputIterator, OutputIterator1, OutputIterator2, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator1, typename OutputIterator2, typename Predicate>__host__ _
  ↳_device__ thrust::pair<OutputIterator1,OutputIterator2> thrust::partition_
  ↳copy(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2,
  ↳Predicate)
- template<typename InputIterator, typename OutputIterator1, typename_
  ↳OutputIterator2, typename Predicate>
  ↳thrust::pair<OutputIterator1, OutputIterator2> thrust::partition_
  ↳copy(InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳OutputIterator1, typename OutputIterator2, typename Predicate>
  ↳thrust::pair<OutputIterator1, OutputIterator2> thrust::partition_
  ↳copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
  ↳OutputIterator2, Predicate)

```

Template Function `thrust::partition_copy(InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)`

Function Documentation

```

template<typename InputIterator, typename OutputIterator1, typename OutputIterator2, typename Predicate>
thrust::pair<OutputIterator1, OutputIterator2> thrust::partition_copy (InputIterator first, InputIt-
                                                                    erator last, OutputIterator1
                                                                    out_true, OutputIterator2
                                                                    out_false, Predicate pred)

```

`partition_copy` differs from `partition` only in that the reordered sequence is written to difference output sequences, rather than in place.

`partition_copy` copies the elements `[first, last)` based on the function object `pred`. All of the elements that satisfy `pred` are copied to the range beginning at `out_true` and all the elements that fail to satisfy it are copied to the range beginning at `out_false`.

The following code snippet demonstrates how to use `partition_copy` to separate a sequence into two output sequences of even and odd numbers.

Return A pair `p` such that `p.first` is the end of the output range beginning at `out_true` and `p.second` is the end of the output range beginning at `out_false`.

Pre The input range shall not overlap with either output range.

Parameters

- `first`: The beginning of the sequence to reorder.
- `last`: The end of the sequence to reorder.
- `out_true`: The destination of the resulting sequence of elements which satisfy `pred`.
- `out_false`: The destination of the resulting sequence of elements which fail to satisfy `pred`.

- `pred`: A function object which decides to which partition each element of the sequence `[first, last)` belongs.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `Predicate`'s `argument_type` and `InputIterator`'s `value_type` is convertible to `OutputIterator1` and `OutputIterator2`'s `value_types`.
- `OutputIterator1`: is a model of [Output Iterator](#).
- `OutputIterator2`: is a model of [Output Iterator](#).
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/partition.h>
...
struct is_even
{
    __host__ __device__
    bool operator() (const int &x)
    {
        return (x % 2) == 0;
    }
};
...
int A[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int result[10];
const int N = sizeof(A)/sizeof(int);
int *evens = result;
int *odds = result + 5;
thrust::partition_copy(A, A + N, evens, odds, is_even());
// A remains {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
// result is now {2, 4, 6, 8, 10, 1, 3, 5, 7, 9}
// evens points to {2, 4, 6, 8, 10}
// odds points to {1, 3, 5, 7, 9}
```

Note The relative order of elements in the two reordered sequences is not necessarily the same as it was in the original sequence. A different algorithm, `stable_partition_copy`, does guarantee to preserve the relative order.

See <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2569.pdf>

See `stable_partition_copy`

See `partition`

Template Function `thrust::partition_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, Predicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::partition_copy`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator1,
  ↳ typename OutputIterator2, typename Predicate>__host__ __device__ thrust::pair
  ↳<OutputIterator1,OutputIterator2> thrust::partition_copy(const_
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳InputIterator, OutputIterator1, OutputIterator2, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator1, typename OutputIterator2, typename Predicate>__host__ _
  ↳_device__ thrust::pair<OutputIterator1,OutputIterator2> thrust::partition_
  ↳copy(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2,
  ↳Predicate)
- template<typename InputIterator, typename OutputIterator1, typename_
  ↳OutputIterator2, typename Predicate>
  ↳thrust::pair<OutputIterator1, OutputIterator2> thrust::partition_
  ↳copy(InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳OutputIterator1, typename OutputIterator2, typename Predicate>
  ↳thrust::pair<OutputIterator1, OutputIterator2> thrust::partition_
  ↳copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
  ↳OutputIterator2, Predicate)

```

Template Function `thrust::partition_copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, Predicate)`

Function Documentation

```

template<typename InputIterator1, typename InputIterator2, typename OutputIterator1, typename OutputIterator2>
thrust::pair<OutputIterator1, OutputIterator2> thrust::partition_copy (InputIterator1 first, InputIterator1 last,
                                                                    InputIterator2 stencil,
                                                                    OutputIterator1 out_true,
                                                                    OutputIterator2 out_false, Predicate pred)

```

`partition_copy` differs from `partition` only in that the reordered sequence is written to difference output sequences, rather than in place.

`partition_copy` copies the elements `[first, last)` based on the function object `pred` which is applied to a range of stencil elements. All of the elements whose corresponding stencil element satisfies `pred` are copied to the range beginning at `out_true` and all the elements whose stencil element fails to satisfy it are copied to the range beginning at `out_false`.

The following code snippet demonstrates how to use `partition_copy` to separate a sequence into two output sequences of even and odd numbers.

Return A pair `p` such that `p.first` is the end of the output range beginning at `out_true` and `p.second` is the end of the output range beginning at `out_false`.

Pre The input ranges shall not overlap with either output range.

Parameters

- `first`: The beginning of the sequence to reorder.
- `last`: The end of the sequence to reorder.
- `stencil`: The beginning of the stencil sequence.
- `out_true`: The destination of the resulting sequence of elements which satisfy `pred`.

- `out_false`: The destination of the resulting sequence of elements which fail to satisfy `pred`.
- `pred`: A function object which decides to which partition each element of the sequence `[first, last)` belongs.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `OutputIterator1` and `OutputIterator2`'s `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), and `InputIterator2`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `OutputIterator1`: is a model of [Output Iterator](#).
- `OutputIterator2`: is a model of [Output Iterator](#).
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/partition.h>
#include <thrust/functional.h>
...
int A[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int S[] = {0, 1, 0, 1, 0, 1, 0, 1, 0, 1};
int result[10];
const int N = sizeof(A)/sizeof(int);
int *evens = result;
int *odds = result + 5;
thrust::stable_partition_copy(A, A + N, S, evens, odds, thrust::identity<int>());
// A remains {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
// S remains {0, 1, 0, 1, 0, 1, 0, 1, 0, 1}
// result is now {2, 4, 6, 8, 10, 1, 3, 5, 7, 9}
// evens points to {2, 4, 6, 8, 10}
// odds points to {1, 3, 5, 7, 9}
```

Note The relative order of elements in the two reordered sequences is not necessarily the same as it was in the original sequence. A different algorithm, `stable_partition_copy`, does guarantee to preserve the relative order.

See <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2569.pdf>

See `stable_partition_copy`

See `partition`

Template Function `thrust::partition_point(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate)`

- Defined in `file_thrust_partition.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::partition_point`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename Predicate>__  
→host__ __device__ ForwardIterator thrust::partition_point(const_  
→thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,_  
→ForwardIterator, Predicate)  
- template<typename ForwardIterator, typename Predicate>  
  ForwardIterator thrust::partition_point(ForwardIterator, ForwardIterator,_  
→Predicate)
```

Template Function `thrust::partition_point(ForwardIterator, ForwardIterator, Predicate)`

Function Documentation

template<typename **ForwardIterator**, typename **Predicate**>

ForwardIterator thrust::partition_point(*ForwardIterator* first, *ForwardIterator* last, *Predicate* pred)

`partition_point` returns an iterator pointing to the end of the true partition of a partitioned range. `partition_point` requires the input range `[first, last)` to be a partition; that is, all elements which satisfy `pred` shall appear before those that do not.

```
#include <thrust/partition.h>  
  
struct is_even  
{  
    __host__ __device__  
    bool operator() (const int &x)  
    {  
        return (x % 2) == 0;  
    }  
};  
  
...  
  
int A[] = {2, 4, 6, 8, 10, 1, 3, 5, 7, 9};  
int * B = thrust::partition_point(A, A + 10, is_even());  
// B - A is 5  
// [A, B) contains only even values
```

Return An iterator `mid` such that `all_of(first, mid, pred)` and `none_of(mid, last, pred)` are both true.

Pre The range `[first, last)` shall be partitioned by `pred`.

Note Though similar, `partition_point` is not redundant with `find_if_not`. `partition_point`'s precondition provides an opportunity for a faster implementation.

Parameters

- `first`: The beginning of the range to consider.
- `last`: The end of the range to consider.
- `pred`: A function object which decides to which partition each element of the range `[first, last)` belongs.

Template Parameters

- ForwardIterator: is a model of [Forward Iterator](#), and ForwardIterator's value_type is convertible to Predicate's argument_type.
- Predicate: is a model of [Predicate](#).

See `partition`

See `find_if_not`

Template Function `thrust::polar`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename T0, typename T1>__host__ __device__ complex<typename detail::promoted_nu

Returns a complex with the specified magnitude and phase.

Parameters

- `m`: The magnitude of the returned complex.
- `theta`: The phase of the returned complex in radians.

Template Function `thrust::pow(const complex<T0>&, const complex<T1>&)`

- Defined in `file_thrust_complex.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::pow” with arguments (const complex<T0>&, const complex<T1>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename _
↪detail::promoted_numerical_type<T0, T1>::type> thrust::pow(const complex < T0 > &
↪, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename _
↪detail::promoted_numerical_type<T0, T1>::type> thrust::pow(const complex < T0 > &
↪, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename _
↪detail::promoted_numerical_type<T0, T1>::type> thrust::pow(const T0 &, const _
↪complex < T1 > &)
```

Template Function `thrust::pow(const complex<T0>&, const T1&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::pow” with arguments (const complex<T0>&, const T1&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::pow(const complex < T0 > &
↳, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::pow(const complex < T0 > &
↳, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::pow(const T0 &, const
↳complex < T1 > &)
```

Template Function thrust::pow(const T0&, const complex<T1>&)

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::pow” with arguments (const T0&, const complex<T1>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::pow(const complex < T0 > &
↳, const complex < T1 > &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::pow(const complex < T0 > &
↳, const T1 &)
- template<typename T0, typename T1>__host__ __device__ complex<typename_
↳detail::promoted_numerical_type<T0, T1>::type> thrust::pow(const T0 &, const
↳complex < T1 > &)
```

Template Function thrust::proj

- Defined in file_thrust_complex.h

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::proj(const T & z)

Returns the projection of a complex on the Riemann sphere. For all finite complex it returns the argument. For complexes with a non finite part returns (INFINITY,+/-0) where the sign of the zero matches the sign of the imaginary part of the argument.

Parameters

- z: The complex argument.

Template Function `thrust::raw_pointer_cast`

- Defined in `file_thrust_memory.h`

Function Documentation

template<typename Pointer>__host__ __device__ thrust::detail::pointer_traits<Pointer>::raw_pointer_cast creates a “raw” pointer from a pointer-like type, simply returning the wrapped pointer, should it exist.

Return `ptr.get()`, if the expression is well formed; `ptr`, otherwise.

See `raw_reference_cast`

Parameters

- `ptr`: The pointer of interest.

Template Function `thrust::raw_reference_cast(T&)`

- Defined in `file_thrust_memory.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::raw_reference_cast`” with arguments (T&) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T>__host__ __device__ detail::raw_reference<T>::type_
↳thrust::raw_reference_cast(T &)
- template<typename T>__host__ __device__ detail::raw_reference<const T>::type_
↳thrust::raw_reference_cast(const T &)
```

Template Function `thrust::raw_reference_cast(const T&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::raw_reference_cast`” with arguments (const T&) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T>__host__ __device__ detail::raw_reference<T>::type_
↳thrust::raw_reference_cast(T &)
- template<typename T>__host__ __device__ detail::raw_reference<const T>::type_
↳thrust::raw_reference_cast(const T &)
```

Template Function `thrust::reduce(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator)`

- Defined in file `_thrust_reduce.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::reduce`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename T, typename
  ↳ BinaryFunction>__host__ __device__ T thrust::reduce(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳ InputIterator, T, BinaryFunction)
- template<typename DerivedPolicy, typename InputIterator, typename T>__host__ __
  ↳ device__ T thrust::reduce(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, InputIterator, InputIterator, T)
- template<typename DerivedPolicy, typename InputIterator>__host__ __device__
  ↳ thrust::iterator_traits<InputIterator>::value_type thrust::reduce(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳ InputIterator)
- template<typename InputIterator, typename T, typename BinaryFunction>
  T thrust::reduce(InputIterator, InputIterator, T, BinaryFunction)
- template<typename InputIterator, typename T>
  T thrust::reduce(InputIterator, InputIterator, T)
- template<typename InputIterator>
  thrust::iterator_traits<InputIterator>::value_type thrust::reduce(InputIterator,
  ↳ InputIterator)
```

Template Function `thrust::reduce(InputIterator, InputIterator)`

Function Documentation

`template<typename InputIterator>`

`thrust::iterator_traits<InputIterator>::value_type thrust::reduce (InputIterator first, InputIterator last)`

`reduce` is a generalization of summation: it computes the sum (or some other binary operation) of all the elements in the range `[first, last)`. This version of `reduce` uses 0 as the initial value of the reduction. `reduce` is similar to the C++ Standard Template Library’s `std::accumulate`. The primary difference between the two functions is that `std::accumulate` guarantees the order of summation, while `reduce` requires associativity of the binary operation to parallelize the reduction.

Note that `reduce` also assumes that the binary reduction operator (in this case `operator+`) is commutative. If the reduction operator is not commutative then `thrust::reduce` should not be used. Instead, one could use `inclusive_scan` (which does not require commutativity) and select the last element of the output array.

The following code snippet demonstrates how to use `reduce` to compute the sum of a sequence of integers.

Return The result of the reduction.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.

Template Parameters

- `InputIterator`: is a model of `Input Iterator` and if `x` and `y` are objects of `InputIterator`'s `value_type`, then `x + y` is defined and is convertible to `InputIterator`'s `value_type`. If `T` is `InputIterator`'s `value_type`, then `T(0)` is defined.

```
#include <thrust/reduce.h>
...
int data[6] = {1, 0, 2, 2, 1, 3};
int result = thrust::reduce(data, data + 6);

// result == 9
```

See <http://www.sgi.com/tech/stl/accumulate.html>

Template Function `thrust::reduce(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, T)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::reduce`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, T)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename T, typename
  ↳ BinaryFunction>__host__ __device__ T thrust::reduce(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳ InputIterator, T, BinaryFunction)
- template<typename DerivedPolicy, typename InputIterator, typename T>__host__ __
  ↳ device__ T thrust::reduce(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, InputIterator, InputIterator, T)
- template<typename DerivedPolicy, typename InputIterator>__host__ __device__
  ↳ thrust::iterator_traits<InputIterator>::value_type thrust::reduce(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳ InputIterator)
- template<typename InputIterator, typename T, typename BinaryFunction>
  T thrust::reduce(InputIterator, InputIterator, T, BinaryFunction)
- template<typename InputIterator, typename T>
  T thrust::reduce(InputIterator, InputIterator, T)
- template<typename InputIterator>
  thrust::iterator_traits<InputIterator>::value_type thrust::reduce(InputIterator,
  ↳ InputIterator)
```

Template Function `thrust::reduce(InputIterator, InputIterator, T)`

Function Documentation

`template<typename InputIterator, typename T>`

`T thrust::reduce (InputIterator first, InputIterator last, T init)`

`reduce` is a generalization of summation: it computes the sum (or some other binary operation) of all the elements in the range `[first, last)`. This version of `reduce` uses `init` as the initial value of the reduction. `reduce` is similar to the C++ Standard Template Library’s `std::accumulate`. The primary difference

between the two functions is that `std::accumulate` guarantees the order of summation, while `reduce` requires associativity of the binary operation to parallelize the reduction.

Note that `reduce` also assumes that the binary reduction operator (in this case `operator+`) is commutative. If the reduction operator is not commutative then `thrust::reduce` should not be used. Instead, one could use `inclusive_scan` (which does not require commutativity) and select the last element of the output array.

The following code snippet demonstrates how to use `reduce` to compute the sum of a sequence of integers including an initialization value.

Return The result of the reduction.

Parameters

- `first`: The beginning of the input sequence.
- `last`: The end of the input sequence.
- `init`: The initial value.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#) and if `x` and `y` are objects of `InputIterator`'s `value_type`, then `x + y` is defined and is convertible to `T`.
- `T`: is convertible to `InputIterator`'s `value_type`.

```
#include <thrust/reduce.h>
...
int data[6] = {1, 0, 2, 2, 1, 3};
int result = thrust::reduce(data, data + 6, 1);

// result == 10
```

See <http://www.sgi.com/tech/stl/accumulate.html>

Template Function `thrust::reduce(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, T, BinaryFunction)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::reduce`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, T, BinaryFunction)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename T, typename
  ↳ BinaryFunction>__host__ __device__ T thrust::reduce(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳ InputIterator, T, BinaryFunction)
- template<typename DerivedPolicy, typename InputIterator, typename T>__host__ __
  ↳ device__ T thrust::reduce(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, InputIterator, InputIterator, T)
- template<typename DerivedPolicy, typename InputIterator>__host__ __device__
  ↳ thrust::iterator_traits<InputIterator>::value_type thrust::reduce(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳ InputIterator)
- template<typename InputIterator, typename T, typename BinaryFunction>
  T thrust::reduce(InputIterator, InputIterator, T, BinaryFunction)
- template<typename InputIterator, typename T>
  T thrust::reduce(InputIterator, InputIterator, T)
- template<typename InputIterator>
  thrust::iterator_traits<InputIterator>::value_type thrust::reduce(InputIterator,
  ↳ InputIterator)
```

Template Function `thrust::reduce(InputIterator, InputIterator, T, BinaryFunction)`

Function Documentation

template<typename **InputIterator**, typename **T**, typename **BinaryFunction**>

T thrust::reduce(*InputIterator first*, *InputIterator last*, *T init*, *BinaryFunction binary_op*)

`reduce` is a generalization of summation: it computes the sum (or some other binary operation) of all the elements in the range `[first, last)`. This version of `reduce` uses `init` as the initial value of the reduction and `binary_op` as the binary function used for summation. `reduce` is similar to the C++ Standard Template Library's `std::accumulate`. The primary difference between the two functions is that `std::accumulate` guarantees the order of summation, while `reduce` requires associativity of `binary_op` to parallelize the reduction.

Note that `reduce` also assumes that the binary reduction operator (in this case `binary_op`) is commutative. If the reduction operator is not commutative then `thrust::reduce` should not be used. Instead, one could use `inclusive_scan` (which does not require commutativity) and select the last element of the output array.

The following code snippet demonstrates how to use `reduce` to compute the maximum value of a sequence of integers.

Return The result of the reduction.

Parameters

- `first`: The beginning of the input sequence.
- `last`: The end of the input sequence.
- `init`: The initial value.
- `binary_op`: The binary function used to 'sum' values.

Template Parameters

- `InputIterator`: is a model of `Input Iterator` and `InputIterator`'s `value_type` is convertible to `T`.
- `T`: is a model of `Assignable`, and is convertible to `BinaryFunction`'s `first_argument_type` and `second_argument_type`.
- `BinaryFunction`: is a model of `Binary Function`, and `BinaryFunction`'s `result_type` is convertible to `OutputType`.

```
#include <thrust/reduce.h>
#include <thrust/functional.h>
...
int data[6] = {1, 0, 2, 2, 1, 3};
int result = thrust::reduce(data, data + 6,
                           -1,
                           thrust::maximum<int>());

// result == 3
```

See <http://www.sgi.com/tech/stl/accumulate.html>

See `transform_reduce`

Template Function `thrust::reduce_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)`

- Defined in file `_thrust_reduce.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::reduce_by_key`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `InputIterator1`, `InputIterator1`, `InputIterator2`, `OutputIterator1`, `OutputIterator2`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename OutputIterator1, typename OutputIterator2, typename BinaryPredicate,
    ↪ typename BinaryFunction>__host__ __device__ thrust::pair<OutputIterator1,
    ↪ OutputIterator2> thrust::reduce_by_key(const thrust::detail::execution_policy_base
    ↪ < DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
    ↪ OutputIterator1, OutputIterator2, BinaryPredicate, BinaryFunction)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename OutputIterator1, typename OutputIterator2, typename BinaryPredicate>__
    ↪ host__ __device__ thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_
    ↪ key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
    ↪ InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2,
    ↪ BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename OutputIterator1, typename OutputIterator2>__host__ __device__
    ↪ thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_key(const
    ↪ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
    ↪ InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ OutputIterator1, typename OutputIterator2, typename BinaryPredicate, typename
    ↪ BinaryFunction>
    ↪ thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
    ↪ OutputIterator2, BinaryPredicate, BinaryFunction)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ OutputIterator1, typename OutputIterator2, typename BinaryPredicate>
    ↪ thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
    ↪ OutputIterator2, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ OutputIterator1, typename OutputIterator2>
    ↪ thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
    ↪ OutputIterator2)
```

Template Function `thrust::reduce_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator1**, typename **OutputItera**


```
thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_key(InputIterator1 keys_first, InputIterator1 keys_last, InputIterator2 values_first, OutputIterator1 keys_output, OutputIterator2 values_output)
```

`reduce_by_key` is a generalization of `reduce` to key-value pairs. For each group of consecutive keys in the range `[keys_first, keys_last)` that are equal, `reduce_by_key` copies the first element of the group to the `keys_output`. The corresponding values in the range are reduced using the `plus` and the result copied to `values_output`.

This version of `reduce_by_key` uses the function object `equal_to` to test for equality and `plus` to reduce values with equal keys.

The following code snippet demonstrates how to use `reduce_by_key` to compact a sequence of key/value pairs and sum values with equal keys.

Return A pair of iterators at end of the ranges `[keys_output, keys_output_last)` and `[values_output, values_output_last)`.

Pre The input ranges shall not overlap either output range.

Parameters

- `keys_first`: The beginning of the input key range.
- `keys_last`: The end of the input key range.
- `values_first`: The beginning of the input value range.
- `keys_output`: The beginning of the output key range.
- `values_output`: The beginning of the output value range.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#),
- `InputIterator2`: is a model of [Input Iterator](#),
- `OutputIterator1`: is a model of [Output Iterator](#) and `InputIterator1`'s `value_type` is convertible to `OutputIterator1`'s `value_type`.
- `OutputIterator2`: is a model of [Output Iterator](#) and `InputIterator2`'s `value_type` is convertible to `OutputIterator2`'s `value_type`.

```
#include <thrust/reduce.h>
...
const int N = 7;
int A[N] = {1, 3, 3, 3, 2, 2, 1}; // input keys
int B[N] = {9, 8, 7, 6, 5, 4, 3}; // input values
int C[N]; // output keys
int D[N]; // output values

thrust::pair<int*, int*> new_end;
new_end = thrust::reduce_by_key(A, A + N, B, C, D);

// The first four keys in C are now {1, 3, 2, 1} and new_end.first - C is 4.
// The first four values in D are now {9, 21, 9, 3} and new_end.second - D is 4.
```

See `reduce`

See `unique_copy`

See `unique_by_key`

See `unique_by_key_copy`

Template Function `thrust::reduce_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::reduce_by_key`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `InputIterator1`, `InputIterator1`, `InputIterator2`, `OutputIterator1`, `OutputIterator2`, `BinaryPredicate`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename OutputIterator1, typename OutputIterator2, typename BinaryPredicate,
    ↪ typename BinaryFunction>__host__ __device__ thrust::pair<OutputIterator1,
    ↪ OutputIterator2> thrust::reduce_by_key(const thrust::detail::execution_policy_base
    ↪ < DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
    ↪ OutputIterator1, OutputIterator2, BinaryPredicate, BinaryFunction)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename OutputIterator1, typename OutputIterator2, typename BinaryPredicate>__
    ↪ host__ __device__ thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_
    ↪ key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
    ↪ InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2,
    ↪ BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename OutputIterator1, typename OutputIterator2>__host__ __device__
    ↪ thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_key(const
    ↪ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
    ↪ InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ OutputIterator1, typename OutputIterator2, typename BinaryPredicate, typename
    ↪ BinaryFunction>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
    ↪ OutputIterator2, BinaryPredicate, BinaryFunction)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ OutputIterator1, typename OutputIterator2, typename BinaryPredicate>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
    ↪ OutputIterator2, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ OutputIterator1, typename OutputIterator2>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
    ↪ OutputIterator2)
```

Template Function `thrust::reduce_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator1**, typename **OutputIterator2**> thrust::pair<**OutputIterator1**, **OutputIterator2**> thrust::reduce_by_key(*InputIterator1 keys_first, InputIterator1 keys_last, InputIterator2 values_first, OutputIterator1 keys_output, OutputIterator2 values_output, BinaryPredicate binary_pred*)

reduce_by_key is a generalization of reduce to key-value pairs. For each group of consecutive keys in the range [keys_first, keys_last) that are equal, reduce_by_key copies the first element of the group to the keys_output. The corresponding values in the range are reduced using the plus and the result copied to values_output.

This version of reduce_by_key uses the function object binary_pred to test for equality and plus to reduce values with equal keys.

The following code snippet demonstrates how to use reduce_by_key to compact a sequence of key/value pairs and sum values with equal keys.

Return A pair of iterators at end of the ranges [keys_output, keys_output_last) and [values_output, values_output_last).

Pre The input ranges shall not overlap either output range.

Parameters

- keys_first: The beginning of the input key range.
- keys_last: The end of the input key range.
- values_first: The beginning of the input value range.
- keys_output: The beginning of the output key range.
- values_output: The beginning of the output value range.
- binary_pred: The binary predicate used to determine equality.

Template Parameters

- InputIterator1: is a model of [Input Iterator](#),
- InputIterator2: is a model of [Input Iterator](#),
- OutputIterator1: is a model of [Output Iterator](#) and InputIterator1's value_type is convertible to OutputIterator1's value_type.
- OutputIterator2: is a model of [Output Iterator](#) and InputIterator2's value_type is convertible to OutputIterator2's value_type.
- BinaryPredicate: is a model of [Binary Predicate](#).

```
#include <thrust/reduce.h>
...
const int N = 7;
int A[N] = {1, 3, 3, 3, 2, 2, 1}; // input keys
int B[N] = {9, 8, 7, 6, 5, 4, 3}; // input values
int C[N]; // output keys
int D[N]; // output values

thrust::pair<int*, int*> new_end;
```

(continues on next page)

(continued from previous page)

```
thrust::equal_to<int> binary_pred;
new_end = thrust::reduce_by_key(A, A + N, B, C, D, binary_pred);

// The first four keys in C are now {1, 3, 2, 1} and new_end.first - C is 4.
// The first four values in D are now {9, 21, 9, 3} and new_end.second - D is 4.
```

See `reduce`

See `unique_copy`

See `unique_by_key`

See `unique_by_key_copy`

Template Function `thrust::reduce_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate, BinaryFunction)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::reduce_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate, BinaryFunction) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator1, typename OutputIterator2, typename BinaryPredicate,
  ↳ typename BinaryFunction>__host__ __device__ thrust::pair<OutputIterator1,
  ↳ OutputIterator2> thrust::reduce_by_key(const thrust::detail::execution_policy_base
  ↳ < DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputIterator1, OutputIterator2, BinaryPredicate, BinaryFunction)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator1, typename OutputIterator2, typename BinaryPredicate>__
  ↳ host__ __device__ thrust::pair<OutputIterator1,OutputIterator2> thrust::reduce_by_
  ↳ key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2,
  ↳ BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator1, typename OutputIterator2>__host__ __device__
  ↳ thrust::pair<OutputIterator1,OutputIterator2> thrust::reduce_by_key(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator1, typename OutputIterator2, typename BinaryPredicate, typename
  ↳ BinaryFunction>
  ↳ thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_
  ↳ key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
  ↳ OutputIterator2, BinaryPredicate, BinaryFunction)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator1, typename OutputIterator2, typename BinaryPredicate>
  ↳ thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_
  ↳ key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
  ↳ OutputIterator2, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator1, typename OutputIterator2>
  ↳ thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_
  ↳ key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
```

Template Function `thrust::reduce_by_key(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate, BinaryFunction)`

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename OutputIterator1, typename OutputIterator2>
thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_key(InputIterator1 keys_first, InputIterator1 keys_last, InputIterator2 values_first, OutputIterator1 keys_output, OutputIterator2 values_output, BinaryPredicate binary_pred, BinaryFunction binary_op)
```

`reduce_by_key` is a generalization of `reduce` to key-value pairs. For each group of consecutive keys in the range `[keys_first, keys_last)` that are equal, `reduce_by_key` copies the first element of the group to the `keys_output`. The corresponding values in the range are reduced using the `BinaryFunction` `binary_op` and the result copied to `values_output`. Specifically, if consecutive key iterators `i` and `(i + 1)` are such that `binary_pred(*i, *(i+1))` is true, then the corresponding values are reduced to a single value with `binary_op`.

This version of `reduce_by_key` uses the function object `binary_pred` to test for equality and `binary_op` to reduce values with equal keys.

The following code snippet demonstrates how to use `reduce_by_key` to compact a sequence of key/value pairs and sum values with equal keys.

Return A pair of iterators at end of the ranges `[keys_output, keys_output_last)` and `[values_output, values_output_last)`.

Pre The input ranges shall not overlap either output range.

Parameters

- `keys_first`: The beginning of the input key range.
- `keys_last`: The end of the input key range.
- `values_first`: The beginning of the input value range.
- `keys_output`: The beginning of the output key range.
- `values_output`: The beginning of the output value range.
- `binary_pred`: The binary predicate used to determine equality.
- `binary_op`: The binary function used to accumulate values.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#),
- `InputIterator2`: is a model of [Input Iterator](#),
- `OutputIterator1`: is a model of [Output Iterator](#) and `InputIterator1`'s `value_type` is convertible to `OutputIterator1`'s `value_type`.
- `OutputIterator2`: is a model of [Output Iterator](#) and `InputIterator2`'s `value_type` is convertible to `OutputIterator2`'s `value_type`.

- BinaryPredicate: is a model of Binary Predicate.
- BinaryFunction: is a model of Binary Function and BinaryFunction's result_type is convertible to OutputIterator2's value_type.

```
#include <thrust/reduce.h>
...
const int N = 7;
int A[N] = {1, 3, 3, 3, 2, 2, 1}; // input keys
int B[N] = {9, 8, 7, 6, 5, 4, 3}; // input values
int C[N]; // output keys
int D[N]; // output values

thrust::pair<int*,int*> new_end;
thrust::equal_to<int> binary_pred;
thrust::plus<int> binary_op;
new_end = thrust::reduce_by_key(A, A + N, B, C, D, binary_pred, binary_op);

// The first four keys in C are now {1, 3, 2, 1} and new_end.first - C is 4.
// The first four values in D are now {9, 21, 9, 3} and new_end.second - D is 4.
```

See `reduce`

See `unique_copy`

See `unique_by_key`

See `unique_by_key_copy`

Template Function `thrust::remove(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::remove” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename T>__host__ __
  ↪device__ ForwardIterator thrust::remove(const thrust::detail::execution_policy_
  ↪base< DerivedPolicy > &, ForwardIterator, ForwardIterator, const T &)
- template<typename ForwardIterator, typename T>
  ForwardIterator thrust::remove(ForwardIterator, ForwardIterator, const T&)
```

Template Function `thrust::remove(ForwardIterator, ForwardIterator, const T&)`

Function Documentation

template<typename **ForwardIterator**, typename **T**>

ForwardIterator thrust::remove(*ForwardIterator* first, *ForwardIterator* last, const *T* &value)

`remove` removes from the range `[first, last)` all elements that are equal to `value`. That is, `remove` returns an iterator `new_last` such that the range `[first, new_last)` contains no elements equal to `value`. The iterators in the range `[new_first, last)` are all still dereferenceable, but the elements that they point

to are unspecified. `remove` is stable, meaning that the relative order of elements that are not equal to `value` is unchanged.

The following code snippet demonstrates how to use `remove` to remove a number of interest from a range.

Return A `ForwardIterator` pointing to the end of the resulting range of elements which are not equal to `value`.

Parameters

- `first`: The beginning of the range of interest.
- `last`: The end of the range of interest.
- `value`: The value to remove from the range `[first, last)`. Elements which are equal to `value` are removed from the sequence.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator` is mutable.
- `T`: is a model of [Equality Comparable](#), and objects of type `T` can be compared for equality with objects of `ForwardIterator`'s `value_type`.

```
#include <thrust/remove.h>
...
const int N = 6;
int A[N] = {3, 1, 4, 1, 5, 9};
int *new_end = thrust::remove(A, A + N, 1);
// The first four values of A are now {3, 4, 5, 9}
// Values beyond new_end are unspecified
```

Note The meaning of “removal” is somewhat subtle. `remove` does not destroy any iterators, and does not change the distance between `first` and `last`. (There’s no way that it could do anything of the sort.) So, for example, if `V` is a [device_vector](#), `remove(V.begin(), V.end(), 0)` does not change `V.size()`: `V` will contain just as many elements as it did before. `remove` returns an iterator that points to the end of the resulting range after elements have been removed from it; it follows that the elements after that iterator are of no interest, and may be discarded. If you are removing elements from a [Sequence](#), you may simply erase them. That is, a reasonable way of removing elements from a [Sequence](#) is `S.erase(remove(S.begin(), S.end(), x), S.end())`.

See <http://www.sgi.com/tech/stl/remove.html>

See `remove_if`

See `remove_copy`

See `remove_copy_if`

Template Function `thrust::remove_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, const T&)`

- Defined in `file_thrust_remove.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::remove_copy” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, const T&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
↳ typename T>__host__ __device__ OutputIterator thrust::remove_copy(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
↳ InputIterator, OutputIterator, const T &)
- template<typename InputIterator, typename OutputIterator, typename T>
  OutputIterator thrust::remove_copy(InputIterator, InputIterator, OutputIterator,
↳ const T&)
```

Template Function thrust::remove_copy(InputIterator, InputIterator, OutputIterator, const T&)

Function Documentation

```
template<typename InputIterator, typename OutputIterator, typename T>
```

```
OutputIterator thrust::remove_copy(InputIterator first, InputIterator last, OutputIterator result,
                                  const T &value)
```

remove_copy copies elements that are not equal to value from the range [first, last) to a range beginning at result. The return value is the end of the resulting range. This operation is stable, meaning that the relative order of the elements that are copied is the same as in the range [first, last).

The following code snippet demonstrates how to use remove_copy to copy a sequence of numbers to an output range while omitting a value of interest.

Return An OutputIterator pointing to the end of the resulting range of elements which are not equal to value.

Pre The range [first, last) shall not overlap the range [result, result + (last - first)).

Parameters

- first: The beginning of the range of interest.
- last: The end of the range of interest.
- result: The resulting range is copied to the sequence beginning at this location.
- value: The value to omit from the copied range.

Template Parameters

- InputIterator: is a model of [Input Iterator](#), and InputIterator's value_type is convertible to a type in OutputIterator's set of value_types.
- OutputIterator: is a model of [Output Iterator](#).
- T: is a model of [Equality Comparable](#), and objects of type T can be compared for equality with objects of InputIterator's value_type.

```
#include <thrust/remove.h>
...
const int N = 6;
int V[N] = {-2, 0, -1, 0, 1, 2};
```

(continues on next page)

(continued from previous page)

```
int result[N-2];
thrust::remove_copy(V, V + N, result, 0);
// V remains {-2, 0, -1, 0, 1, 2}
// result is now {-2, -1, 1, 2}
```

See http://www.sgi.com/tech/stl/remove_copy.html

See `remove`

See `remove_if`

See `remove_copy_if`

Template Function `thrust::remove_copy_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, Predicate)`

- Defined in `file_thrust_remove.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::remove_copy_if`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `InputIterator`, `InputIterator`, `OutputIterator`, `Predicate`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
→ typename Predicate>__host__ __device__ OutputIterator thrust::remove_copy_
→ if(const thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
→ InputIterator, OutputIterator, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
→ typename OutputIterator, typename Predicate>__host__ __device__ OutputIterator
→ thrust::remove_copy_if(const thrust::detail::execution_policy_base< DerivedPolicy
→ > &, InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate)
- template<typename InputIterator, typename OutputIterator, typename Predicate>
OutputIterator thrust::remove_copy_if(InputIterator, InputIterator,
→ OutputIterator, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
→ OutputIterator, typename Predicate>
OutputIterator thrust::remove_copy_if(InputIterator1, InputIterator1,
→ InputIterator2, OutputIterator, Predicate)
```

Template Function `thrust::remove_copy_if(InputIterator, InputIterator, OutputIterator, Predicate)`

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**, typename **Predicate**>

OutputIterator thrust::remove_copy_if(*InputIterator* first, *InputIterator* last, *OutputIterator* result, *Predicate* pred)

`remove_copy_if` copies elements from the range `[first, last)` to a range beginning at `result`, except that elements for which `pred` is `true` are not copied. The return value is the end of the resulting range. This operation is stable, meaning that the relative order of the elements that are copied is the same as the range `[first, last)`.

The following code snippet demonstrates how to use `remove_copy_if` to copy a sequence of numbers to an output range while omitting even numbers.

Return An `OutputIterator` pointing to the end of the resulting range.

Pre The range `[first, last)` shall not overlap the range `[result, result + (last - first))`.

Parameters

- `first`: The beginning of the range of interest.
- `last`: The end of the range of interest.
- `result`: The resulting range is copied to the sequence beginning at this location.
- `pred`: A predicate to evaluate for each element of the range `[first, last)`. Elements for which `pred` evaluates to `false` are not copied to the resulting sequence.

Template Parameters

- `InputIterator`: is a model of `Input Iterator`, `InputIterator`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`, and `InputIterator`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `OutputIterator`: is a model of `Output Iterator`.
- `Predicate`: is a model of `Predicate`.

```
#include <thrust/remove.h>
...
struct is_even
{
    __host__ __device__
    bool operator() (const int x)
    {
        return (x % 2) == 0;
    }
};
...
const int N = 6;
int V[N] = {-2, 0, -1, 0, 1, 2};
int result[2];
thrust::remove_copy_if(V, V + N, result, is_even());
// V remains {-2, 0, -1, 0, 1, 2}
// result is now {-1, 1}
```

See http://www.sgi.com/tech/stl/remove_copy_if.html

See `remove`

See `remove_copy`

See `remove_if`

Template Function `thrust::remove_copy_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::remove_copy_if” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
→typename Predicate>__host__ __device__ OutputIterator thrust::remove_copy_
→if(const thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
→InputIterator, OutputIterator, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
→ typename OutputIterator, typename Predicate>__host__ __device__ OutputIterator
→thrust::remove_copy_if(const thrust::detail::execution_policy_base< DerivedPolicy
→> &, InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate)
- template<typename InputIterator, typename OutputIterator, typename Predicate>
  OutputIterator thrust::remove_copy_if(InputIterator, InputIterator,
→OutputIterator, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
→OutputIterator, typename Predicate>
  OutputIterator thrust::remove_copy_if(InputIterator1, InputIterator1,
→InputIterator2, OutputIterator, Predicate)
```

Template Function `thrust::remove_copy_if(InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate)`

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename OutputIterator, typename Predicate>
OutputIterator thrust::remove_copy_if (InputIterator1 first, InputIterator1 last, InputIterator2 stencil,
                                     OutputIterator result, Predicate pred)
```

`remove_copy_if` copies elements from the range `[first, last)` to a range beginning at `result`, except that elements for which `pred` of the corresponding stencil value is `true` are not copied. The return value is the end of the resulting range. This operation is stable, meaning that the relative order of the elements that are copied is the same as the range `[first, last)`.

The following code snippet demonstrates how to use `remove_copy_if` to copy a sequence of numbers to an output range while omitting specific elements.

Return An `OutputIterator` pointing to the end of the resulting range.

Pre The range `[stencil, stencil + (last - first))` shall not overlap the range `[result, result + (last - first))`.

Parameters

- `first`: The beginning of the range of interest.
- `last`: The end of the range of interest.
- `stencil`: The beginning of the stencil sequence.
- `result`: The resulting range is copied to the sequence beginning at this location.
- `pred`: A predicate to evaluate for each element of the range `[first, last)`. Elements for which `pred` evaluates to `false` are not copied to the resulting sequence.

Template Parameters

- InputIterator1: is a model of [Input Iterator](#), InputIterator1's value_type is convertible to a type in OutputIterator's set of value_types.
- InputIterator2: is a model of [Input Iterator](#), and InputIterator2's value_type is convertible to Predicate's argument_type.
- OutputIterator: is a model of [Output Iterator](#).
- Predicate: is a model of [Predicate](#).

```
#include <thrust/remove.h>
...
const int N = 6;
int V[N] = {-2, 0, -1, 0, 1, 2};
int S[N] = { 1, 1, 0, 1, 0, 1};
int result[2];
thrust::remove_copy_if(V, V + N, S, result, thrust::identity<int>());
// V remains {-2, 0, -1, 0, 1, 2}
// result is now {-1, 1}
```

See http://www.sgi.com/tech/stl/remove_copy_if.html

See [remove](#)

See [remove_copy](#)

See [remove_if](#)

See [copy_if](#)

Template Function `thrust::remove_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate)`

- Defined in file `thrust_remove.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::remove_if” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
  ↳ typename Predicate>__host__ __device__ ForwardIterator thrust::remove_if(const_
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator, InputIterator, Predicate)
- template<typename DerivedPolicy, typename ForwardIterator, typename Predicate>__
  ↳ host__ __device__ ForwardIterator thrust::remove_if(const_
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator, Predicate)
- template<typename ForwardIterator, typename InputIterator, typename Predicate>
  ↳ ForwardIterator thrust::remove_if(ForwardIterator, ForwardIterator, InputIterator,
  ↳ Predicate)
- template<typename ForwardIterator, typename Predicate>
  ↳ ForwardIterator thrust::remove_if(ForwardIterator, ForwardIterator, Predicate)
```

Template Function `thrust::remove_if(ForwardIterator, ForwardIterator, Predicate)`

Function Documentation

template<typename **ForwardIterator**, typename **Predicate**>

ForwardIterator thrust::remove_if(*ForwardIterator first*, *ForwardIterator last*, *Predicate pred*)

`remove_if` removes from the range `[first, last)` every element `x` such that `pred(x)` is `true`. That is, `remove_if` returns an iterator `new_last` such that the range `[first, new_last)` contains no elements for which `pred` is `true`. The iterators in the range `[new_last, last)` are all still dereferenceable, but the elements that they point to are unspecified. `remove_if` is stable, meaning that the relative order of elements that are not removed is unchanged.

The following code snippet demonstrates how to use `remove_if` to remove all even numbers from an array of integers.

Return A `ForwardIterator` pointing to the end of the resulting range of elements for which `pred` evaluated to `true`.

Parameters

- `first`: The beginning of the range of interest.
- `last`: The end of the range of interest.
- `pred`: A predicate to evaluate for each element of the range `[first, last)`. Elements for which `pred` evaluates to `true` are removed from the sequence.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), `ForwardIterator` is mutable, and `ForwardIterator`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/remove.h>
...
struct is_even
{
    __host__ __device__
    bool operator()(const int x)
    {
        return (x % 2) == 0;
    }
};
...
const int N = 6;
int A[N] = {1, 4, 2, 8, 5, 7};
int *new_end = thrust::remove_if(A, A + N, is_even());
// The first three values of A are now {1, 5, 7}
// Values beyond new_end are unspecified
```

Note The meaning of “removal” is somewhat subtle. `remove_if` does not destroy any iterators, and does not change the distance between `first` and `last`. (There’s no way that it could do anything of the sort.) So, for example, if `V` is a [device_vector](#), `remove_if(V.begin(), V.end(), pred)` does not change `V.size()`: `V` will contain just as many elements as it did before. `remove_if` returns an iterator that points to the end of the resulting range after elements have been removed from it; it follows that the elements after that iterator are of no interest, and may be discarded. If you are removing elements from a [Sequence](#), you may simply erase them. That is, a reasonable way of removing elements from a [Sequence](#) is `S.erase(remove_if(S.begin(), S.end(), pred), S.end())`.

See http://www.sgi.com/tech/stl/remove_if.html

See `remove`

See `remove_copy`

See `remove_copy_if`

Template Function `thrust::remove_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, Predicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::remove_if” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, Predicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
→ typename Predicate>__host__ __device__ ForwardIterator thrust::remove_if(const_
→thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
→ForwardIterator, InputIterator, Predicate)
- template<typename DerivedPolicy, typename ForwardIterator, typename Predicate>__
→host__ __device__ ForwardIterator thrust::remove_if(const_
→thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
→ForwardIterator, Predicate)
- template<typename ForwardIterator, typename InputIterator, typename Predicate>
ForwardIterator thrust::remove_if(ForwardIterator, ForwardIterator, InputIterator,
→ Predicate)
- template<typename ForwardIterator, typename Predicate>
ForwardIterator thrust::remove_if(ForwardIterator, ForwardIterator, Predicate)
```

Template Function `thrust::remove_if(ForwardIterator, ForwardIterator, InputIterator, Predicate)`

Function Documentation

`template<typename ForwardIterator, typename InputIterator, typename Predicate>`

ForwardIterator `thrust::remove_if(ForwardIterator first, ForwardIterator last, InputIterator stencil, Predicate pred)`

`remove_if` removes from the range `[first, last)` every element `x` such that `pred(x)` is true. That is, `remove_if` returns an iterator `new_last` such that the range `[first, new_last)` contains no elements for which `pred` of the corresponding stencil value is true. The iterators in the range `[new_last, last)` are all still dereferenceable, but the elements that they point to are unspecified. `remove_if` is stable, meaning that the relative order of elements that are not removed is unchanged.

The following code snippet demonstrates how to use `remove_if` to remove specific elements from an array of integers.

Return A `ForwardIterator` pointing to the end of the resulting range of elements for which `pred` evaluated to true.

Pre The range `[first, last)` shall not overlap the range `[result, result + (last - first))`.

Pre The range `[stencil, stencil + (last - first))` shall not overlap the range `[result, result + (last - first))`.

Parameters

- `first`: The beginning of the range of interest.
- `last`: The end of the range of interest.
- `stencil`: The beginning of the stencil sequence.
- `pred`: A predicate to evaluate for each element of the range `[stencil, stencil + (last - first))`. Elements for which `pred` evaluates to `true` are removed from the sequence `[first, last)`.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#) and `ForwardIterator` is mutable.
- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/remove.h>
...
const int N = 6;
int A[N] = {1, 4, 2, 8, 5, 7};
int S[N] = {0, 1, 1, 1, 0, 0};

int *new_end = thrust::remove_if(A, A + N, S, thrust::identity<int>());
// The first three values of A are now {1, 5, 7}
// Values beyond new_end are unspecified
```

Note The range `[first, last)` is not permitted to overlap with the range `[stencil, stencil + (last - first))`.

See http://www.sgi.com/tech/stl/remove_if.html

See `remove`

See `remove_copy`

See `remove_copy_if`

Template Function `thrust::replace(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, const T&)`

- Defined in file `thrust_replace.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::replace`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, const T&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename T>__host__ __
  device__ void thrust::replace(const thrust::detail::execution_policy_base<
  DerivedPolicy > &, ForwardIterator, ForwardIterator, const T &, const T &)
- template<typename ForwardIterator, typename T>
  void thrust::replace(ForwardIterator, ForwardIterator, const T&, const T&)
```

Template Function `thrust::replace(ForwardIterator, ForwardIterator, const T&, const T&)`

Function Documentation

```
template<typename ForwardIterator, typename T>
```

```
void thrust::replace(ForwardIterator first, ForwardIterator last, const T &old_value, const T  
                    &new_value)
```

`replace` replaces every element in the range `[first, last)` equal to `old_value` with `new_value`. That is: for every iterator `i`, if `*i == old_value` then it performs the assignment `*i = new_value`.

The following code snippet demonstrates how to use `replace` to replace a value of interest in a `device_vector` with another.

Parameters

- `first`: The beginning of the sequence of interest.
- `last`: The end of the sequence of interest.
- `old_value`: The value to replace.
- `new_value`: The new value to replace `old_value`.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator` is mutable.
- `T`: is a model of [Assignable](#), `T` is a model of [EqualityComparable](#), objects of `T` may be compared for equality with objects of `ForwardIterator`'s `value_type`, and `T` is convertible to `ForwardIterator`'s `value_type`.

```
#include <thrust/replace.h>
#include <thrust/device_vector.h>

...

thrust::device_vector<int> A(4);
A[0] = 1;
A[1] = 2;
A[2] = 3;
A[3] = 1;

thrust::replace(A.begin(), A.end(), 1, 99);

// A contains [99, 2, 3, 99]
```

See <http://www.sgi.com/tech/stl/replace.html>

See `replace_if`

See `replace_copy`

See `replace_copy_if`

Template Function `thrust::replace_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, const T&, const T&)`

- Defined in file `thrust_replace.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::replace_copy`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `InputIterator`, `InputIterator`, `OutputIterator`, `const T&`, `const T&`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
→ typename T>__host__ __device__ OutputIterator thrust::replace_copy(const
→ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
→ InputIterator, OutputIterator, const T &, const T &)
- template<typename InputIterator, typename OutputIterator, typename T>
  OutputIterator thrust::replace_copy(InputIterator, InputIterator, OutputIterator,
→ const T&, const T&)
```

Template Function `thrust::replace_copy(InputIterator, InputIterator, OutputIterator, const T&, const T&)`

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**, typename **T**>

OutputIterator thrust::replace_copy(*InputIterator* first, *InputIterator* last, *OutputIterator* result, **const T** &old_value, **const T** &new_value)

`replace_copy` copies elements from the range `[first, last)` to the range `[result, result + (last-first))`, except that any element equal to `old_value` is not copied; `new_value` is copied instead.

More precisely, for every integer `n` such that `0 <= n < last-first`, `replace_copy` performs the assignment `*(result+n) = new_value` if `*(first+n) == old_value`, and `*(result+n) = *(first+n)` otherwise.

```
#include <thrust/replace.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> A(4);
A[0] = 1;
A[1] = 2;
A[2] = 3;
A[3] = 1;

thrust::device_vector<int> B(4);

thrust::replace_copy(A.begin(), A.end(), B.begin(), 1, 99);

// B contains [99, 2, 3, 99]
```

Return `result + (last-first)`

Pre first may equal result, but the ranges [first, last) and [result, result + (last - first)) shall not overlap otherwise.

Parameters

- first: The beginning of the sequence to copy from.
- last: The end of the sequence to copy from.
- result: The beginning of the sequence to copy to.
- old_value: The value to replace.
- new_value: The replacement value for which `*i == old_value` evaluates to true.

Template Parameters

- InputIterator: is a model of [Input Iterator](#).
- OutputIterator: is a model of [Output Iterator](#).
- T: is a model of [Assignable](#), T is a model of [Equality Comparable](#), T may be compared for equality with InputIterator's value_type, and T is convertible to OutputIterator's value_type.

See http://www.sgi.com/tech/stl/replace_copy.html

See `copy`

See `replace`

See `replace_if`

See `replace_copy_if`

Template Function `thrust::replace_copy_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, Predicate, const T&)`

- Defined in file `thrust_replace.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::replace_copy_if`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, Predicate, const T&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
→typename Predicate, typename T>__host__ __device__ OutputIterator thrust::replace_
→copy_if(const thrust::detail::execution_policy_base< DerivedPolicy > &,
→InputIterator, InputIterator, OutputIterator, Predicate, const T &)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
→ typename OutputIterator, typename Predicate, typename T>__host__ __device__
→OutputIterator thrust::replace_copy_if(const thrust::detail::execution_policy_base
→< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
→OutputIterator, Predicate, const T &)
- template<typename InputIterator, typename OutputIterator, typename Predicate,
→typename T>
  OutputIterator thrust::replace_copy_if(InputIterator, InputIterator,
→OutputIterator, Predicate, const T&)
```

```
- template<typename InputIterator1, typename InputIterator2, typename
→OutputIterator, typename Predicate, typename T>
  OutputIterator thrust::replace_copy_if(InputIterator1, InputIterator2, OutputIterator, Predicate, const T&)
```

Template Function `thrust::replace_copy_if(InputIterator, InputIterator, OutputIterator, Predicate, const T&)`

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**, typename **Predicate**, typename **T**>
OutputIterator thrust::replace_copy_if(*InputIterator first*, *InputIterator last*, *OutputIterator result*,

Predicate pred, **const T** &*new_value*)

replace_copy_if copies elements from the range [first, last) to the range [result, result + (last-first)), except that any element for which pred is true is not copied; new_value is copied instead.

More precisely, for every integer n such that 0 ≤ n < last-first, replace_copy_if performs the assignment *(result+n) = new_value if pred(*(first+n)), and *(result+n) = *(first+n) otherwise.

```
#include <thrust/replace.h>
#include <thrust/device_vector.h>

struct is_less_than_zero
{
    __host__ __device__
    bool operator()(int x)
    {
        return x < 0;
    }
};

...

thrust::device_vector<int> A(4);
A[0] = 1;
A[1] = -3;
A[2] = 2;
A[3] = -1;

thrust::device_vector<int> B(4);
is_less_than_zero pred;

thrust::replace_copy_if(A.begin(), A.end(), B.begin(), pred, 0);

// B contains [1, 0, 2, 0]
```

Return result + (last-first)

Pre first may equal result, but the ranges [first, last) and [result, result + (last - first)) shall not overlap otherwise.

Parameters

- first: The beginning of the sequence to copy from.
- last: The end of the sequence to copy from.
- result: The beginning of the sequence to copy to.

- `pred`: The predicate to test on every value of the range `[first, last)`.
- `new_value`: The replacement value to assign `pred(*i)` evaluates to `true`.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `OutputIterator`: is a model of [Output Iterator](#).
- `Predicate`: is a model of [Predicate](#).
- `T`: is a model of [Assignable](#), and `T` is convertible to `OutputIterator`'s `value_type`.

See http://www.sgi.com/tech/stl/replace_copy_if.html

See `replace`

See `replace_if`

See `replace_copy`

Template Function `thrust::replace_copy_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate, const T&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::replace_copy_if`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate, const T&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
→typename Predicate, typename T>__host__ __device__ OutputIterator thrust::replace_
→copy_if(const thrust::detail::execution_policy_base< DerivedPolicy > &,
→InputIterator, InputIterator, OutputIterator, Predicate, const T &)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
→typename OutputIterator, typename Predicate, typename T>__host__ __device__
→OutputIterator thrust::replace_copy_if(const thrust::detail::execution_policy_base
→< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
→OutputIterator, Predicate, const T &)
- template<typename InputIterator, typename OutputIterator, typename Predicate,
→typename T>
  OutputIterator thrust::replace_copy_if(InputIterator, InputIterator,
→OutputIterator, Predicate, const T&)
- template<typename InputIterator1, typename InputIterator2, typename
→OutputIterator, typename Predicate, typename T>
  OutputIterator thrust::replace_copy_if(InputIterator1, InputIterator1,
→InputIterator2, OutputIterator, Predicate, const T&)
```

Template Function `thrust::replace_copy_if(InputIterator1, InputIterator1, InputIterator2, OutputIterator, Predicate, const T&)`

This version of `replace_copy_if` copies elements from the range `[first, last)` to the range `[result, result + (last-first))`, except that any element whose corresponding stencil element causes `pred` to be `true` is not copied; `new_value` is copied instead.

```
#include <thrust/replace.h>
#include <thrust/device_vector.h>

struct is_less_than_zero
{
    __host__ __device__
    bool operator() (int x)
    {
        return x < 0;
    }
};

...

thrust::device_vector<int> A(4);
A[0] = 10;
A[1] = 20;
A[2] = 30;
A[3] = 40;

thrust::device_vector<int> S(4);
S[0] = -1;
S[1] = 0;
S[2] = -1;
S[3] = 0;

thrust::device_vector<int> B(4);
is_less_than_zero pred;

thrust::replace_if(A.begin(), A.end(), S.begin(), B.begin(), pred, 0);

// B contains [0, 20, 0, 40]
```

Pre first may equal result, but the ranges [first, last) and [result, result + (last - first)) shall not overlap otherwise.

Parameters

- 241

- last: The end of the sequence to copy from.
- stencil: The beginning of the stencil sequence.
- result: The beginning of the sequence to copy to.
- pred: The predicate to test on every value of the range [stencil, stencil + (last - first)).
- new_value: The replacement value to assign when pred(*s) evaluates to true.

Template Parameters

- InputIterator1: is a model of [Input Iterator](#).
- InputIterator2: is a model of [Input Iterator](#) and InputIterator2's value_type is convertible to Predicate's argument_type.
- OutputIterator: is a model of [Output Iterator](#).
- Predicate: is a model of [Predicate](#).
- T: is a model of [Assignable](#), and T is convertible to OutputIterator's value_type.

See `replace_copy`

See `replace_if`

Template Function `thrust::replace_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate, const T&)`

- Defined in file `thrust_replace.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::replace_if” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate, const T&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,  
→ typename Predicate, typename T>__host__ __device__ void thrust::replace_if(const_  
→thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,_  
→ForwardIterator, InputIterator, Predicate, const T &)  
- template<typename DerivedPolicy, typename ForwardIterator, typename Predicate,_  
→typename T>__host__ __device__ void thrust::replace_if(const_  
→thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,_  
→ForwardIterator, Predicate, const T &)  
- template<typename ForwardIterator, typename InputIterator, typename Predicate,_  
→typename T>  
  void thrust::replace_if(ForwardIterator, ForwardIterator, InputIterator,_  
→Predicate, const T&)  
- template<typename ForwardIterator, typename Predicate, typename T>  
  void thrust::replace_if(ForwardIterator, ForwardIterator, Predicate, const T&)
```

Template Function `thrust::replace_if(ForwardIterator, ForwardIterator, Predicate, const T&)`

Function Documentation

template<typename **ForwardIterator**, typename **Predicate**, typename **T**>

void thrust::replace_if(*ForwardIterator first*, *ForwardIterator last*, *Predicate pred*, const *T* &*new_value*)

`replace_if` replaces every element in the range `[first, last)` for which `pred` returns `true` with `new_value`. That is: for every iterator `i`, if `pred(*i)` is `true` then it performs the assignment `*i = new_value`.

The following code snippet demonstrates how to use `replace_if` to replace a `device_vector`'s negative elements with 0.

Parameters

- `first`: The beginning of the sequence of interest.
- `last`: The end of the sequence of interest.
- `pred`: The predicate to test on every value of the range `[first, last)`.
- `new_value`: The new value to replace elements which `pred(*i)` evaluates to `true`.

Template Parameters

- `ForwardIterator`: is a model of `Forward Iterator`, `ForwardIterator` is mutable, and `ForwardIterator`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `Predicate`: is a model of `Predicate`.
- `T`: is a model of `Assignable`, and `T` is convertible to `ForwardIterator`'s `value_type`.

```
#include <thrust/replace.h>
#include <thrust/device_vector.h>
...
struct is_less_than_zero
{
    __host__ __device__
    bool operator()(int x)
    {
        return x < 0;
    }
};
...

thrust::device_vector<int> A(4);
A[0] = 1;
A[1] = -3;
A[2] = 2;
A[3] = -1;

is_less_than_zero pred;

thrust::replace_if(A.begin(), A.end(), pred, 0);

// A contains [1, 0, 2, 0]
```

See http://www.sgi.com/tech/stl/replace_if.html

See `replace`

See `replace_copy`

See `replace_copy_if`

Template Function `thrust::replace_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, Predicate, const T&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::replace_if” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, Predicate, const T&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
→ typename Predicate, typename T>__host__ __device__ void thrust::replace_if(const_
→thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
→ForwardIterator, InputIterator, Predicate, const T &)
- template<typename DerivedPolicy, typename ForwardIterator, typename Predicate,
→typename T>__host__ __device__ void thrust::replace_if(const_
→thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
→ForwardIterator, Predicate, const T &)
- template<typename ForwardIterator, typename InputIterator, typename Predicate,
→typename T>
  void thrust::replace_if(ForwardIterator, ForwardIterator, InputIterator,
→Predicate, const T&)
- template<typename ForwardIterator, typename Predicate, typename T>
  void thrust::replace_if(ForwardIterator, ForwardIterator, Predicate, const T&)
```

Template Function `thrust::replace_if(ForwardIterator, ForwardIterator, InputIterator, Predicate, const T&)`

Function Documentation

```
template<typename ForwardIterator, typename InputIterator, typename Predicate, typename T>
void thrust::replace_if(ForwardIterator first, ForwardIterator last, InputIterator stencil, Predicate
pred, const T &new_value)
```

`replace_if` replaces every element in the range `[first, last)` for which `pred(*s)` returns true with `new_value`. That is: for every iterator `i` in the range `[first, last)`, and `s` in the range `[stencil, stencil + (last - first))`, if `pred(*s)` is true then it performs the assignment `*i = new_value`.

The following code snippet demonstrates how to use `replace_if` to replace a `device_vector`'s element with 0 when its corresponding stencil element is less than zero.

Parameters

- `first`: The beginning of the sequence of interest.
- `last`: The end of the sequence of interest.
- `stencil`: The beginning of the stencil sequence.
- `pred`: The predicate to test on every value of the range `[first, last)`.

- `new_value`: The new value to replace elements which `pred(*i)` evaluates to `true`.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator` is mutable.
- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `Predicate`: is a model of [Predicate](#).
- `T`: is a model of [Assignable](#), and `T` is convertible to `ForwardIterator`'s `value_type`.

```
#include <thrust/replace.h>
#include <thrust/device_vector.h>

struct is_less_than_zero
{
    __host__ __device__
    bool operator()(int x)
    {
        return x < 0;
    }
};

...

thrust::device_vector<int> A(4);
A[0] = 10;
A[1] = 20;
A[2] = 30;
A[3] = 40;

thrust::device_vector<int> S(4);
S[0] = -1;
S[1] = 0;
S[2] = -1;
S[3] = 0;

is_less_than_zero pred;
thrust::replace_if(A.begin(), A.end(), S.begin(), pred, 0);

// A contains [0, 20, 0, 40]
```

See http://www.sgi.com/tech/stl/replace_if.html

See `replace`

See `replace_copy`

See `replace_copy_if`

Template Function `thrust::return_temporary_buffer`

- Defined in `file_thrust_memory.h`

Function Documentation

template<typename DerivedPolicy, typename Pointer>__host__ __device__ void thrust::return_temporary_buffer deallocates storage associated with a given Thrust system previously allocated by `get_temporary_buffer`.

Thrust uses `return_temporary_buffer` internally when deallocating temporary storage required by algorithm implementations.

The following code snippet demonstrates how to use `return_temporary_buffer` to deallocate a range of memory previously allocated by `get_temporary_buffer`.

Pre `p` shall have been previously allocated by `thrust::get_temporary_buffer`.

Parameters

- `system`: The Thrust system with which the storage is associated.
- `p`: A pointer previously returned by `thrust::get_temporary_buffer`. If `ptr` is null, `return_temporary_buffer` does nothing.

Template Parameters

- `DerivedPolicy`: The name of the derived execution policy.

```
#include <thrust/memory.h>
...
// allocate storage for 100 ints with thrust::get_temporary_buffer
const int N = 100;

typedef thrust::pair<
    thrust::pointer<int, thrust::device_system_tag>,
    std::ptrdiff_t
> ptr_and_size_t;

thrust::device_system_tag device_sys;
ptr_and_size_t ptr_and_size = thrust::get_temporary_buffer<int>(device_sys, N);

// manipulate up to 100 ints
for(int i = 0; i < ptr_and_size.second; ++i)
{
    *ptr_and_size.first = i;
}

// deallocate storage with thrust::return_temporary_buffer
thrust::return_temporary_buffer(device_sys, ptr_and_size.first);
```

See *free*

See *get_temporary_buffer*

Template Function `thrust::reverse(const thrust::detail::execution_policy_base<DerivedPolicy>&, BidirectionalIterator, BidirectionalIterator)`

- Defined in file `_thrust_reverse.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::reverse” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, BidirectionalIterator, BidirectionalIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename BidirectionalIterator>
  void thrust::reverse(BidirectionalIterator, BidirectionalIterator)
- template<typename DerivedPolicy, typename BidirectionalIterator>__host__ __device__
  void thrust::reverse(const thrust::detail::execution_policy_base< DerivedPolicy_
  &, BidirectionalIterator, BidirectionalIterator)
```

Template Function thrust::reverse(BidirectionalIterator, BidirectionalIterator)

Function Documentation

template<typename **BidirectionalIterator**>

void thrust::reverse(*BidirectionalIterator first, BidirectionalIterator last*)

reverse reverses a range. That is: for every i such that $0 \leq i \leq (\text{last} - \text{first}) / 2$, it exchanges $*(\text{first} + i)$ and $*(\text{last} - (i + 1))$.

The following code snippet demonstrates how to use `reverse` to reverse a *device_vector* of integers.

Parameters

- first: The beginning of the range to reverse.
- last: The end of the range to reverse.

Template Parameters

- BidirectionalIterator: is a model of Bidirectional Iterator and BidirectionalIterator is mutable.

```
#include <thrust/reverse.h>
...
const int N = 6;
int data[N] = {0, 1, 2, 3, 4, 5};
thrust::device_vector<int> v(data, data + N);
thrust::reverse(v.begin(), v.end());
// v is now {5, 4, 3, 2, 1, 0}
```

See <http://www.sgi.com/tech/stl/reverse.html>

See `reverse_copy`

See `reverse_iterator`

Template Function thrust::reverse_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, BidirectionalIterator, BidirectionalIterator, OutputIterator)

- Defined in file_thrust_reverse.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::reverse_copy” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, BidirectionalIterator, BidirectionalIterator, OutputIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename BidirectionalIterator, typename OutputIterator>
  OutputIterator thrust::reverse_copy(BidirectionalIterator, BidirectionalIterator,
  ↪OutputIterator)
- template<typename DerivedPolicy, typename BidirectionalIterator, typename
  ↪OutputIterator>__host__ __device__ OutputIterator thrust::reverse_copy(const
  ↪thrust::detail::execution_policy_base< DerivedPolicy > &, BidirectionalIterator,
  ↪BidirectionalIterator, OutputIterator)
```

Template Function thrust::reverse_copy(BidirectionalIterator, BidirectionalIterator, OutputIterator)

Function Documentation

template<typename **BidirectionalIterator**, typename **OutputIterator**>

OutputIterator thrust::reverse_copy(*BidirectionalIterator* first, *BidirectionalIterator* last, *OutputIterator* result)

reverse_copy differs from reverse only in that the reversed range is written to a different output range, rather than inplace.

reverse_copy copies elements from the range [first, last) to the range [result, result + (last - first)) such that the copy is a reverse of the original range. Specifically: for every i such that 0 ≤ i < (last - first), reverse_copy performs the assignment *(result + (last - first) - i) = *(first + i).

The return value is result + (last - first).

The following code snippet demonstrates how to use reverse_copy to reverse an input *device_vector* of integers to an output *device_vector*.

Pre The range [first, last) and the range [result, result + (last - first)) shall not overlap.

Parameters

- first: The beginning of the range to reverse.
- last: The end of the range to reverse.
- result: The beginning of the output range.

Template Parameters

- BidirectionalIterator: is a model of Bidirectional Iterator, and BidirectionalIterator's value_type is convertible to OutputIterator's value_type.
- OutputIterator: is a model of Output Iterator.

```
#include <thrust/reverse.h>
...
const int N = 6;
int data[N] = {0, 1, 2, 3, 4, 5};
```

(continues on next page)

(continued from previous page)

```
thrust::device_vector<int> input(data, data + N);
thrust::device_vector<int> output(N);
thrust::reverse_copy(v.begin(), v.end(), output.begin());
// input is still {0, 1, 2, 3, 4, 5}
// output is now {5, 4, 3, 2, 1, 0}
```

See http://www.sgi.com/tech/stl/reverse_copy.html

See `reverse`

See `reverse_iterator`

Template Function `thrust::scatter(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator)`

- Defined in file `_thrust_scatter.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::scatter`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename RandomAccessIterator>__host__ __device__ void thrust::scatter(const_
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, RandomAccessIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ RandomAccessIterator>
  void thrust::scatter(InputIterator1, InputIterator1, InputIterator2,
  ↳ RandomAccessIterator)
```

Template Function `thrust::scatter(InputIterator1, InputIterator1, InputIterator2, RandomAccessIterator)`

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename RandomAccessIterator>
void thrust::scatter(InputIterator1 first, InputIterator1 last, InputIterator2 map, RandomAccessIter-
                    ator result)
```

`scatter` copies elements from a source range into an output array according to a map. For each iterator `i` in the range `[first, last)`, the value `*i` is assigned to `output[* (map + (i - first))]`. The output iterator must permit random access. If the same index appears more than once in the range `[map, map + (last - first))`, the result is undefined.

The following code snippet demonstrates how to use `scatter` to reorder a range.

Pre The iterator `result + i` shall not refer to any element referenced by any iterator `j` in the range `[first, last)` for all iterators `i` in the range `[map, map + (last - first))`.

Pre The iterator `result + i` shall not refer to any element referenced by any iterator `j` in the range `[map, map + (last - first))` for all iterators `i` in the range `[map, map + (last - first))`.

Pre The expression `result[*i]` shall be valid for all iterators in the range `[map, map + (last - first))`.

Parameters

- `first`: Beginning of the sequence of values to scatter.
- `last`: End of the sequence of values to scatter.
- `map`: Beginning of the sequence of output indices.
- `result`: Destination of the source elements.

Template Parameters

- `InputIterator1`: must be a model of [Input Iterator](#) and `InputIterator1`'s `value_type` must be convertible to `RandomAccessIterator`'s `value_type`.
- `InputIterator2`: must be a model of [Input Iterator](#) and `InputIterator2`'s `value_type` must be convertible to `RandomAccessIterator`'s `difference_type`.
- `RandomAccessIterator`: must be a model of [Random Access iterator](#).

```
#include <thrust/scatter.h>
#include <thrust/device_vector.h>
...
// mark even indices with a 1; odd indices with a 0
int values[10] = {1, 0, 1, 0, 1, 0, 1, 0, 1, 0};
thrust::device_vector<int> d_values(values, values + 10);

// scatter all even indices into the first half of the
// range, and odd indices vice versa
int map[10] = {0, 5, 1, 6, 2, 7, 3, 8, 4, 9};
thrust::device_vector<int> d_map(map, map + 10);

thrust::device_vector<int> d_output(10);
thrust::scatter(d_values.begin(), d_values.end(),
               d_map.begin(), d_output.begin());
// d_output is now {1, 1, 1, 1, 1, 0, 0, 0, 0, 0}
```

Note `scatter` is the inverse of `thrust::gather`.

Template Function `thrust::scatter_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator3, RandomAccessIterator)`

- Defined in `file_thrust_scatter.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::scatter_if`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator3, RandomAccessIterator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename InputIterator3, typename RandomAccessIterator, typename Predicate>__
  ↳ host__ __device__ void thrust::scatter_if(const thrust::detail::execution_policy_
  ↳ base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator3, RandomAccessIterator, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename InputIterator3, typename RandomAccessIterator>__host__ __device__ void
  ↳ thrust::scatter_if(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, InputIterator3,
  ↳ RandomAccessIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ InputIterator3, typename RandomAccessIterator, typename Predicate>
  void thrust::scatter_if(InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator3, RandomAccessIterator, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ InputIterator3, typename RandomAccessIterator>
  void thrust::scatter_if(InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator3, RandomAccessIterator)

```

Template Function `thrust::scatter_if(InputIterator1, InputIterator1, InputIterator2, InputIterator3, RandomAccessIterator)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **InputIterator3**, typename **RandomAccessIterator**>
 void thrust::scatter_if(*InputIterator1 first, InputIterator1 last, InputIterator2 map, InputIterator3 stencil, RandomAccessIterator output*)

`scatter_if` conditionally copies elements from a source range into an output array according to a map. For each iterator `i` in the range `[first, last)` such that `*(stencil + (i - first))` is true, the value `*i` is assigned to `output[* (map + (i - first))]`. The output iterator must permit random access. If the same index appears more than once in the range `[map, map + (last - first))` the result is undefined.

```

#include <thrust/scatter.h>
...
int V[8] = {10, 20, 30, 40, 50, 60, 70, 80};
int M[8] = {0, 5, 1, 6, 2, 7, 3, 4};
int S[8] = {1, 0, 1, 0, 1, 0, 1, 0};
int D[8] = {0, 0, 0, 0, 0, 0, 0, 0};

thrust::scatter_if(V, V + 8, M, S, D);

// D contains [10, 30, 50, 70, 0, 0, 0, 0];

```

Pre The iterator `result + i` shall not refer to any element referenced by any iterator `j` in the range `[first, last)` for all iterators `i` in the range `[map, map + (last - first))`.

Pre The iterator `result + i` shall not refer to any element referenced by any iterator `j` in the range `[map, map + (last - first))` for all iterators `i` in the range `[map, map + (last - first))`.

Pre The iterator `result + i` shall not refer to any element referenced by any iterator `j` in the range `[stencil, stencil + (last - first))` for all iterators `i` in the range `[map, map + (last - first))`.

Pre The expression `result[*i]` shall be valid for all iterators `i` in the range `[map, map + (last - first))` for which the following condition holds: `*(stencil + i) != false`.

Parameters

- `first`: Beginning of the sequence of values to scatter.
- `last`: End of the sequence of values to scatter.
- `map`: Beginning of the sequence of output indices.
- `stencil`: Beginning of the sequence of predicate values.
- `output`: Beginning of the destination range.

Template Parameters

- `InputIterator1`: must be a model of [Input Iterator](#) and `InputIterator1`'s `value_type` must be convertible to `RandomAccessIterator`'s `value_type`.
- `InputIterator2`: must be a model of [Input Iterator](#) and `InputIterator2`'s `value_type` must be convertible to `RandomAccessIterator`'s `difference_type`.
- `InputIterator3`: must be a model of [Input Iterator](#) and `InputIterator3`'s `value_type` must be convertible to `bool`.
- `RandomAccessIterator`: must be a model of [Random Access iterator](#).

Note `scatter_if` is the inverse of `thrust::gather_if`.

Template Function `thrust::scatter_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator3, RandomAccessIterator, Predicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::scatter_if`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator3, RandomAccessIterator, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename InputIterator3, typename RandomAccessIterator, typename Predicate>__
  ↳ host__ __device__ void thrust::scatter_if(const thrust::detail::execution_policy_
  ↳ base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator3, RandomAccessIterator, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename InputIterator3, typename RandomAccessIterator>__host__ __device__ void
  ↳ thrust::scatter_if(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, InputIterator3,
  ↳ RandomAccessIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ InputIterator3, typename RandomAccessIterator, typename Predicate>
  ↳ void thrust::scatter_if(InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator3, RandomAccessIterator, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ InputIterator3, typename RandomAccessIterator>
  ↳ void thrust::scatter_if(InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator3, RandomAccessIterator)
```


Template Function `thrust::scatter_if(InputIterator1, InputIterator1, InputIterator2, InputIterator3, RandomAccessIterator, Predicate)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **InputIterator3**, typename **RandomAccessIterator**, typename **Predicate**>
 void thrust::scatter_if(*InputIterator1 first, InputIterator1 last, InputIterator2 map, InputIterator3 stencil, RandomAccessIterator output, Predicate pred*)

`scatter_if` conditionally copies elements from a source range into an output array according to a map. For each iterator `i` in the range `[first, last)` such that `pred(*(stencil + (i - first)))` is true, the value `*i` is assigned to `output[*(map + (i - first))]`. The output iterator must permit random access. If the same index appears more than once in the range `[map, map + (last - first))` the result is undefined.

```
#include <thrust/scatter.h>

struct is_even
{
    __host__ __device__
    bool operator()(int x)
    {
        return (x % 2) == 0;
    }
};

...

int V[8] = {10, 20, 30, 40, 50, 60, 70, 80};
int M[8] = {0, 5, 1, 6, 2, 7, 3, 4};
int S[8] = {2, 1, 2, 1, 2, 1, 2, 1};
int D[8] = {0, 0, 0, 0, 0, 0, 0, 0};

is_even pred;
thrust::scatter_if(V, V + 8, M, S, D, pred);

// D contains [10, 30, 50, 70, 0, 0, 0, 0];
```

Pre The iterator `result + i` shall not refer to any element referenced by any iterator `j` in the range `[first, last)` for all iterators `i` in the range `[map, map + (last - first))`.

Pre The iterator `result + i` shall not refer to any element referenced by any iterator `j` in the range `[map, map + (last - first))` for all iterators `i` in the range `[map, map + (last - first))`.

Pre The iterator `result + i` shall not refer to any element referenced by any iterator `j` in the range `[stencil, stencil + (last - first))` for all iterators `i` in the range `[map, map + (last - first))`.

Pre The expression `result[*i]` shall be valid for all iterators `i` in the range `[map, map + (last - first))` for which the following condition holds: `pred(*(stencil + i)) != false`.

Parameters

- `first`: Beginning of the sequence of values to scatter.
- `last`: End of the sequence of values to scatter.
- `map`: Beginning of the sequence of output indices.
- `stencil`: Beginning of the sequence of predicate values.

- output: Beginning of the destination range.
- pred: Predicate to apply to the stencil values.

Template Parameters

- InputIterator1: must be a model of [Input Iterator](#) and InputIterator1's value_type must be convertible to RandomAccessIterator's value_type.
- InputIterator2: must be a model of [Input Iterator](#) and InputIterator2's value_type must be convertible to RandomAccessIterator's difference_type.
- InputIterator3: must be a model of [Input Iterator](#) and InputIterator3's value_type must be convertible to Predicate's argument_type.
- RandomAccessIterator: must be a model of [Random Access iterator](#).
- Predicate: must be a model of [Predicate](#).

Note scatter_if is the inverse of thrust::gather_if.

Template Function thrust::sequence(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)

- Defined in file_thrust_sequence.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::sequence” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename T>__host__ __  
  ↪device__ void thrust::sequence(const thrust::detail::execution_policy_base<_  
  ↪DerivedPolicy > &, ForwardIterator, ForwardIterator, T)  
- template<typename DerivedPolicy, typename ForwardIterator, typename T>__host__ __  
  ↪device__ void thrust::sequence(const thrust::detail::execution_policy_base<_  
  ↪DerivedPolicy > &, ForwardIterator, ForwardIterator, T, T)  
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__  
  ↪void thrust::sequence(const thrust::detail::execution_policy_base< DerivedPolicy >  
  ↪ &, ForwardIterator, ForwardIterator)  
- template<typename ForwardIterator, typename T>  
  void thrust::sequence(ForwardIterator, ForwardIterator, T)  
- template<typename ForwardIterator, typename T>  
  void thrust::sequence(ForwardIterator, ForwardIterator, T, T)  
- template<typename ForwardIterator>  
  void thrust::sequence(ForwardIterator, ForwardIterator)
```

Template Function thrust::sequence(ForwardIterator, ForwardIterator)

Function Documentation

template<typename **ForwardIterator**>

`void thrust::sequence(ForwardIterator first, ForwardIterator last)`
 sequence fills the range `[first, last)` with a sequence of numbers.

For each iterator `i` in the range `[first, last)`, this version of `sequence` performs the assignment `*i = (i - first)`.

The following code snippet demonstrates how to use `sequence` to fill a range with a sequence of numbers.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator` is mutable, and if `x` and `y` are objects of `ForwardIterator`'s `value_type`, then `x + y` is defined, and if `T` is `ForwardIterator`'s `value_type`, then `T(0)` is defined.

```
#include <thrust/sequence.h>
...
const int N = 10;
int A[N];
thrust::sequence(A, A + 10);
// A is now {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Note Unlike the similar C++ STL function `std::iota`, `sequence` offers no guarantee on order of execution.

See <http://www.sgi.com/tech/stl/iota.html>

Template Function `thrust::sequence(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, T)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::sequence`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, T)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename T>__host__ __
  ↪device__ void thrust::sequence(const thrust::detail::execution_policy_base<
  ↪DerivedPolicy > &, ForwardIterator, ForwardIterator, T)
- template<typename DerivedPolicy, typename ForwardIterator, typename T>__host__ __
  ↪device__ void thrust::sequence(const thrust::detail::execution_policy_base<
  ↪DerivedPolicy > &, ForwardIterator, ForwardIterator, T, T)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
  ↪void thrust::sequence(const thrust::detail::execution_policy_base< DerivedPolicy >
  ↪ &, ForwardIterator, ForwardIterator)
- template<typename ForwardIterator, typename T>
  void thrust::sequence(ForwardIterator, ForwardIterator, T)
- template<typename ForwardIterator, typename T>
  void thrust::sequence(ForwardIterator, ForwardIterator, T, T)
- template<typename ForwardIterator>
  void thrust::sequence(ForwardIterator, ForwardIterator)
```

Template Function `thrust::sequence(ForwardIterator, ForwardIterator, T)`

Function Documentation

template<typename **ForwardIterator**, typename **T**>

void **thrust::sequence** (*ForwardIterator first, ForwardIterator last, T init*)

sequence fills the range `[first, last)` with a sequence of numbers.

For each iterator `i` in the range `[first, last)`, this version of `sequence` performs the assignment `*i = init + (i - first)`.

The following code snippet demonstrates how to use `sequence` to fill a range with a sequence of numbers starting from the value 1.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.
- `init`: The first value of the sequence of numbers.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator` is mutable, and if `x` and `y` are objects of `ForwardIterator`'s `value_type`, then `x + y` is defined, and if `T` is `ForwardIterator`'s `value_type`, then `T(0)` is defined.
- `T`: is a model of [Assignable](#), and `T` is convertible to `ForwardIterator`'s `value_type`.

```
#include <thrust/sequence.h>
...
const int N = 10;
int A[N];
thrust::sequence(A, A + 10, 1);
// A is now {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Note Unlike the similar C++ STL function `std::iota`, `sequence` offers no guarantee on order of execution.

See <http://www.sgi.com/tech/stl/iota.html>

Template Function `thrust::sequence(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, T, T)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::sequence`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, T, T)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename ForwardIterator, typename T>__host__ __
  ↪device__ void thrust::sequence(const thrust::detail::execution_policy_base<
  ↪DerivedPolicy > &, ForwardIterator, ForwardIterator, T)
- template<typename DerivedPolicy, typename ForwardIterator, typename T>__host__ __
  ↪device__ void thrust::sequence(const thrust::detail::execution_policy_base<
  ↪DerivedPolicy > &, ForwardIterator, ForwardIterator, T, T)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
  ↪void thrust::sequence(const thrust::detail::execution_policy_base< DerivedPolicy >
  ↪ &, ForwardIterator, ForwardIterator)
- template<typename ForwardIterator, typename T>
  void thrust::sequence(ForwardIterator, ForwardIterator, T)
- template<typename ForwardIterator, typename T>
  void thrust::sequence(ForwardIterator, ForwardIterator, T, T)
- template<typename ForwardIterator>
  void thrust::sequence(ForwardIterator, ForwardIterator)

```

Template Function `thrust::sequence(ForwardIterator, ForwardIterator, T, T)`

Function Documentation

template<typename **ForwardIterator**, typename **T**>

void thrust::sequence(*ForwardIterator first, ForwardIterator last, T init, T step*)

sequence fills the range `[first, last)` with a sequence of numbers.

For each iterator `i` in the range `[first, last)`, this version of `sequence` performs the assignment `*i = init + step * (i - first)`.

The following code snippet demonstrates how to use `sequence` to fill a range with a sequence of numbers starting from the value 1 with a step size of 3.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.
- `init`: The first value of the sequence of numbers
- `step`: The difference between consecutive elements.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator` is mutable, and if `x` and `y` are objects of `ForwardIterator`'s `value_type`, then `x + y` is defined, and if `T` is `ForwardIterator`'s `value_type`, then `T(0)` is defined.
- `T`: is a model of [Assignable](#), and `T` is convertible to `ForwardIterator`'s `value_type`.

```

#include <thrust/sequence.h>
...
const int N = 10;
int A[N];
thrust::sequence(A, A + 10, 1, 3);
// A is now {1, 4, 7, 10, 13, 16, 19, 22, 25, 28}

```

Note Unlike the similar C++ STL function `std::iota`, `sequence` offers no guarantee on order of execution.

See <http://www.sgi.com/tech/stl/iota.html>

Template Function `thrust::set_difference(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`

- Defined in file `_thrust_set_operations.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::set_difference” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename StrictWeakCompare>__host__ __device__
  ↳ OutputIterator thrust::set_difference(const thrust::detail::execution_policy_base
  ↳ < DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::set_
  ↳ difference(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename StrictWeakCompare>
  ↳ OutputIterator thrust::set_difference(InputIterator1, InputIterator1,
  ↳ InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator>
  ↳ OutputIterator thrust::set_difference(InputIterator1, InputIterator1,
  ↳ InputIterator2, InputIterator2, OutputIterator)
```

Template Function `thrust::set_difference(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`

Function Documentation

`template<typename InputIterator1, typename InputIterator2, typename OutputIterator>`
`OutputIterator thrust::set_difference(InputIterator1 first1, InputIterator1 last1, InputIterator2`
`first2, InputIterator2 last2, OutputIterator result)`

`set_difference` constructs a sorted range that is the set difference of the sorted ranges `[first1, last1)` and `[first2, last2)`. The return value is the end of the output range.

In the simplest case, `set_difference` performs the “difference” operation from set theory: the output range contains a copy of every element that is contained in `[first1, last1)` and not contained in `[first2, last1)`. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if `[first1, last1)` contains `m` elements that are equivalent to each other and if `[first2, last2)` contains `n` elements that are equivalent to them, the last `max(m-n, 0)` elements from `[first1, last1)` range shall be copied to the output range.

This version of `set_difference` compares elements using `operator<`.

The following code snippet demonstrates how to use `set_difference` to compute the set difference of two sets of integers sorted in ascending order.

Return The end of the output range.

Pre The ranges `[first1, last1)` and `[first2, last2)` shall be sorted with respect to `operator<`.

Pre The resulting range shall not overlap with either input range.

Parameters

- `first1`: The beginning of the first input range.
- `last1`: The end of the first input range.
- `first2`: The beginning of the second input range.
- `last2`: The end of the second input range.
- `result`: The beginning of the output range.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `OutputIterator`: is a model of [Output Iterator](#).

```
#include <thrust/set_operations.h>
...
int A1[6] = {0, 1, 3, 4, 5, 6, 9};
int A2[5] = {1, 3, 5, 7, 9};

int result[3];

int *result_end = thrust::set_difference(A1, A1 + 6, A2, A2 + 5, result);
// result is now {0, 4, 6}
```

See http://www.sgi.com/tech/stl/set_difference.html

See `includes`

See `set_union`

See `set_intersection`

See `set_symmetric_difference`

See `sort`

See `is_sorted`

Template Function `thrust::set_difference(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::set_difference” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename StrictWeakCompare>__host__ __device__
  ↳ OutputIterator thrust::set_difference(const thrust::detail::execution_policy_base
  ↳ < DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::set_
  ↳ difference(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename StrictWeakCompare>
  ↳ OutputIterator thrust::set_difference(InputIterator1, InputIterator1,
  ↳ InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator>
  ↳ OutputIterator thrust::set_difference(InputIterator1, InputIterator1,
  ↳ InputIterator2, InputIterator2, OutputIterator)
```

Template Function `thrust::set_difference(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**, typename **StrictWeakCompare**>
OutputIterator thrust::set_difference(*InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2, OutputIterator result, StrictWeakCompare comp*)

`set_difference` constructs a sorted range that is the set difference of the sorted ranges `[first1, last1)` and `[first2, last2)`. The return value is the end of the output range.

In the simplest case, `set_difference` performs the “difference” operation from set theory: the output range contains a copy of every element that is contained in `[first1, last1)` and not contained in `[first2, last1)`. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if `[first1, last1)` contains `m` elements that are equivalent to each other and if `[first2, last2)` contains `n` elements that are equivalent to them, the last `max(m-n, 0)` elements from `[first1, last1)` range shall be copied to the output range.

This version of `set_difference` compares elements using a function object `comp`.

The following code snippet demonstrates how to use `set_difference` to compute the set difference of two sets of integers sorted in descending order.

Return The end of the output range.

Pre The ranges `[first1, last1)` and `[first2, last2)` shall be sorted with respect to `comp`.

Pre The resulting range shall not overlap with either input range.

Parameters

- first1: The beginning of the first input range.
- last1: The end of the first input range.
- first2: The beginning of the second input range.
- last2: The end of the second input range.
- result: The beginning of the output range.
- comp: Comparison operator.

Template Parameters

- InputIterator1: is a model of [Input Iterator](#), InputIterator1's value_type is convertible to StrictWeakCompare's first_argument_type. and InputIterator1's value_type is convertible to a type in OutputIterator's set of value_types.
- InputIterator2: is a model of [Input Iterator](#), InputIterator2's value_type is convertible to StrictWeakCompare's second_argument_type. and InputIterator2's value_type is convertible to a type in OutputIterator's set of value_types.
- OutputIterator: is a model of [Output Iterator](#).
- StrictWeakCompare: is a model of [Strict Weak Ordering](#).

```
#include <thrust/set_operations.h>
#include <thrust/functional.h>
...
int A1[6] = {9, 6, 5, 4, 3, 1, 0};
int A2[5] = {9, 7, 5, 3, 1};

int result[3];

int *result_end = thrust::set_difference(A1, A1 + 6, A2, A2 + 5, result,
    thrust::greater<int>());
// result is now {6, 4, 0}
```

See http://www.sgi.com/tech/stl/set_difference.html

See includes

See set_union

See set_intersection

See set_symmetric_difference

See sort

See is_sorted

Template Function `thrust::set_difference_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)`

- Defined in file_thrust_set_operations.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::set_difference_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
    ↪ typename OutputIterator2, typename StrictWeakCompare>__host__ __device__
    ↪ thrust::pair<OutputIterator1, OutputIterator2> thrust::set_difference_by_key(const
    ↪ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
    ↪ InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4,
    ↪ OutputIterator1, OutputIterator2, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
    ↪ typename OutputIterator2>__host__ __device__ thrust::pair<OutputIterator1,
    ↪ OutputIterator2> thrust::set_difference_by_key(const thrust::detail::execution_
    ↪ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
    ↪ InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ InputIterator3, typename InputIterator4, typename OutputIterator1, typename
    ↪ OutputIterator2, typename StrictWeakCompare>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::set_difference_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪ InputIterator3, InputIterator4, OutputIterator1, OutputIterator2,
    ↪ StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ InputIterator3, typename InputIterator4, typename OutputIterator1, typename
    ↪ OutputIterator2>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::set_difference_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪ InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)
```

Template Function `thrust::set_difference_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **InputIterator3**, typename **InputIterator4**, typename **OutputIterator1**, typename **OutputIterator2**>

```

thrust::pair<OutputIterator1, OutputIterator2> thrust::set_difference_by_key (InputIterator1
                                                                    keys_first1,    In-
                                                                    putIterator1
                                                                    keys_last1,    In-
                                                                    putIterator2
                                                                    keys_first2,    In-
                                                                    putIterator2
                                                                    keys_last2,    In-
                                                                    putIterator3
                                                                    values_first1,
                                                                    InputIterator4
                                                                    values_first2,
                                                                    OutputItera-
                                                                    tor1 keys_result,
                                                                    OutputIterator2
                                                                    values_result)

```

`set_difference_by_key` performs a key-value difference operation from set theory. `set_difference_by_key` constructs a sorted range that is the difference of the sorted ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)`. Associated with each element from the input and output key ranges is a value element. The associated input value ranges need not be sorted.

In the simplest case, `set_difference_by_key` performs the “difference” operation from set theory: the keys output range contains a copy of every element that is contained in `[keys_first1, keys_last1)` and not contained in `[keys_first2, keys_last2)`. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if `[keys_first1, keys_last1)` contains m elements that are equivalent to each other and if `[keys_first2, keys_last2)` contains n elements that are equivalent to them, the last $\max(m-n, 0)$ elements from `[keys_first1, keys_last1)` range shall be copied to the output range.

Each time a key element is copied from `[keys_first1, keys_last1)` or `[keys_first2, keys_last2)` is copied to the keys output range, the corresponding value element is copied from the corresponding values input range (beginning at `values_first1` or `values_first2`) to the values output range.

This version of `set_difference_by_key` compares key elements using `operator<`.

The following code snippet demonstrates how to use `set_difference_by_key` to compute the set difference of two sets of integers sorted in ascending order with their values.

Return A pair `p` such that `p.first` is the end of the output range of keys, and such that `p.second` is the end of the output range of values.

Pre The ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)` shall be sorted with respect to `operator<`.

Pre The resulting ranges shall not overlap with any input range.

Parameters

- `keys_first1`: The beginning of the first input range of keys.
- `keys_last1`: The end of the first input range of keys.
- `keys_first2`: The beginning of the second input range of keys.
- `keys_last2`: The end of the second input range of keys.
- `values_first1`: The beginning of the first input range of values.
- `values_first2`: The beginning of the first input range of values.

- `keys_result`: The beginning of the output range of keys.
- `values_result`: The beginning of the output range of values.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator3`: is a model of [Input Iterator](#), and `InputIterator3`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `InputIterator4`: is a model of [Input Iterator](#), and `InputIterator4`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `OutputIterator1`: is a model of [Output Iterator](#).
- `OutputIterator2`: is a model of [Output Iterator](#).

```
#include <thrust/set_operations.h>
...
int A_keys[6] = {0, 1, 3, 4, 5, 6, 9};
int A_vals[6] = {0, 0, 0, 0, 0, 0, 0};

int B_keys[5] = {1, 3, 5, 7, 9};
int B_vals[5] = {1, 1, 1, 1, 1};

int keys_result[3];
int vals_result[3];

thrust::pair<int*,int*> end = thrust::set_difference_by_key(A_keys, A_keys + 6, B_
↳keys, B_keys + 5, A_vals, B_vals, keys_result, vals_result);
// keys_result is now {0, 4, 6}
// vals_result is now {0, 0, 0}
```

See `set_union_by_key`

See `set_intersection_by_key`

See `set_symmetric_difference_by_key`

See `sort_by_key`

See `is_sorted`

Template Function `thrust::set_difference_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::set_difference_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
    ↪ typename OutputIterator2, typename StrictWeakCompare>__host__ __device__
    ↪ thrust::pair<OutputIterator1,OutputIterator2> thrust::set_difference_by_key(const
    ↪ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
    ↪ InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4,
    ↪ OutputIterator1, OutputIterator2, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
    ↪ typename OutputIterator2>__host__ __device__ thrust::pair<OutputIterator1,
    ↪ OutputIterator2> thrust::set_difference_by_key(const thrust::detail::execution_
    ↪ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
    ↪ InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ InputIterator3, typename InputIterator4, typename OutputIterator1, typename
    ↪ OutputIterator2, typename StrictWeakCompare>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::set_difference_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪ InputIterator3, InputIterator4, OutputIterator1, OutputIterator2,
    ↪ StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ InputIterator3, typename InputIterator4, typename OutputIterator1, typename
    ↪ OutputIterator2>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::set_difference_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪ InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)
```

Template Function `thrust::set_difference_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **InputIterator3**, typename **InputIterator4**, typename **OutputIterator1**, typename **OutputIterator2**, typename **StrictWeakCompare**>

```
thrust::pair<OutputIterator1, OutputIterator2> thrust::set_difference_by_key (InputIterator1
                                                                    keys_first1,    In-
                                                                    putIterator1
                                                                    keys_last1,    In-
                                                                    putIterator2
                                                                    keys_first2,    In-
                                                                    putIterator2
                                                                    keys_last2,    In-
                                                                    putIterator3
                                                                    values_first1,
                                                                    InputIterator4
                                                                    values_first2,
                                                                    OutputItera-
                                                                    tor1 keys_result,
                                                                    OutputIterator2
                                                                    values_result,
                                                                    StrictWeakCom-
                                                                    pare comp)
```

`set_difference_by_key` performs a key-value difference operation from set theory. `set_difference_by_key` constructs a sorted range that is the difference of the sorted ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)`. Associated with each element from the input and output key ranges is a value element. The associated input value ranges need not be sorted.

In the simplest case, `set_difference_by_key` performs the “difference” operation from set theory: the keys output range contains a copy of every element that is contained in `[keys_first1, keys_last1)` and not contained in `[keys_first2, keys_last2)`. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if `[keys_first1, keys_last1)` contains m elements that are equivalent to each other and if `[keys_first2, keys_last2)` contains n elements that are equivalent to them, the last $\max(m-n, 0)$ elements from `[keys_first1, keys_last1)` range shall be copied to the output range.

Each time a key element is copied from `[keys_first1, keys_last1)` or `[keys_first2, keys_last2)` is copied to the keys output range, the corresponding value element is copied from the corresponding values input range (beginning at `values_first1` or `values_first2`) to the values output range.

This version of `set_difference_by_key` compares key elements using a function object `comp`.

The following code snippet demonstrates how to use `set_difference_by_key` to compute the set difference of two sets of integers sorted in descending order with their values.

Return A pair `p` such that `p.first` is the end of the output range of keys, and such that `p.second` is the end of the output range of values.

Pre The ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)` shall be sorted with respect to `comp`.

Pre The resulting ranges shall not overlap with any input range.

Parameters

- `keys_first1`: The beginning of the first input range of keys.
- `keys_last1`: The end of the first input range of keys.
- `keys_first2`: The beginning of the second input range of keys.
- `keys_last2`: The end of the second input range of keys.
- `values_first1`: The beginning of the first input range of values.

- `values_first2`: The beginning of the first input range of values.
- `keys_result`: The beginning of the output range of keys.
- `values_result`: The beginning of the output range of values.
- `comp`: Comparison operator.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator3`: is a model of [Input Iterator](#), and `InputIterator3`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `InputIterator4`: is a model of [Input Iterator](#), and `InputIterator4`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `OutputIterator1`: is a model of [Output Iterator](#).
- `OutputIterator2`: is a model of [Output Iterator](#).
- `StrictWeakCompare`: is a model of [Strict Weak Ordering](#).

```
#include <thrust/set_operations.h>
#include <thrust/functional.h>
...
int A_keys[6] = {9, 6, 5, 4, 3, 1, 0};
int A_vals[6] = {0, 0, 0, 0, 0, 0, 0};

int B_keys[5] = {9, 7, 5, 3, 1};
int B_vals[5] = {1, 1, 1, 1, 1};

int keys_result[3];
int vals_result[3];

thrust::pair<int*,int*> end = thrust::set_difference_by_key(A_keys, A_keys + 6, B_
↳keys, B_keys + 5, A_vals, B_vals, keys_result, vals_result, thrust::greater<int>
↳());
// keys_result is now {0, 4, 6}
// vals_result is now {0, 0, 0}
```

See `set_union_by_key`

See `set_intersection_by_key`

See `set_symmetric_difference_by_key`

See `sort_by_key`

See `is_sorted`

Template Function `thrust::set_intersection(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`

- Defined in `file_thrust_set_operations.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::set_intersection`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `InputIterator1`, `InputIterator1`, `InputIterator2`, `InputIterator2`, `OutputIterator`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename StrictWeakCompare>__host__ __device__
  ↳ OutputIterator thrust::set_intersection(const thrust::detail::execution_policy_
  ↳ base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::set_
  ↳ intersection(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename StrictWeakCompare>
  ↳ OutputIterator thrust::set_intersection(InputIterator1, InputIterator1,
  ↳ InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator>
  ↳ OutputIterator thrust::set_intersection(InputIterator1, InputIterator1,
  ↳ InputIterator2, InputIterator2, OutputIterator)
```

Template Function `thrust::set_intersection(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename OutputIterator>
OutputIterator thrust::set_intersection (InputIterator1 first1, InputIterator1 last1, InputIterator2
                                       first2, InputIterator2 last2, OutputIterator result)
```

`set_intersection` constructs a sorted range that is the intersection of sorted ranges `[first1, last1)` and `[first2, last2)`. The return value is the end of the output range.

In the simplest case, `set_intersection` performs the “intersection” operation from set theory: the output range contains a copy of every element that is contained in both `[first1, last1)` and `[first2, last2)`. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if a value appears m times in `[first1, last1)` and n times in `[first2, last2)` (where m may be zero), then it appears $\min(m, n)$ times in the output range. `set_intersection` is stable, meaning that both elements are copied from the first range rather than the second, and that the relative order of elements in the output range is the same as in the first input range.

This version of `set_intersection` compares objects using `operator<`.

The following code snippet demonstrates how to use `set_intersection` to compute the set intersection of two sets of integers sorted in ascending order.

Return The end of the output range.

Pre The ranges `[first1, last1)` and `[first2, last2)` shall be sorted with respect to `operator<`.

Pre The resulting range shall not overlap with either input range.

Parameters

- `first1`: The beginning of the first input range.
- `last1`: The end of the first input range.
- `first2`: The beginning of the second input range.
- `last2`: The end of the second input range.
- `result`: The beginning of the output range.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `OutputIterator`: is a model of [Output Iterator](#).

```
#include <thrust/set_operations.h>
...
int A1[6] = {1, 3, 5, 7, 9, 11};
int A2[7] = {1, 1, 2, 3, 5, 8, 13};

int result[7];

int *result_end = thrust::set_intersection(A1, A1 + 6, A2, A2 + 7, result);
// result is now {1, 3, 5}
```

See http://www.sgi.com/tech/stl/set_intersection.html

See `includes`

See `set_union`

See `set_intersection`

See `set_symmetric_difference`

See `sort`

See `is_sorted`

Template Function `thrust::set_intersection(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::set_intersection” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename StrictWeakCompare>__host__ __device__
  ↳OutputIterator thrust::set_intersection(const thrust::detail::execution_policy_
  ↳base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::set_
  ↳intersection(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳OutputIterator, typename StrictWeakCompare>
  ↳OutputIterator thrust::set_intersection(InputIterator1, InputIterator1,
  ↳InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳OutputIterator>
  ↳OutputIterator thrust::set_intersection(InputIterator1, InputIterator1,
  ↳InputIterator2, InputIterator2, OutputIterator)
```

Template Function thrust::set_intersection(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename OutputIterator, typename StrictWeakCompare>
OutputIterator thrust::set_intersection(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2,
InputIterator2 last2, OutputIterator result,
StrictWeakCompare comp)
```

set_intersection constructs a sorted range that is the intersection of sorted ranges [first1, last1) and [first2, last2). The return value is the end of the output range.

In the simplest case, set_intersection performs the “intersection” operation from set theory: the output range contains a copy of every element that is contained in both [first1, last1) and [first2, last2). The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if a value appears m times in [first1, last1) and n times in [first2, last2) (where m may be zero), then it appears $\min(m, n)$ times in the output range. set_intersection is stable, meaning that both elements are copied from the first range rather than the second, and that the relative order of elements in the output range is the same as in the first input range.

This version of set_intersection compares elements using a function object comp.

The following code snippet demonstrates how to use set_intersection to compute the set intersection of sets of integers sorted in descending order.

Return The end of the output range.

Pre The ranges [first1, last1) and [first2, last2) shall be sorted with respect to comp.

Pre The resulting range shall not overlap with either input range.

Parameters

- first1: The beginning of the first input range.
- last1: The end of the first input range.
- first2: The beginning of the second input range.
- last2: The end of the second input range.
- result: The beginning of the output range.
- comp: Comparison operator.

Template Parameters

- InputIterator1: is a model of [Input Iterator](#), InputIterator1 and InputIterator2 have the same value_type, InputIterator1's value_type is a model of [LessThan Comparable](#), the ordering on InputIterator1's value_type is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and InputIterator1's value_type is convertible to a type in OutputIterator's set of value_types.
- InputIterator2: is a model of [Input Iterator](#), InputIterator2 and InputIterator1 have the same value_type, InputIterator2's value_type is a model of [LessThan Comparable](#), the ordering on InputIterator2's value_type is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and InputIterator2's value_type is convertible to a type in OutputIterator's set of value_types.
- OutputIterator: is a model of [Output Iterator](#).

```
#include <thrust/set_operations.h>
...
int A1[6] = {11, 9, 7, 5, 3, 1};
int A2[7] = {13, 8, 5, 3, 2, 1, 1};

int result[3];

int *result_end = thrust::set_intersection(A1, A1 + 6, A2, A2 + 7, result,
    thrust::greater<int>());
// result is now {5, 3, 1}
```

See http://www.sgi.com/tech/stl/set_intersection.html

See `includes`

See `set_union`

See `set_intersection`

See `set_symmetric_difference`

See `sort`

See `is_sorted`

Template Function `thrust::set_intersection_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, OutputIterator1, OutputIterator2)`

- Defined in `file_thrust_set_operations.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::set_intersection_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, OutputIterator1, OutputIterator2) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename OutputIterator1, typename OutputIterator2,
    ↪ typename StrictWeakCompare>__host__ __device__ thrust::pair<OutputIterator1,
    ↪ OutputIterator2> thrust::set_intersection_by_key(const thrust::detail::execution_
    ↪ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
    ↪ InputIterator2, InputIterator3, OutputIterator1, OutputIterator2,
    ↪ StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename OutputIterator1, typename OutputIterator2>__
    ↪ host__ __device__ thrust::pair<OutputIterator1,OutputIterator2> thrust::set_
    ↪ intersection_by_key(const thrust::detail::execution_policy_base< DerivedPolicy > &
    ↪ , InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3,
    ↪ OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ InputIterator3, typename OutputIterator1, typename OutputIterator2, typename
    ↪ StrictWeakCompare>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::set_intersection_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪ InputIterator3, OutputIterator1, OutputIterator2, StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ InputIterator3, typename OutputIterator1, typename OutputIterator2>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::set_intersection_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪ InputIterator3, OutputIterator1, OutputIterator2)
```

Template Function `thrust::set_intersection_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, OutputIterator1, OutputIterator2)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **InputIterator3**, typename **OutputIterat**

```

thrust::pair<OutputIterator1, OutputIterator2> thrust::set_intersection_by_key(
    InputIterator1
    keys_first1,
    InputIterator1
    keys_last1,
    InputIterator2
    keys_first2,
    InputIterator2
    keys_last2,
    InputIterator3
    values_first1,
    OutputIt-
    erator1
    keys_result,
    OutputIt-
    ator2      val-
    ues_result)

```

`set_intersection_by_key` performs a key-value intersection operation from set theory. `set_intersection_by_key` constructs a sorted range that is the intersection of the sorted ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)`. Associated with each element from the input and output key ranges is a value element. The associated input value ranges need not be sorted.

In the simplest case, `set_intersection_by_key` performs the “intersection” operation from set theory: the keys output range contains a copy of every element that is contained in both `[keys_first1, keys_last1)` `[keys_first2, keys_last2)`. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if an element appears m times in `[keys_first1, keys_last1)` and n times in `[keys_first2, keys_last2)` (where m may be zero), then it appears $\min(m, n)$ times in the keys output range. `set_intersection_by_key` is stable, meaning both that elements are copied from the first input range rather than the second, and that the relative order of elements in the output range is the same as the first input range.

Each time a key element is copied from `[keys_first1, keys_last1)` to the keys output range, the corresponding value element is copied from `[values_first1, values_last1)` to the values output range.

This version of `set_intersection_by_key` compares objects using `operator<`.

The following code snippet demonstrates how to use `set_intersection_by_key` to compute the set intersection of two sets of integers sorted in ascending order with their values.

Return A pair `p` such that `p.first` is the end of the output range of keys, and such that `p.second` is the end of the output range of values.

Note Unlike the other key-value set operations, `set_intersection_by_key` is unique in that it has no `values_first2` parameter because elements from the second input range are never copied to the output range.

Pre The ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)` shall be sorted with respect to `operator<`.

Pre The resulting ranges shall not overlap with any input range.

Parameters

- `keys_first1`: The beginning of the first input range of keys.
- `keys_last1`: The end of the first input range of keys.
- `keys_first2`: The beginning of the second input range of keys.
- `keys_last2`: The end of the second input range of keys.

- `values_first1`: The beginning of the first input range of values.
- `keys_result`: The beginning of the output range of keys.
- `values_result`: The beginning of the output range of values.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator3`: is a model of [Input Iterator](#), and `InputIterator3`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `OutputIterator1`: is a model of [Output Iterator](#).
- `OutputIterator2`: is a model of [Output Iterator](#).

```
#include <thrust/set_operations.h>
...
int A_keys[6] = {1, 3, 5, 7, 9, 11};
int A_vals[6] = {0, 0, 0, 0, 0, 0};

int B_keys[7] = {1, 1, 2, 3, 5, 8, 13};

int keys_result[7];
int vals_result[7];

thrust::pair<int*, int*> end = thrust::set_intersection_by_key(A_keys, A_keys + 6,
↳B_keys, B_keys + 7, A_vals, keys_result, vals_result);

// keys_result is now {1, 3, 5}
// vals_result is now {0, 0, 0}
```

See `set_union_by_key`

See `set_difference_by_key`

See `set_symmetric_difference_by_key`

See `sort_by_key`

See `is_sorted`

Template Function `thrust::set_intersection_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, OutputIterator1, OutputIterator2, StrictWeakCompare)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::set_intersection_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, OutputIterator1, OutputIterator2, StrictWeakCompare) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename OutputIterator1, typename OutputIterator2,
    ↪ typename StrictWeakCompare>__host__ __device__ thrust::pair<OutputIterator1,
    ↪ OutputIterator2> thrust::set_intersection_by_key(const thrust::detail::execution_
    ↪ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
    ↪ InputIterator2, InputIterator3, OutputIterator1, OutputIterator2,
    ↪ StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename OutputIterator1, typename OutputIterator2>__
    ↪ host__ __device__ thrust::pair<OutputIterator1,OutputIterator2> thrust::set_
    ↪ intersection_by_key(const thrust::detail::execution_policy_base< DerivedPolicy > &
    ↪ , InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3,
    ↪ OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ InputIterator3, typename OutputIterator1, typename OutputIterator2, typename
    ↪ StrictWeakCompare>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::set_intersection_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪ InputIterator3, OutputIterator1, OutputIterator2, StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ InputIterator3, typename OutputIterator1, typename OutputIterator2>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::set_intersection_by_
    ↪ key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪ InputIterator3, OutputIterator1, OutputIterator2)
```

Template Function `thrust::set_intersection_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, OutputIterator1, OutputIterator2, StrictWeakCompare)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **InputIterator3**, typename **OutputIterat**

```
thrust::pair<OutputIterator1, OutputIterator2> thrust::set_intersection_by_key (InputIterator1
                                                                    keys_first1,
                                                                    InputIterator1
                                                                    keys_last1,
                                                                    InputIterator2
                                                                    keys_first2,
                                                                    InputIterator2
                                                                    keys_last2,
                                                                    InputIterator3
                                                                    values_first1,
                                                                    OutputIt-
                                                                    erator1
                                                                    keys_result,
                                                                    OutputIt-
                                                                    ator2      val-
                                                                    ues_result,
                                                                    StrictWeak-
                                                                    Compare
                                                                    comp)
```

`set_intersection_by_key` performs a key-value intersection operation from set theory. `set_intersection_by_key` constructs a sorted range that is the intersection of the sorted ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)`. Associated with each element from the input and output key ranges is a value element. The associated input value ranges need not be sorted.

In the simplest case, `set_intersection_by_key` performs the “intersection” operation from set theory: the keys output range contains a copy of every element that is contained in both `[keys_first1, keys_last1)` `[keys_first2, keys_last2)`. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if an element appears `m` times in `[keys_first1, keys_last1)` and `n` times in `[keys_first2, keys_last2)` (where `m` may be zero), then it appears `min(m, n)` times in the keys output range. `set_intersection_by_key` is stable, meaning both that elements are copied from the first input range rather than the second, and that the relative order of elements in the output range is the same as the first input range.

Each time a key element is copied from `[keys_first1, keys_last1)` to the keys output range, the corresponding value element is copied from `[values_first1, values_last1)` to the values output range.

This version of `set_intersection_by_key` compares objects using a function object `comp`.

The following code snippet demonstrates how to use `set_intersection_by_key` to compute the set intersection of two sets of integers sorted in descending order with their values.

Return A pair `p` such that `p.first` is the end of the output range of keys, and such that `p.second` is the end of the output range of values.

Note Unlike the other key-value set operations, `set_intersection_by_key` is unique in that it has no `values_first2` parameter because elements from the second input range are never copied to the output range.

Pre The ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)` shall be sorted with respect to `comp`.

Pre The resulting ranges shall not overlap with any input range.

Parameters

- `keys_first1`: The beginning of the first input range of keys.
- `keys_last1`: The end of the first input range of keys.

- `keys_first2`: The beginning of the second input range of keys.
- `keys_last2`: The end of the second input range of keys.
- `values_first1`: The beginning of the first input range of values.
- `keys_result`: The beginning of the output range of keys.
- `values_result`: The beginning of the output range of values.
- `comp`: Comparison operator.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator3`: is a model of [Input Iterator](#), and `InputIterator3`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `OutputIterator1`: is a model of [Output Iterator](#).
- `OutputIterator2`: is a model of [Output Iterator](#).
- `StrictWeakCompare`: is a model of [Strict Weak Ordering](#).

```
#include <thrust/set_operations.h>
#include <thrust/functional.h>
...
int A_keys[6] = {11, 9, 7, 5, 3, 1};
int A_vals[6] = { 0, 0, 0, 0, 0, 0};

int B_keys[7] = {13, 8, 5, 3, 2, 1, 1};

int keys_result[7];
int vals_result[7];

thrust::pair<int*,int*> end = thrust::set_intersection_by_key(A_keys, A_keys + 6,
↳B_keys, B_keys + 7, A_vals, keys_result, vals_result, thrust::greater<int>());

// keys_result is now {5, 3, 1}
// vals_result is now {0, 0, 0}
```

See `set_union_by_key`

See `set_difference_by_key`

See `set_symmetric_difference_by_key`

See `sort_by_key`

See `is_sorted`

Template Function `thrust::set_symmetric_difference(const thrust::detail::execution_policy_base<DerivedPolicy>& InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`

- Defined in `file_thrust_set_operations.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::set_symmetric_difference`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename StrictWeakCompare>__host__ __device__
  ↳ OutputIterator thrust::set_symmetric_difference(const thrust::detail::execution_
  ↳ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::set_symmetric_
  ↳ difference(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename StrictWeakCompare>
  ↳ OutputIterator thrust::set_symmetric_difference(InputIterator1, InputIterator1,
  ↳ InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator>
  ↳ OutputIterator thrust::set_symmetric_difference(InputIterator1, InputIterator1,
  ↳ InputIterator2, InputIterator2, OutputIterator)
```

Template Function `thrust::set_symmetric_difference(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename OutputIterator>
OutputIterator thrust::set_symmetric_difference (InputIterator1 first1, InputIterator1 last1, In-
                                              putIterator2 first2, InputIterator2 last2, Out-
                                              putIterator result)
```

`set_symmetric_difference` constructs a sorted range that is the set symmetric difference of the sorted ranges `[first1, last1)` and `[first2, last2)`. The return value is the end of the output range.

In the simplest case, `set_symmetric_difference` performs a set theoretic calculation: it constructs the union of the two sets $A - B$ and $B - A$, where A and B are the two input ranges. That is, the output range contains a copy of every element that is contained in `[first1, last1)` but not `[first2, last1)`, and a copy of every element that is contained in `[first2, last2)` but not `[first1, last1)`. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if `[first1, last1)` contains m elements that are equivalent to each other and `[first2, last1)` contains n elements that are equivalent to them, then $|m - n|$ of those elements shall be copied to the output range: the last $m - n$ elements from `[first1, last1)` if $m > n$, and the last $n - m$ of these elements from `[first2, last2)` if $m < n$.

This version of `set_union` compares elements using `operator<`.

The following code snippet demonstrates how to use `set_symmetric_difference` to compute the symmetric difference of two sets of integers sorted in ascending order.

Return The end of the output range.

Pre The ranges `[first1, last1)` and `[first2, last2)` shall be sorted with respect to `operator<`.

Pre The resulting range shall not overlap with either input range.

Parameters

- `first1`: The beginning of the first input range.
- `last1`: The end of the first input range.
- `first2`: The beginning of the second input range.
- `last2`: The end of the second input range.
- `result`: The beginning of the output range.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `OutputIterator`: is a model of [Output Iterator](#).

```
#include <thrust/set_operations.h>
...
int A1[6] = {0, 1, 2, 2, 4, 6, 7};
int A2[5] = {1, 1, 2, 5, 8};

int result[6];

int *result_end = thrust::set_symmetric_difference(A1, A1 + 6, A2, A2 + 5,
↪result);
// result = {0, 4, 5, 6, 7, 8}
```

See http://www.sgi.com/tech/stl/set_symmetric_difference.html

See `merge`

See `includes`

See `set_difference`

See `set_union`

See `set_intersection`

See `sort`

See `is_sorted`

Template Function `thrust::set_symmetric_difference(const thrust::detail::execution_policy_base<DerivedPolicy>& InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::set_symmetric_difference`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `InputIterator1`, `InputIterator1`, `InputIterator2`, `InputIterator2`, `OutputIterator`, `StrictWeakCompare`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename OutputIterator, typename StrictWeakCompare>__host__ __device__
    ↪OutputIterator thrust::set_symmetric_difference(const thrust::detail::execution_
    ↪policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
    ↪InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename OutputIterator>__host__ __device__ OutputIterator thrust::set_symmetric_
    ↪difference(const thrust::detail::execution_policy_base< DerivedPolicy > &,
    ↪InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪OutputIterator, typename StrictWeakCompare>
    ↪OutputIterator thrust::set_symmetric_difference(InputIterator1, InputIterator1,
    ↪InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪OutputIterator>
    ↪OutputIterator thrust::set_symmetric_difference(InputIterator1, InputIterator1,
    ↪InputIterator2, InputIterator2, OutputIterator)
```

Template Function `thrust::set_symmetric_difference(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**, typename **StrictWeakCompare**>
OutputIterator thrust::set_symmetric_difference(*InputIterator1 first1, InputIterator1 last1, In-*
putIterator2 first2, InputIterator2 last2, Out-
putIterator result, StrictWeakCompare comp)

`set_symmetric_difference` constructs a sorted range that is the set symmetric difference of the sorted ranges `[first1, last1)` and `[first2, last2)`. The return value is the end of the output range.

In the simplest case, `set_symmetric_difference` performs a set theoretic calculation: it constructs the union of the two sets $A - B$ and $B - A$, where A and B are the two input ranges. That is, the output range contains a copy of every element that is contained in `[first1, last1)` but not `[first2, last1)`, and a copy of every element that is contained in `[first2, last2)` but not `[first1, last1)`. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if `[first1, last1)` contains m elements that are equivalent to each other and `[first2, last1)` contains n elements that are equivalent to them, then $|m - n|$ of those elements shall be copied to the output range: the last $m - n$ elements from `[first1, last1)` if $m > n$, and the last $n - m$ of these elements from `[first2, last2)` if $m < n$.

This version of `set_union` compares elements using a function object `comp`.

The following code snippet demonstrates how to use `set_symmetric_difference` to compute the symmetric difference of two sets of integers sorted in descending order.

Return The end of the output range.

Pre The ranges `[first1, last1)` and `[first2, last2)` shall be sorted with respect to `comp`.

Pre The resulting range shall not overlap with either input range.

Parameters

- `first1`: The beginning of the first input range.
- `last1`: The end of the first input range.
- `first2`: The beginning of the second input range.
- `last2`: The end of the second input range.
- `result`: The beginning of the output range.
- `comp`: Comparison operator.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `OutputIterator`: is a model of [Output Iterator](#).

```
#include <thrust/set_operations.h>
...
int A1[6] = {7, 6, 4, 2, 2, 1, 0};
int A2[5] = {8, 5, 2, 1, 1};

int result[6];

int *result_end = thrust::set_symmetric_difference(A1, A1 + 6, A2, A2 + 5,
↪result);
// result = {8, 7, 6, 5, 4, 0}
```

See http://www.sgi.com/tech/stl/set_symmetric_difference.html

See `merge`

See `includes`

See `set_difference`

See `set_union`

See `set_intersection`

See `sort`

See `is_sorted`

Template Function `thrust::set_symmetric_difference_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)`

- Defined in file `_thrust_set_operations.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::set_symmetric_difference_by_key`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
    ↪ typename OutputIterator2, typename StrictWeakCompare>__host__ __device__
    ↪ thrust::pair<OutputIterator1, OutputIterator2> thrust::set_symmetric_difference_by_
    ↪ key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
    ↪ InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3,
    ↪ InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
    ↪ typename OutputIterator2>__host__ __device__ thrust::pair<OutputIterator1,
    ↪ OutputIterator2> thrust::set_symmetric_difference_by_key(const
    ↪ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
    ↪ InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4,
    ↪ OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ InputIterator3, typename InputIterator4, typename OutputIterator1, typename
    ↪ OutputIterator2, typename StrictWeakCompare>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::set_symmetric_difference_
    ↪ by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪ InputIterator3, InputIterator4, OutputIterator1, OutputIterator2,
    ↪ StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪ InputIterator3, typename InputIterator4, typename OutputIterator1, typename
    ↪ OutputIterator2>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::set_symmetric_difference_
    ↪ by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪ InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)
```

Template Function `thrust::set_symmetric_difference_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)`

Function Documentation

template<typename `InputIterator1`, typename `InputIterator2`, typename `InputIterator3`, typename `InputIterator4`, typename `OutputIterator1`, typename `OutputIterator2`>

```

thrust::pair<OutputIterator1, OutputIterator2> thrust::set_symmetric_difference_by_key (InputIterator1
keys_first1,
In-
putIt-
er-
a-
tor1
keys_last1,
In-
putIt-
er-
a-
tor2
keys_first2,
In-
putIt-
er-
a-
tor2
keys_last2,
In-
putIt-
er-
a-
tor3
val-
ues_first1,
In-
putIt-
er-
a-
tor4
val-
ues_first2,
Out-
putIt-
er-
a-
tor1
keys_result,
Out-
putIt-
er-
a-
tor2
val-
ues_result)

```

`set_symmetric_difference_by_key` performs a key-value symmetric difference operation from set theory. `set_difference_by_key` constructs a sorted range that is the symmetric difference of the sorted ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)`. Associated with each element from the input and output key ranges is a value element. The associated input value ranges need not be sorted.

In the simplest case, `set_symmetric_difference_by_key` performs a set theoretic calculation: it constructs the union of the two sets $A - B$ and $B - A$, where A and B are the two input ranges. That is, the

output range contains a copy of every element that is contained in `[keys_first1, keys_last1)` but not `[keys_first2, keys_last1)`, and a copy of every element that is contained in `[keys_first2, keys_last2)` but not `[keys_first1, keys_last1)`. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if `[keys_first1, keys_last1)` contains m elements that are equivalent to each other and `[keys_first2, keys_last1)` contains n elements that are equivalent to them, then $|m - n|$ of those elements shall be copied to the output range: the last $m - n$ elements from `[keys_first1, keys_last1)` if $m > n$, and the last $n - m$ of these elements from `[keys_first2, keys_last2)` if $m < n$.

Each time a key element is copied from `[keys_first1, keys_last1)` or `[keys_first2, keys_last2)` is copied to the keys output range, the corresponding value element is copied from the corresponding values input range (beginning at `values_first1` or `values_first2`) to the values output range.

This version of `set_symmetric_difference_by_key` compares key elements using `operator<`.

The following code snippet demonstrates how to use `set_symmetric_difference_by_key` to compute the symmetric difference of two sets of integers sorted in ascending order with their values.

Return A pair `p` such that `p.first` is the end of the output range of keys, and such that `p.second` is the end of the output range of values.

Pre The ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)` shall be sorted with respect to `operator<`.

Pre The resulting ranges shall not overlap with any input range.

Parameters

- `keys_first1`: The beginning of the first input range of keys.
- `keys_last1`: The end of the first input range of keys.
- `keys_first2`: The beginning of the second input range of keys.
- `keys_last2`: The end of the second input range of keys.
- `values_first1`: The beginning of the first input range of values.
- `values_first2`: The beginning of the first input range of values.
- `keys_result`: The beginning of the output range of keys.
- `values_result`: The beginning of the output range of values.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator3`: is a model of [Input Iterator](#), and `InputIterator3`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `InputIterator4`: is a model of [Input Iterator](#), and `InputIterator4`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.

- OutputIterator1: is a model of Output Iterator.
- OutputIterator2: is a model of Output Iterator.

```
#include <thrust/set_operations.h>
...
int A_keys[6] = {0, 1, 2, 2, 4, 6, 7};
int A_vals[6] = {0, 0, 0, 0, 0, 0, 0};

int B_keys[5] = {1, 1, 2, 5, 8};
int B_vals[5] = {1, 1, 1, 1, 1};

int keys_result[6];
int vals_result[6];

thrust::pair<int*,int*> end = thrust::set_symmetric_difference_by_key(A_keys, A_
↳keys + 6, B_keys, B_keys + 5, A_vals, B_vals, keys_result, vals_result);
// keys_result is now {0, 4, 5, 6, 7, 8}
// vals_result is now {0, 0, 1, 0, 0, 1}
```

See `set_union_by_key`

See `set_intersection_by_key`

See `set_difference_by_key`

See `sort_by_key`

See `is_sorted`

Template Function `thrust::set_symmetric_difference_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::set_symmetric_difference_by_key`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `InputIterator1`, `InputIterator1`, `InputIterator2`, `InputIterator2`, `InputIterator3`, `InputIterator4`, `OutputIterator1`, `OutputIterator2`, `StrictWeakCompare`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
↳ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
↳ typename OutputIterator2, typename StrictWeakCompare>__host__ __device__
↳ thrust::pair<OutputIterator1,OutputIterator2> thrust::set_symmetric_difference_by_
↳ key(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3,
↳ InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
↳ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
↳ typename OutputIterator2>__host__ __device__ thrust::pair<OutputIterator1,
↳ OutputIterator2> thrust::set_symmetric_difference_by_key(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
↳ InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4,
↳ OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
↳ InputIterator3, typename InputIterator4, typename OutputIterator1, typename
↳ OutputIterator2, typename StrictWeakCompare>
```

1.3. Full API `thrust::pair<OutputIterator1, OutputIterator2> thrust::set_symmetric_difference_ 285`
`by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,`
`InputIterator3, InputIterator4, OutputIterator1, OutputIterator2,`
`StrictWeakCompare)`

```
- template<typename InputIterator1, typename InputIterator2, typename
```

Template Function `thrust::set_symmetric_difference_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)`

Function Documentation

template<typename `InputIterator1`, typename `InputIterator2`, typename `InputIterator3`, typename `InputIterator4`, typename `OutputIterator1`, typename `OutputIterator2`, typename `StrictWeakCompare`>

```

thrust::pair<OutputIterator1, OutputIterator2> thrust::set_symmetric_difference_by_key (InputIterator1
keys_first1,
In-
putIt-
er-
a-
tor1
keys_last1,
In-
putIt-
er-
a-
tor2
keys_first2,
In-
putIt-
er-
a-
tor2
keys_last2,
In-
putIt-
er-
a-
tor3
val-
ues_first1,
In-
putIt-
er-
a-
tor4
val-
ues_first2,
Out-
putIt-
er-
a-
tor1
keys_result,
Out-
putIt-
er-
a-
tor2
val-
ues_result,
StrictWeak-
Com-
pare
comp)

```

`set_symmetric_difference_by_key` performs a key-value symmetric difference operation from set theory. `set_difference_by_key` constructs a sorted range that is the symmetric difference of the sorted ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)`. Associated with each

element from the input and output key ranges is a value element. The associated input value ranges need not be sorted.

In the simplest case, `set_symmetric_difference_by_key` performs a set theoretic calculation: it constructs the union of the two sets $A - B$ and $B - A$, where A and B are the two input ranges. That is, the output range contains a copy of every element that is contained in $[keys_first1, keys_last1)$ but not $[keys_first2, keys_last1)$, and a copy of every element that is contained in $[keys_first2, keys_last2)$ but not $[keys_first1, keys_last1)$. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if $[keys_first1, keys_last1)$ contains m elements that are equivalent to each other and $[keys_first2, keys_last1)$ contains n elements that are equivalent to them, then $|m - n|$ of those elements shall be copied to the output range: the last $m - n$ elements from $[keys_first1, keys_last1)$ if $m > n$, and the last $n - m$ of these elements from $[keys_first2, keys_last2)$ if $m < n$.

Each time a key element is copied from $[keys_first1, keys_last1)$ or $[keys_first2, keys_last2)$ is copied to the keys output range, the corresponding value element is copied from the corresponding values input range (beginning at `values_first1` or `values_first2`) to the values output range.

This version of `set_symmetric_difference_by_key` compares key elements using a function object `comp`.

The following code snippet demonstrates how to use `set_symmetric_difference_by_key` to compute the symmetric difference of two sets of integers sorted in descending order with their values.

Return A pair `p` such that `p.first` is the end of the output range of keys, and such that `p.second` is the end of the output range of values.

Pre The ranges $[keys_first1, keys_last1)$ and $[keys_first2, keys_last2)$ shall be sorted with respect to `comp`.

Pre The resulting ranges shall not overlap with any input range.

Parameters

- `keys_first1`: The beginning of the first input range of keys.
- `keys_last1`: The end of the first input range of keys.
- `keys_first2`: The beginning of the second input range of keys.
- `keys_last2`: The end of the second input range of keys.
- `values_first1`: The beginning of the first input range of values.
- `values_first2`: The beginning of the first input range of values.
- `keys_result`: The beginning of the output range of keys.
- `values_result`: The beginning of the output range of values.
- `comp`: Comparison operator.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined

in the [LessThan Comparable](#) requirements, and InputIterator2's value_type is convertible to a type in OutputIterator's set of value_types.

- InputIterator3: is a model of [Input Iterator](#), and InputIterator3's value_type is convertible to a type in OutputIterator2's set of value_types.
- InputIterator4: is a model of [Input Iterator](#), and InputIterator4's value_type is convertible to a type in OutputIterator2's set of value_types.
- OutputIterator1: is a model of [Output Iterator](#).
- OutputIterator2: is a model of [Output Iterator](#).
- StrictWeakCompare: is a model of [Strict Weak Ordering](#).

```
#include <thrust/set_operations.h>
#include <thrust/functional.h>
...
int A_keys[6] = {7, 6, 4, 2, 2, 1, 0};
int A_vals[6] = {0, 0, 0, 0, 0, 0, 0};

int B_keys[5] = {8, 5, 2, 1, 1};
int B_vals[5] = {1, 1, 1, 1, 1};

int keys_result[6];
int vals_result[6];

thrust::pair<int*,int*> end = thrust::set_symmetric_difference_by_key(A_keys, A_
    ↪keys + 6, B_keys, B_keys + 5, A_vals, B_vals, keys_result, vals_result);
// keys_result is now {8, 7, 6, 5, 4, 0}
// vals_result is now {1, 0, 0, 1, 0, 0}
```

See [set_union_by_key](#)

See [set_intersection_by_key](#)

See [set_difference_by_key](#)

See [sort_by_key](#)

See [is_sorted](#)

Template Function `thrust::set_union(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`

- Defined in file `thrust_set_operations.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::set_union” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename StrictWeakCompare>__host__ __device__
  ↳ OutputIterator thrust::set_union(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2, InputIterator2,
  ↳ OutputIterator, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::set_
  ↳ union(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename StrictWeakCompare>
  OutputIterator thrust::set_union(InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator>
  OutputIterator thrust::set_union(InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator2, OutputIterator)

```

Template Function `thrust::set_union(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**>
OutputIterator thrust::set_union(*InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, In-*
putIterator2 last2, OutputIterator result)

set_union constructs a sorted range that is the union of the sorted ranges [first1, last1) and [first2, last2). The return value is the end of the output range.

In the simplest case, set_union performs the “union” operation from set theory: the output range contains a copy of every element that is contained in [first1, last1), [first2, last2), or both. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if [first1, last1) contains *m* elements that are equivalent to each other and if [first2, last2) contains *n* elements that are equivalent to them, then all *m* elements from the first range shall be copied to the output range, in order, and then $\max(n - m, 0)$ elements from the second range shall be copied to the output, in order.

This version of set_union compares elements using operator<.

The following code snippet demonstrates how to use set_union to compute the union of two sets of integers sorted in ascending order.

Return The end of the output range.

Pre The ranges [first1, last1) and [first2, last2) shall be sorted with respect to operator<.

Pre The resulting range shall not overlap with either input range.

Parameters

- first1: The beginning of the first input range.
- last1: The end of the first input range.
- first2: The beginning of the second input range.
- last2: The end of the second input range.

- `result`: The beginning of the output range.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `OutputIterator`: is a model of [Output Iterator](#).

```
#include <thrust/set_operations.h>
...
int A1[7] = {0, 2, 4, 6, 8, 10, 12};
int A2[5] = {1, 3, 5, 7, 9};

int result[11];

int *result_end = thrust::set_union(A1, A1 + 7, A2, A2 + 5, result);
// result = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12}
```

See http://www.sgi.com/tech/stl/set_union.html

See `merge`

See `includes`

See `set_union`

See `set_intersection`

See `set_symmetric_difference`

See `sort`

See `is_sorted`

Template Function `thrust::set_union(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::set_union`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename StrictWeakCompare>__host__ __device__
  ↳ OutputIterator thrust::set_union(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2, InputIterator2,
  ↳ OutputIterator, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::set_
  ↳ union(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename StrictWeakCompare>
  OutputIterator thrust::set_union(InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator2, OutputIterator, StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator>
  OutputIterator thrust::set_union(InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator2, OutputIterator)

```

Template Function `thrust::set_union(InputIterator1, InputIterator1, InputIterator2, InputIterator2, OutputIterator, StrictWeakCompare)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**, typename **StrictWeakCompare**>
OutputIterator thrust::set_union(*InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, In-*
putIterator2 last2, OutputIterator result, StrictWeakCompare comp)

set_union constructs a sorted range that is the union of the sorted ranges [first1, last1) and [first2, last2). The return value is the end of the output range.

In the simplest case, set_union performs the “union” operation from set theory: the output range contains a copy of every element that is contained in [first1, last1), [first2, last2), or both. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if [first1, last1) contains *m* elements that are equivalent to each other and if [first2, last2) contains *n* elements that are equivalent to them, then all *m* elements from the first range shall be copied to the output range, in order, and then $\max(n - m, 0)$ elements from the second range shall be copied to the output, in order.

This version of set_union compares elements using a function object comp.

The following code snippet demonstrates how to use set_union to compute the union of two sets of integers sorted in ascending order.

Return The end of the output range.

Pre The ranges [first1, last1) and [first2, last2) shall be sorted with respect to comp.

Pre The resulting range shall not overlap with either input range.

Parameters

- first1: The beginning of the first input range.
- last1: The end of the first input range.
- first2: The beginning of the second input range.
- last2: The end of the second input range.

- `result`: The beginning of the output range.
- `comp`: Comparison operator.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1`'s `value_type` is convertible to `StrictWeakCompare`'s `first_argument_type`. and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2`'s `value_type` is convertible to `StrictWeakCompare`'s `second_argument_type`. and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `OutputIterator`: is a model of [Output Iterator](#).
- `StrictWeakCompare`: is a model of [Strict Weak Ordering](#).

```
#include <thrust/set_operations.h>
#include <thrust/functional.h>
...
int A1[7] = {12, 10, 8, 6, 4, 2, 0};
int A2[5] = {9, 7, 5, 3, 1};

int result[11];

int *result_end = thrust::set_union(A1, A1 + 7, A2, A2 + 5, result,
    thrust::greater<int>());
// result = {12, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0}
```

See http://www.sgi.com/tech/stl/set_union.html

See [merge](#)

See [includes](#)

See [set_union](#)

See [set_intersection](#)

See [set_symmetric_difference](#)

See [sort](#)

See [is_sorted](#)

Template Function `thrust::set_union_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)`

- Defined in `file_thrust_set_operations.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::set_union_by_key`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `InputIterator1`, `InputIterator1`, `InputIterator2`, `InputIterator2`, `InputIterator3`, `InputIterator4`, `OutputIterator1`, `OutputIterator2`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
  ↳ typename OutputIterator2, typename StrictWeakCompare> __host__ __device__
  ↳ thrust::pair<OutputIterator1, OutputIterator2> thrust::set_union_by_key(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4,
  ↳ OutputIterator1, OutputIterator2, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
  ↳ typename OutputIterator2> __host__ __device__ thrust::pair<OutputIterator1,
  ↳ OutputIterator2> thrust::set_union_by_key(const thrust::detail::execution_policy_
  ↳ base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ InputIterator3, typename InputIterator4, typename OutputIterator1, typename
  ↳ OutputIterator2, typename StrictWeakCompare>
  thrust::pair<OutputIterator1, OutputIterator2> thrust::set_union_by_
  ↳ key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
  ↳ InputIterator3, InputIterator4, OutputIterator1, OutputIterator2,
  ↳ StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ InputIterator3, typename InputIterator4, typename OutputIterator1, typename
  ↳ OutputIterator2>
  thrust::pair<OutputIterator1, OutputIterator2> thrust::set_union_by_
  ↳ key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
  ↳ InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)

```

Template Function `thrust::set_union_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **InputIterator3**, typename **InputIterator4**, typename **OutputIterator1**, typename **OutputIterator2**> thrust::pair<**OutputIterator1**, **OutputIterator2**> thrust::set_union_by_key

(*InputIterator1* keys_first1,
InputIterator1 keys_last1,
InputIterator2 keys_first2,
InputIterator2 keys_last2,
InputIterator3 val-
ues_first1, *InputIterator4*
values_first2, *OutputIt-*
erator1 keys_result,
OutputIterator2 val-
ues_result)

`set_union_by_key` performs a key-value union operation from set theory. `set_union_by_key` constructs a sorted range that is the union of the sorted ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)`. Associated with each element from the input and output key ranges is a value element. The associated input value ranges need not be sorted.

In the simplest case, `set_union_by_key` performs the “union” operation from set theory: the output range contains a copy of every element that is contained in `[keys_first1, keys_last1)`, `[keys_first2, keys_last1)`, or both. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if `[keys_first1, keys_last1)` contains `m` elements that are equivalent to each other and if `[keys_first2, keys_last2)` contains `n` elements that are equivalent to

them, then all m elements from the first range shall be copied to the output range, in order, and then $\max(n - m, 0)$ elements from the second range shall be copied to the output, in order.

Each time a key element is copied from $[\text{keys_first1}, \text{keys_last1})$ or $[\text{keys_first2}, \text{keys_last2})$ is copied to the keys output range, the corresponding value element is copied from the corresponding values input range (beginning at values_first1 or values_first2) to the values output range.

This version of `set_union_by_key` compares key elements using `operator<`.

The following code snippet demonstrates how to use `set_symmetric_difference_by_key` to compute the symmetric difference of two sets of integers sorted in ascending order with their values.

Return A pair `p` such that `p.first` is the end of the output range of keys, and such that `p.second` is the end of the output range of values.

Pre The ranges $[\text{keys_first1}, \text{keys_last1})$ and $[\text{keys_first2}, \text{keys_last2})$ shall be sorted with respect to `operator<`.

Pre The resulting ranges shall not overlap with any input range.

Parameters

- `keys_first1`: The beginning of the first input range of keys.
- `keys_last1`: The end of the first input range of keys.
- `keys_first2`: The beginning of the second input range of keys.
- `keys_last2`: The end of the second input range of keys.
- `values_first1`: The beginning of the first input range of values.
- `values_first2`: The beginning of the first input range of values.
- `keys_result`: The beginning of the output range of keys.
- `values_result`: The beginning of the output range of values.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator3`: is a model of [Input Iterator](#), and `InputIterator3`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `InputIterator4`: is a model of [Input Iterator](#), and `InputIterator4`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `OutputIterator1`: is a model of [Output Iterator](#).
- `OutputIterator2`: is a model of [Output Iterator](#).

```
#include <thrust/set_operations.h>
...
int A_keys[6] = {0, 2, 4, 6, 8, 10, 12};
int A_vals[6] = {0, 0, 0, 0, 0, 0};

int B_keys[5] = {1, 3, 5, 7, 9};
int B_vals[5] = {1, 1, 1, 1, 1};

int keys_result[11];
int vals_result[11];

thrust::pair<int*,int*> end = thrust::set_symmetric_difference_by_key(A_keys, A_
    ↪keys + 6, B_keys, B_keys + 5, A_vals, B_vals, keys_result, vals_result);
// keys_result is now {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12}
// vals_result is now {0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0}
```

See `set_symmetric_difference_by_key`

See `set_intersection_by_key`

See `set_difference_by_key`

See `sort_by_key`

See `is_sorted`

Template Function `thrust::set_union_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::set_union_by_key`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
    ↪ typename OutputIterator2, typename StrictWeakCompare>__host__ __device__
    ↪thrust::pair<OutputIterator1,OutputIterator2> thrust::set_union_by_key(const
    ↪thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
    ↪InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4,
    ↪OutputIterator1, OutputIterator2, StrictWeakCompare)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename InputIterator3, typename InputIterator4, typename OutputIterator1,
    ↪ typename OutputIterator2>__host__ __device__ thrust::pair<OutputIterator1,
    ↪OutputIterator2> thrust::set_union_by_key(const thrust::detail::execution_policy_
    ↪base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
    ↪InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪InputIterator3, typename InputIterator4, typename OutputIterator1, typename
    ↪OutputIterator2, typename StrictWeakCompare>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::set_union_by_
    ↪key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪InputIterator3, InputIterator4, OutputIterator1, OutputIterator2,
```

```
    ↪StrictWeakCompare)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪InputIterator3, typename InputIterator4, typename OutputIterator1, typename
    ↪OutputIterator2>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::set_union_by_
    ↪key(InputIterator1, InputIterator1, InputIterator2, InputIterator2,
    ↪InputIterator3, InputIterator4, OutputIterator1, OutputIterator2)
```

Template Function `thrust::set_union_by_key(InputIterator1, InputIterator1, InputIterator2, InputIterator2, InputIterator3, InputIterator4, OutputIterator1, OutputIterator2, StrictWeakCompare)`

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename InputIterator3, typename InputIterator4,
        thrust::pair<OutputIterator1, OutputIterator2> thrust::set_union_by_key (InputIterator1 keys_first1,
                                                                              InputIterator1 keys_last1,
                                                                              InputIterator2 keys_first2,
                                                                              InputIterator2 keys_last2,
                                                                              InputIterator3 values_first1,
                                                                              InputIterator4 values_first2,
                                                                              OutputIterator1 keys_result,
                                                                              OutputIterator2 values_result,
                                                                              StrictWeakCompare comp)
```

`set_union_by_key` performs a key-value union operation from set theory. `set_union_by_key` constructs a sorted range that is the union of the sorted ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)`. Associated with each element from the input and output key ranges is a value element. The associated input value ranges need not be sorted.

In the simplest case, `set_union_by_key` performs the “union” operation from set theory: the output range contains a copy of every element that is contained in `[keys_first1, keys_last1)`, `[keys_first2, keys_last2)`, or both. The general case is more complicated, because the input ranges may contain duplicate elements. The generalization is that if `[keys_first1, keys_last1)` contains m elements that are equivalent to each other and if `[keys_first2, keys_last2)` contains n elements that are equivalent to them, then all m elements from the first range shall be copied to the output range, in order, and then $\max(n - m, 0)$ elements from the second range shall be copied to the output, in order.

Each time a key element is copied from `[keys_first1, keys_last1)` or `[keys_first2, keys_last2)` is copied to the keys output range, the corresponding value element is copied from the corresponding values input range (beginning at `values_first1` or `values_first2`) to the values output range.

This version of `set_union_by_key` compares key elements using a function object `comp`.

The following code snippet demonstrates how to use `set_symmetric_difference_by_key` to compute the symmetric difference of two sets of integers sorted in descending order with their values.

Return A pair `p` such that `p.first` is the end of the output range of keys, and such that `p.second` is the end of the output range of values.

Pre The ranges `[keys_first1, keys_last1)` and `[keys_first2, keys_last2)` shall be sorted with respect to `comp`.

Pre The resulting ranges shall not overlap with any input range.

Parameters

- `keys_first1`: The beginning of the first input range of keys.
- `keys_last1`: The end of the first input range of keys.
- `keys_first2`: The beginning of the second input range of keys.

- `keys_last2`: The end of the second input range of keys.
- `values_first1`: The beginning of the first input range of values.
- `values_first2`: The beginning of the first input range of values.
- `keys_result`: The beginning of the output range of keys.
- `values_result`: The beginning of the output range of values.
- `comp`: Comparison operator.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), `InputIterator1` and `InputIterator2` have the same `value_type`, `InputIterator1`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator1`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator1`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), `InputIterator2` and `InputIterator1` have the same `value_type`, `InputIterator2`'s `value_type` is a model of [LessThan Comparable](#), the ordering on `InputIterator2`'s `value_type` is a strict weak ordering, as defined in the [LessThan Comparable](#) requirements, and `InputIterator2`'s `value_type` is convertible to a type in `OutputIterator`'s set of `value_types`.
- `InputIterator3`: is a model of [Input Iterator](#), and `InputIterator3`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `InputIterator4`: is a model of [Input Iterator](#), and `InputIterator4`'s `value_type` is convertible to a type in `OutputIterator2`'s set of `value_types`.
- `OutputIterator1`: is a model of [Output Iterator](#).
- `OutputIterator2`: is a model of [Output Iterator](#).
- `StrictWeakCompare`: is a model of [Strict Weak Ordering](#).

```
#include <thrust/set_operations.h>
#include <thrust/functional.h>
...
int A_keys[6] = {12, 10, 8, 6, 4, 2, 0};
int A_vals[6] = { 0,  0, 0, 0, 0, 0, 0};

int B_keys[5] = {9, 7, 5, 3, 1};
int B_vals[5] = {1, 1, 1, 1, 1};

int keys_result[11];
int vals_result[11];

thrust::pair<int*,int*> end = thrust::set_symmetric_difference_by_key(A_keys, A_
    ↪keys + 6, B_keys, B_keys + 5, A_vals, B_vals, keys_result, vals_result,
    ↪thrust::greater<int>());
// keys_result is now {12, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0}
// vals_result is now { 0,  1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0}
```

See `set_symmetric_difference_by_key`

See `set_intersection_by_key`

See `set_difference_by_key`

See `sort_by_key`

See `is_sorted`

Template Function `thrust::sin`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::sin(const complex < T > & z)
Returns the complex sine of a complex number.

Parameters

- `z`: The complex argument.

Template Function `thrust::sinh`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::sinh(const complex < T > & z)
Returns the complex hyperbolic sine of a complex number.

Parameters

- `z`: The complex argument.

Template Function `thrust::sort(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator, RandomAccessIterator)`

- Defined in `file_thrust_sort.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::sort” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator, RandomAccessIterator) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename RandomAccessIterator, typename
↳ StrictWeakOrdering>__host__ __device__ void thrust::sort(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, RandomAccessIterator,
↳ RandomAccessIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename RandomAccessIterator>__host__ __device__
↳ void thrust::sort(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ RandomAccessIterator, RandomAccessIterator)
- template<typename RandomAccessIterator, typename StrictWeakOrdering>__host__ __
↳ device__ void thrust::sort(RandomAccessIterator, RandomAccessIterator,
↳ StrictWeakOrdering)
- template<typename RandomAccessIterator>
↳ void thrust::sort(RandomAccessIterator, RandomAccessIterator)
```

Template Function `thrust::sort(RandomAccessIterator, RandomAccessIterator)`

Function Documentation

template<typename **RandomAccessIterator**>

void `thrust::sort` (*RandomAccessIterator first, RandomAccessIterator last*)

`sort` sorts the elements in `[first, last)` into ascending order, meaning that if `i` and `j` are any two valid iterators in `[first, last)` such that `i` precedes `j`, then `*j` is not less than `*i`. Note: `sort` is not guaranteed to be stable. That is, suppose that `*i` and `*j` are equivalent: neither one is less than the other. It is not guaranteed that the relative order of these two elements will be preserved by `sort`.

This version of `sort` compares objects using `operator<`.

The following code snippet demonstrates how to use `sort` to sort a sequence of integers.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.

Template Parameters

- `RandomAccessIterator`: is a model of [Random Access Iterator](#), `RandomAccessIterator` is mutable, and `RandomAccessIterator`'s `value_type` is a model of [LessThan Comparable](#), and the ordering relation on `RandomAccessIterator`'s `value_type` is a *strict weak ordering*, as defined in the [LessThan Comparable](#) requirements.

```
#include <thrust/sort.h>
...
const int N = 6;
int A[N] = {1, 4, 2, 8, 5, 7};
thrust::sort(A, A + N);
// A is now {1, 2, 4, 5, 7, 8}
```

See <http://www.sgi.com/tech/stl/sort.html>

See `stable_sort`

See `sort_by_key`

Template Function `thrust::sort(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator, RandomAccessIterator, StrictWeakOrdering)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::sort`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `RandomAccessIterator`, `RandomAccessIterator`, `StrictWeakOrdering`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:


```

- template<typename DerivedPolicy, typename RandomAccessIterator, typename
↳ StrictWeakOrdering>__host__ __device__ void thrust::sort(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, RandomAccessIterator,
↳ RandomAccessIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename RandomAccessIterator>__host__ __device__
↳ void thrust::sort(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ RandomAccessIterator, RandomAccessIterator)
- template<typename RandomAccessIterator, typename StrictWeakOrdering>__host__ __
↳ device__ void thrust::sort(RandomAccessIterator, RandomAccessIterator,
↳ StrictWeakOrdering)
- template<typename RandomAccessIterator>
  void thrust::sort(RandomAccessIterator, RandomAccessIterator)

```

Template Function `thrust::sort(RandomAccessIterator, RandomAccessIterator, StrictWeakOrdering)`

Function Documentation

template<typename RandomAccessIterator, typename StrictWeakOrdering>__host__ __device__ void

`sort` sorts the elements in `[first, last)` into ascending order, meaning that if `i` and `j` are any two valid iterators in `[first, last)` such that `i` precedes `j`, then `*j` is not less than `*i`. Note: `sort` is not guaranteed to be stable. That is, suppose that `*i` and `*j` are equivalent: neither one is less than the other. It is not guaranteed that the relative order of these two elements will be preserved by `sort`.

This version of `sort` compares objects using a function object `comp`.

The following code demonstrates how to sort integers in descending order using the `greater<int>` comparison operator.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.
- `comp`: Comparison operator.

Template Parameters

- `RandomAccessIterator`: is a model of [Random Access Iterator](#), `RandomAccessIterator` is mutable, and `RandomAccessIterator`'s `value_type` is convertible to `StrictWeakOrdering`'s `first_argument_type` and `second_argument_type`.
- `StrictWeakOrdering`: is a model of [Strict Weak Ordering](#).

```

#include <thrust/sort.h>
#include <thrust/functional.h>
...
const int N = 6;
int A[N] = {1, 4, 2, 8, 5, 7};
thrust::sort(A, A + N, thrust::greater<int>());
// A is now {8, 7, 5, 4, 2, 1};

```

See <http://www.sgi.com/tech/stl/sort.html>

See `stable_sort`

See `sort_by_key`

Template Function `thrust::sort_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2)`

- Defined in `file_thrust_sort.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::sort_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename RandomAccessIterator1, typename
↳ RandomAccessIterator2, typename StrictWeakOrdering>__host__ __device__ void
↳ thrust::sort_by_key(const thrust::detail::execution_policy_base< DerivedPolicy > &
↳ , RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2,
↳ StrictWeakOrdering)
- template<typename DerivedPolicy, typename RandomAccessIterator1, typename
↳ RandomAccessIterator2>__host__ __device__ void thrust::sort_by_key(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, RandomAccessIterator1,
↳ RandomAccessIterator1, RandomAccessIterator2)
- template<typename RandomAccessIterator1, typename RandomAccessIterator2, typename
↳ StrictWeakOrdering>
  void thrust::sort_by_key(RandomAccessIterator1, RandomAccessIterator1,
↳ RandomAccessIterator2, StrictWeakOrdering)
- template<typename RandomAccessIterator1, typename RandomAccessIterator2>
  void thrust::sort_by_key(RandomAccessIterator1, RandomAccessIterator1,
↳ RandomAccessIterator2)
```

Template Function `thrust::sort_by_key(RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2)`

Function Documentation

`template<typename RandomAccessIterator1, typename RandomAccessIterator2>`

`void thrust:::sort_by_key (RandomAccessIterator1 keys_first, RandomAccessIterator1 keys_last, RandomAccessIterator2 values_first)`

`sort_by_key` performs a key-value sort. That is, `sort_by_key` sorts the elements in `[keys_first, keys_last)` and `[values_first, values_first + (keys_last - keys_first))` into ascending key order, meaning that if `i` and `j` are any two valid iterators in `[keys_first, keys_last)` such that `i` precedes `j`, and `p` and `q` are iterators in `[values_first, values_first + (keys_last - keys_first))` corresponding to `i` and `j` respectively, then `*j` is not less than `*i`.

Note: `sort_by_key` is not guaranteed to be stable. That is, suppose that `*i` and `*j` are equivalent: neither one is less than the other. It is not guaranteed that the relative order of these two keys or the relative order of their corresponding values will be preserved by `sort_by_key`.

This version of `sort_by_key` compares key objects using `operator<`.

The following code snippet demonstrates how to use `sort_by_key` to sort an array of character values using integers as sorting keys.

Pre The range `[keys_first, keys_last)` shall not overlap the range `[values_first, values_first + (keys_last - keys_first))`.

Parameters

- `keys_first`: The beginning of the key sequence.
- `keys_last`: The end of the key sequence.
- `values_first`: The beginning of the value sequence.

Template Parameters

- `RandomAccessIterator1`: is a model of `Random Access Iterator`, `RandomAccessIterator1` is mutable, and `RandomAccessIterator1`'s `value_type` is a model of `LessThan Comparable`, and the ordering relation on `RandomAccessIterator1`'s `value_type` is a *strict weak ordering*, as defined in the `LessThan Comparable` requirements.
- `RandomAccessIterator2`: is a model of `Random Access Iterator`, and `RandomAccessIterator2` is mutable.

```
#include <thrust/sort.h>
...
const int N = 6;
int keys[N] = { 1, 4, 2, 8, 5, 7};
char values[N] = {'a', 'b', 'c', 'd', 'e', 'f'};
thrust::sort_by_key(keys, keys + N, values);
// keys is now { 1, 2, 4, 5, 7, 8}
// values is now {'a', 'c', 'b', 'e', 'f', 'd'}
```

See <http://www.sgi.com/tech/stl/sort.html>

See `stable_sort_by_key`

See `sort`

Template Function `thrust::sort_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2, StrictWeakOrdering)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::sort_by_key`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2, StrictWeakOrdering)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename RandomAccessIterator1, typename
  ↳RandomAccessIterator2, typename StrictWeakOrdering>__host__ __device__ void
  ↳thrust::sort_by_key(const thrust::detail::execution_policy_base< DerivedPolicy > &
  ↳, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2,
  ↳StrictWeakOrdering)
- template<typename DerivedPolicy, typename RandomAccessIterator1, typename
  ↳RandomAccessIterator2>__host__ __device__ void thrust::sort_by_key(const
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, RandomAccessIterator1,
  ↳RandomAccessIterator1, RandomAccessIterator2)
- template<typename RandomAccessIterator1, typename RandomAccessIterator2, typename
  ↳StrictWeakOrdering>
  void thrust::sort_by_key(RandomAccessIterator1, RandomAccessIterator1,
  ↳RandomAccessIterator2, StrictWeakOrdering)
- template<typename RandomAccessIterator1, typename RandomAccessIterator2>
  void thrust::sort_by_key(RandomAccessIterator1, RandomAccessIterator1,
  ↳RandomAccessIterator2)
```

Template Function `thrust::sort_by_key(RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2, StrictWeakOrdering)`

Function Documentation

template<typename **RandomAccessIterator1**, typename **RandomAccessIterator2**, typename **StrictWeakOrdering**>
void thrust::sort_by_key(*RandomAccessIterator1* keys_first, *RandomAccessIterator1* keys_last, *RandomAccessIterator2* values_first, *StrictWeakOrdering* comp)

`sort_by_key` performs a key-value sort. That is, `sort_by_key` sorts the elements in `[keys_first, keys_last)` and `[values_first, values_first + (keys_last - keys_first))` into ascending key order, meaning that if `i` and `j` are any two valid iterators in `[keys_first, keys_last)` such that `i` precedes `j`, and `p` and `q` are iterators in `[values_first, values_first + (keys_last - keys_first))` corresponding to `i` and `j` respectively, then `*j` is not less than `*i`.

Note: `sort_by_key` is not guaranteed to be stable. That is, suppose that `*i` and `*j` are equivalent: neither one is less than the other. It is not guaranteed that the relative order of these two keys or the relative order of their corresponding values will be preserved by `sort_by_key`.

This version of `sort_by_key` compares key objects using a function object `comp`.

The following code snippet demonstrates how to use `sort_by_key` to sort an array of character values using integers as sorting keys. The keys are sorted in descending order using the `greater<int>` comparison operator.

Pre The range `[keys_first, keys_last)` shall not overlap the range `[values_first, values_first + (keys_last - keys_first))`.

Parameters

- `keys_first`: The beginning of the key sequence.
- `keys_last`: The end of the key sequence.
- `values_first`: The beginning of the value sequence.
- `comp`: Comparison operator.

Template Parameters

- `RandomAccessIterator1`: is a model of `Random Access Iterator`, `RandomAccessIterator1` is mutable, and `RandomAccessIterator1`'s `value_type` is convertible to `StrictWeakOrdering`'s `first_argument_type` and `second_argument_type`.
- `RandomAccessIterator2`: is a model of `Random Access Iterator`, and `RandomAccessIterator2` is mutable.
- `StrictWeakOrdering`: is a model of `Strict Weak Ordering`.

```
#include <thrust/sort.h>
...
const int N = 6;
int keys[N] = { 1, 4, 2, 8, 5, 7};
char values[N] = {'a', 'b', 'c', 'd', 'e', 'f'};
thrust::sort_by_key(keys, keys + N, values, thrust::greater<int>());
// keys is now { 8, 7, 5, 4, 2, 1}
// values is now {'d', 'f', 'e', 'b', 'c', 'a'}
```

See <http://www.sgi.com/tech/stl/sort.html>

See `stable_sort_by_key`

See `sort`

Template Function `thrust::sqrt`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::sqrt(const complex < T > & z)
Returns the complex square root of a complex number.

Parameters

- `z`: The complex argument.

Template Function `thrust::stable_partition(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate)`

- Defined in `file_thrust_partition.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::stable_partition`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
  ↳ typename Predicate>__host__ __device__ ForwardIterator thrust::stable_
  ↳ partition(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ ForwardIterator, ForwardIterator, InputIterator, Predicate)
- template<typename DerivedPolicy, typename ForwardIterator, typename Predicate>__
  ↳ host__ __device__ ForwardIterator thrust::stable_partition(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator, Predicate)
- template<typename ForwardIterator, typename InputIterator, typename Predicate>
  ↳ ForwardIterator thrust::stable_partition(ForwardIterator, ForwardIterator,
  ↳ InputIterator, Predicate)
- template<typename ForwardIterator, typename Predicate>
  ↳ ForwardIterator thrust::stable_partition(ForwardIterator, ForwardIterator,
  ↳ Predicate)
```

Template Function `thrust::stable_partition(ForwardIterator, ForwardIterator, Predicate)`

Function Documentation

template<typename ForwardIterator, typename Predicate>

ForwardIterator thrust::stable_partition(*ForwardIterator first*, *ForwardIterator last*, *Predicate pred*)

stable_partition is much like partition : it reorders the elements in the range [first, last) based on the function object pred, such that all of the elements that satisfy pred precede all of the elements that fail to satisfy it. The postcondition is that, for some iterator middle in the range [first, last), pred(*i) is true for every iterator i in the range [first, middle) and false for every iterator i in the range [middle, last). The return value of stable_partition is middle.

stable_partition differs from partition in that stable_partition is guaranteed to preserve relative order. That is, if x and y are elements in [first, last), and stencil_x and stencil_y are the stencil elements in corresponding positions within [stencil, stencil + (last - first)), and pred(stencil_x) == pred(stencil_y), and if x precedes y, then it will still be true after stable_partition that x precedes y.

The following code snippet demonstrates how to use stable_partition to reorder a sequence so that even numbers precede odd numbers.

Return An iterator referring to the first element of the second partition, that is, the sequence of the elements which do not satisfy pred.

Parameters

- first: The first element of the sequence to reorder.
- last: One position past the last element of the sequence to reorder.
- pred: A function object which decides to which partition each element of the sequence [first, last) belongs.

Template Parameters

- ForwardIterator: is a model of [Forward Iterator](#), and ForwardIterator's value_type is convertible to Predicate's argument_type, and ForwardIterator is mutable.
- Predicate: is a model of [Predicate](#).

```
#include <thrust/partition.h>
...
struct is_even
{
    __host__ __device__
    bool operator() (const int &x)
    {
        return (x % 2) == 0;
    }
};
...
int A[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
const int N = sizeof(A)/sizeof(int);
thrust::stable_partition(A, A + N,
                        is_even());
// A is now {2, 4, 6, 8, 10, 1, 3, 5, 7, 9}
```

See http://www.sgi.com/tech/stl/stable_partition.html

See [partition](#)

See [stable_partition_copy](#)

Template Function `thrust::stable_partition(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, Predicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::stable_partition” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, Predicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
  ↳ typename Predicate>__host__ __device__ ForwardIterator thrust::stable_
  ↳ partition(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ ForwardIterator, ForwardIterator, InputIterator, Predicate)
- template<typename DerivedPolicy, typename ForwardIterator, typename Predicate>__
  ↳ host__ __device__ ForwardIterator thrust::stable_partition(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator, Predicate)
- template<typename ForwardIterator, typename InputIterator, typename Predicate>
  ForwardIterator thrust::stable_partition(ForwardIterator, ForwardIterator,
  ↳ InputIterator, Predicate)
- template<typename ForwardIterator, typename Predicate>
  ForwardIterator thrust::stable_partition(ForwardIterator, ForwardIterator,
  ↳ Predicate)
```

Template Function `thrust::stable_partition(ForwardIterator, ForwardIterator, InputIterator, Predicate)`

Function Documentation

template<typename **ForwardIterator**, typename **InputIterator**, typename **Predicate**>

ForwardIterator thrust::stable_partition(*ForwardIterator* first, *ForwardIterator* last, *InputIterator* stencil, *Predicate* pred)

`stable_partition` is much like `partition`: it reorders the elements in the range `[first, last)` based on the function object `pred` applied to a stencil range `[stencil, stencil + (last - first))`, such that all of the elements whose corresponding stencil element satisfies `pred` precede all of the elements whose corresponding stencil element fails to satisfy it. The postcondition is that, for some iterator `middle` in the range `[first, last)`, `pred(*stencil_i)` is true for every iterator `stencil_i` in the range `[stencil, stencil + (middle - first))` and false for every iterator `stencil_i` in the range `[stencil + (middle - first), stencil + (last - first))`. The return value of `stable_partition` is `middle`.

`stable_partition` differs from `partition` in that `stable_partition` is guaranteed to preserve relative order. That is, if `x` and `y` are elements in `[first, last)`, such that `pred(x) == pred(y)`, and if `x` precedes `y`, then it will still be true after `stable_partition` that `x` precedes `y`.

The following code snippet demonstrates how to use `stable_partition` to reorder a sequence so that even numbers precede odd numbers.

Return An iterator referring to the first element of the second partition, that is, the sequence of the elements whose stencil elements do not satisfy `pred`.

Pre The range `[first, last)` shall not overlap with the range `[stencil, stencil + (last - first))`.

Parameters

- `first`: The first element of the sequence to reorder.
- `last`: One position past the last element of the sequence to reorder.
- `stencil`: The beginning of the stencil sequence.
- `pred`: A function object which decides to which partition each element of the sequence [`first`, `last`) belongs.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator` is mutable.
- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/partition.h>
...
struct is_even
{
    __host__ __device__
    bool operator() (const int &x)
    {
        return (x % 2) == 0;
    }
};
...
int A[] = {0, 1, 0, 1, 0, 1, 0, 1, 0, 1};
int S[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
const int N = sizeof(A)/sizeof(int);
thrust::stable_partition(A, A + N, S, is_even());
// A is now {1, 1, 1, 1, 1, 0, 0, 0, 0, 0}
// S is unmodified
```

See http://www.sgi.com/tech/stl/stable_partition.html

See `partition`

See `stable_partition_copy`

Template Function `thrust::stable_partition_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)`

- Defined in `file_thrust_partition.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::stable_partition_copy`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `InputIterator`, `InputIterator`, `OutputIterator1`, `OutputIterator2`, `Predicate`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:


```

- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator1,
  ↳ typename OutputIterator2, typename Predicate>__host__ __device__ thrust::pair
  ↳<OutputIterator1,OutputIterator2> thrust::stable_partition_copy(const_
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳InputIterator, OutputIterator1, OutputIterator2, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator1, typename OutputIterator2, typename Predicate>__host__ _
  ↳_device__ thrust::pair<OutputIterator1,OutputIterator2> thrust::stable_partition_
  ↳copy(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2,
  ↳Predicate)
- template<typename InputIterator, typename OutputIterator1, typename_
  ↳OutputIterator2, typename Predicate>
  ↳thrust::pair<OutputIterator1, OutputIterator2> thrust::stable_partition_
  ↳copy(InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename_
  ↳OutputIterator1, typename OutputIterator2, typename Predicate>
  ↳thrust::pair<OutputIterator1, OutputIterator2> thrust::stable_partition_
  ↳copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
  ↳OutputIterator2, Predicate)

```

Template Function `thrust::stable_partition_copy(InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)`

Function Documentation

```

template<typename InputIterator, typename OutputIterator1, typename OutputIterator2, typename Predicate>
thrust::pair<OutputIterator1, OutputIterator2> thrust::stable_partition_copy (InputIterator
                                                                    first, InputIterator
                                                                    last, OutputIter-
                                                                    ator1 out_true,
                                                                    OutputIter-
                                                                    ator2 out_false,
                                                                    Predicate pred)

```

`stable_partition_copy` differs from `stable_partition` only in that the reordered sequence is written to different output sequences, rather than in place.

`stable_partition_copy` copies the elements `[first, last)` based on the function object `pred`. All of the elements that satisfy `pred` are copied to the range beginning at `out_true` and all the elements that fail to satisfy it are copied to the range beginning at `out_false`.

`stable_partition_copy` differs from `partition_copy` in that `stable_partition_copy` is guaranteed to preserve relative order. That is, if `x` and `y` are elements in `[first, last)`, such that `pred(x) == pred(y)`, and if `x` precedes `y`, then it will still be true after `stable_partition_copy` that `x` precedes `y` in the output.

The following code snippet demonstrates how to use `stable_partition_copy` to reorder a sequence so that even numbers precede odd numbers.

Return A pair `p` such that `p.first` is the end of the output range beginning at `out_true` and `p.second` is the end of the output range beginning at `out_false`.

Pre The input ranges shall not overlap with either output range.

Parameters

- `first`: The first element of the sequence to reorder.
- `last`: One position past the last element of the sequence to reorder.
- `out_true`: The destination of the resulting sequence of elements which satisfy `pred`.
- `out_false`: The destination of the resulting sequence of elements which fail to satisfy `pred`.
- `pred`: A function object which decides to which partition each element of the sequence `[first, last)` belongs.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `Predicate`'s `argument_type` and `InputIterator`'s `value_type` is convertible to `OutputIterator1` and `OutputIterator2`'s `value_types`.
- `OutputIterator1`: is a model of [Output Iterator](#).
- `OutputIterator2`: is a model of [Output Iterator](#).
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/partition.h>
...
struct is_even
{
    __host__ __device__
    bool operator() (const int &x)
    {
        return (x % 2) == 0;
    }
};
...
int A[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int result[10];
const int N = sizeof(A)/sizeof(int);
int *evens = result;
int *odds = result + 5;
thrust::stable_partition_copy(A, A + N, evens, odds, is_even());
// A remains {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
// result is now {2, 4, 6, 8, 10, 1, 3, 5, 7, 9}
// evens points to {2, 4, 6, 8, 10}
// odds points to {1, 3, 5, 7, 9}
```

See <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2569.pdf>

See `partition_copy`

See `stable_partition`

Template Function `thrust::stable_partition_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, Predicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::stable_partition_copy” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, Predicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator1,
    ↪ typename OutputIterator2, typename Predicate>__host__ __device__ thrust::pair
    ↪<OutputIterator1,OutputIterator2> thrust::stable_partition_copy(const_
    ↪thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
    ↪InputIterator, OutputIterator1, OutputIterator2, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename OutputIterator1, typename OutputIterator2, typename Predicate>__host__ _
    ↪_device__ thrust::pair<OutputIterator1,OutputIterator2> thrust::stable_partition_
    ↪copy(const thrust::detail::execution_policy_base< DerivedPolicy > &,
    ↪InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2,
    ↪Predicate)
- template<typename InputIterator, typename OutputIterator1, typename_
    ↪OutputIterator2, typename Predicate>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::stable_partition_
    ↪copy(InputIterator, InputIterator, OutputIterator1, OutputIterator2, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename_
    ↪OutputIterator1, typename OutputIterator2, typename Predicate>
    thrust::pair<OutputIterator1, OutputIterator2> thrust::stable_partition_
    ↪copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
    ↪OutputIterator2, Predicate)
```

Template Function thrust::stable_partition_copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, Predicate)

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator1**, typename **OutputIterator2**, typename **Predicate**>
thrust::pair<*OutputIterator1*, *OutputIterator2*> thrust::stable_partition_copy(*InputIterator1*

first, *InputIterator1* *last*,
InputIterator2
stencil, *OutputIterator1* *out_true*,
OutputIterator2 *out_false*,
Predicate pred)

stable_partition_copy differs from stable_partition only in that the reordered sequence is written to different output sequences, rather than in place.

stable_partition_copy copies the elements [*first*, *last*) based on the function object *pred* which is applied to a range of stencil elements. All of the elements whose corresponding stencil element satisfies *pred* are copied to the range beginning at *out_true* and all the elements whose stencil element fails to satisfy it are copied to the range beginning at *out_false*.

stable_partition_copy differs from partition_copy in that stable_partition_copy is guaranteed to preserve relative order. That is, if *x* and *y* are elements in [*first*, *last*), such that *pred*(*x*) == *pred*(*y*), and if *x* precedes *y*, then it will still be true after stable_partition_copy that *x* precedes *y* in the output.

The following code snippet demonstrates how to use stable_partition_copy to reorder a sequence so

that even numbers precede odd numbers.

Return A pair `p` such that `p.first` is the end of the output range beginning at `out_true` and `p.second` is the end of the output range beginning at `out_false`.

Pre The input ranges shall not overlap with either output range.

Parameters

- `first`: The first element of the sequence to reorder.
- `last`: One position past the last element of the sequence to reorder.
- `stencil`: The beginning of the stencil sequence.
- `out_true`: The destination of the resulting sequence of elements which satisfy `pred`.
- `out_false`: The destination of the resulting sequence of elements which fail to satisfy `pred`.
- `pred`: A function object which decides to which partition each element of the sequence `[first, last)` belongs.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `OutputIterator1` and `OutputIterator2`'s `value_types`.
- `InputIterator2`: is a model of [Input Iterator](#), and `InputIterator2`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `OutputIterator1`: is a model of [Output Iterator](#).
- `OutputIterator2`: is a model of [Output Iterator](#).
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/partition.h>
#include <thrust/functional.h>
...
int A[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int S[] = {0, 1, 0, 1, 0, 1, 0, 1, 0, 1};
int result[10];
const int N = sizeof(A)/sizeof(int);
int *evens = result;
int *odds = result + 5;
thrust::stable_partition_copy(A, A + N, S, evens, odds, thrust::identity<int>());
// A remains {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
// S remains {0, 1, 0, 1, 0, 1, 0, 1, 0, 1}
// result is now {2, 4, 6, 8, 10, 1, 3, 5, 7, 9}
// evens points to {2, 4, 6, 8, 10}
// odds points to {1, 3, 5, 7, 9}
```

See <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2569.pdf>

See `partition_copy`

See `stable_partition`

Template Function `thrust::stable_sort(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator, RandomAccessIterator)`

- Defined in `file_thrust_sort.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::stable_sort” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator, RandomAccessIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename RandomAccessIterator, typename
↳ StrictWeakOrdering>__host__ __device__ void thrust::stable_sort(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, RandomAccessIterator,
↳ RandomAccessIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename RandomAccessIterator>__host__ __device__
↳ void thrust::stable_sort(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, RandomAccessIterator, RandomAccessIterator)
- template<typename RandomAccessIterator, typename StrictWeakOrdering>
↳ void thrust::stable_sort(RandomAccessIterator, RandomAccessIterator,
↳ StrictWeakOrdering)
- template<typename RandomAccessIterator>
↳ void thrust::stable_sort(RandomAccessIterator, RandomAccessIterator)
```

Template Function thrust::stable_sort(RandomAccessIterator, RandomAccessIterator)

Function Documentation

template<typename **RandomAccessIterator**>

void thrust::stable_sort (*RandomAccessIterator first, RandomAccessIterator last*)

stable_sort is much like sort: it sorts the elements in [first, last) into ascending order, meaning that if i and j are any two valid iterators in [first, last) such that i precedes j, then *j is not less than *i.

As the name suggests, stable_sort is stable: it preserves the relative ordering of equivalent elements. That is, if x and y are elements in [first, last) such that x precedes y, and if the two elements are equivalent (neither x < y nor y < x) then a postcondition of stable_sort is that x still precedes y.

This version of stable_sort compares objects using operator<.

The following code snippet demonstrates how to use sort to sort a sequence of integers.

Parameters

- first: The beginning of the sequence.
- last: The end of the sequence.

Template Parameters

- RandomAccessIterator: is a model of [Random Access Iterator](#), RandomAccessIterator is mutable, and RandomAccessIterator's value_type is a model of [LessThan Comparable](#), and the ordering relation on RandomAccessIterator's value_type is a *strict weak ordering*, as defined in the [LessThan Comparable](#) requirements.

```
#include <thrust/sort.h>
...
const int N = 6;
int A[N] = {1, 4, 2, 8, 5, 7};
thrust::stable_sort(A, A + N);
// A is now {1, 2, 4, 5, 7, 8}
```

See http://www.sgi.com/tech/stl/stable_sort.html

See `sort`

See `stable_sort_by_key`

Template Function `thrust::stable_sort(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator, RandomAccessIterator, StrictWeakOrdering)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::stable_sort” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator, RandomAccessIterator, StrictWeakOrdering) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename RandomAccessIterator, typename
↳ StrictWeakOrdering>__host__ __device__ void thrust::stable_sort(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, RandomAccessIterator,
↳ RandomAccessIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename RandomAccessIterator>__host__ __device__
↳ void thrust::stable_sort(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, RandomAccessIterator, RandomAccessIterator)
- template<typename RandomAccessIterator, typename StrictWeakOrdering>
↳ void thrust::stable_sort(RandomAccessIterator, RandomAccessIterator,
↳ StrictWeakOrdering)
- template<typename RandomAccessIterator>
↳ void thrust::stable_sort(RandomAccessIterator, RandomAccessIterator)
```

Template Function `thrust::stable_sort(RandomAccessIterator, RandomAccessIterator, StrictWeakOrdering)`

Function Documentation

`template<typename RandomAccessIterator, typename StrictWeakOrdering>`

`void thrust::stable_sort(RandomAccessIterator first, RandomAccessIterator last, StrictWeakOrdering comp)`

`stable_sort` is much like `sort`: it sorts the elements in `[first, last)` into ascending order, meaning that if `i` and `j` are any two valid iterators in `[first, last)` such that `i` precedes `j`, then `*j` is not less than `*i`.

As the name suggests, `stable_sort` is stable: it preserves the relative ordering of equivalent elements. That is, if `x` and `y` are elements in `[first, last)` such that `x` precedes `y`, and if the two elements are equivalent (neither `x < y` nor `y < x`) then a postcondition of `stable_sort` is that `x` still precedes `y`.

This version of `stable_sort` compares objects using a function object `comp`.

The following code demonstrates how to sort integers in descending order using the `greater<int>` comparison operator.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.

- comp: Comparison operator.

Template Parameters

- RandomAccessIterator: is a model of [Random Access Iterator](#), RandomAccessIterator is mutable, and RandomAccessIterator's value_type is convertible to StrictWeakOrdering's first_argument_type and second_argument_type.
- StrictWeakOrdering: is a model of [Strict Weak Ordering](#).

```
#include <thrust/sort.h>
#include <thrust/functional.h>
...
const int N = 6;
int A[N] = {1, 4, 2, 8, 5, 7};
thrust::sort(A, A + N, thrust::greater<int>());
// A is now {8, 7, 5, 4, 2, 1};
```

See http://www.sgi.com/tech/stl/stable_sort.html

See `sort`

See `stable_sort_by_key`

Template Function `thrust::stable_sort_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2)`

- Defined in `file_thrust_sort.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::stable_sort_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename RandomAccessIterator1, typename_
↳RandomAccessIterator2, typename StrictWeakOrdering>__host__ __device__ void_
↳thrust::stable_sort_by_key(const thrust::detail::execution_policy_base<_
↳DerivedPolicy > &, RandomAccessIterator1, RandomAccessIterator1,_
↳RandomAccessIterator2, StrictWeakOrdering)
- template<typename DerivedPolicy, typename RandomAccessIterator1, typename_
↳RandomAccessIterator2>__host__ __device__ void thrust::stable_sort_by_key(const_
↳thrust::detail::execution_policy_base< DerivedPolicy > &, RandomAccessIterator1,_
↳RandomAccessIterator1, RandomAccessIterator2)
- template<typename RandomAccessIterator1, typename RandomAccessIterator2, typename_
↳StrictWeakOrdering>
  void thrust::stable_sort_by_key(RandomAccessIterator1, RandomAccessIterator1,_
↳RandomAccessIterator2, StrictWeakOrdering)
- template<typename RandomAccessIterator1, typename RandomAccessIterator2>
  void thrust::stable_sort_by_key(RandomAccessIterator1, RandomAccessIterator1,_
↳RandomAccessIterator2)
```

Template Function `thrust::stable_sort_by_key(RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2)`

Function Documentation

template<typename **RandomAccessIterator1**, typename **RandomAccessIterator2**>

void thrust::stable_sort_by_key(*RandomAccessIterator1 keys_first, RandomAccessIterator1 keys_last, RandomAccessIterator2 values_first*)

`stable_sort_by_key` performs a key-value sort. That is, `stable_sort_by_key` sorts the elements in `[keys_first, keys_last)` and `[values_first, values_first + (keys_last - keys_first))` into ascending key order, meaning that if `i` and `j` are any two valid iterators in `[keys_first, keys_last)` such that `i` precedes `j`, and `p` and `q` are iterators in `[values_first, values_first + (keys_last - keys_first))` corresponding to `i` and `j` respectively, then `*j` is not less than `*i`.

As the name suggests, `stable_sort_by_key` is stable: it preserves the relative ordering of equivalent elements. That is, if `x` and `y` are elements in `[keys_first, keys_last)` such that `x` precedes `y`, and if the two elements are equivalent (neither `x < y` nor `y < x`) then a postcondition of `stable_sort_by_key` is that `x` still precedes `y`.

This version of `stable_sort_by_key` compares key objects using `operator<`.

The following code snippet demonstrates how to use `stable_sort_by_key` to sort an array of characters using integers as sorting keys.

Pre The range `[keys_first, keys_last)` shall not overlap the range `[values_first, values_first + (keys_last - keys_first))`.

Parameters

- `keys_first`: The beginning of the key sequence.
- `keys_last`: The end of the key sequence.
- `values_first`: The beginning of the value sequence.

Template Parameters

- `RandomAccessIterator1`: is a model of `Random Access Iterator`, `RandomAccessIterator1` is mutable, and `RandomAccessIterator1`'s `value_type` is a model of `LessThan Comparable`, and the ordering relation on `RandomAccessIterator1`'s `value_type` is a *strict weak ordering*, as defined in the `LessThan Comparable` requirements.
- `RandomAccessIterator2`: is a model of `Random Access Iterator`, and `RandomAccessIterator2` is mutable.

```
#include <thrust/sort.h>
...
const int N = 6;
int keys[N] = { 1, 4, 2, 8, 5, 7};
char values[N] = {'a', 'b', 'c', 'd', 'e', 'f'};
thrust::stable_sort_by_key(keys, keys + N, values);
// keys is now { 1, 2, 4, 5, 7, 8}
// values is now {'a', 'c', 'b', 'e', 'f', 'd'}
```

See <http://www.sgi.com/tech/stl/sort.html>

See `sort_by_key`

See `stable_sort`

Template Function `thrust::stable_sort_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2, StrictWeakOrdering)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::stable_sort_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2, StrictWeakOrdering) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename RandomAccessIterator1, typename
↳ RandomAccessIterator2, typename StrictWeakOrdering>__host__ __device__ void
↳ thrust::stable_sort_by_key(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, RandomAccessIterator1, RandomAccessIterator1,
↳ RandomAccessIterator2, StrictWeakOrdering)
- template<typename DerivedPolicy, typename RandomAccessIterator1, typename
↳ RandomAccessIterator2>__host__ __device__ void thrust::stable_sort_by_key(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, RandomAccessIterator1,
↳ RandomAccessIterator1, RandomAccessIterator2)
- template<typename RandomAccessIterator1, typename RandomAccessIterator2, typename
↳ StrictWeakOrdering>
  void thrust::stable_sort_by_key(RandomAccessIterator1, RandomAccessIterator1,
↳ RandomAccessIterator2, StrictWeakOrdering)
- template<typename RandomAccessIterator1, typename RandomAccessIterator2>
  void thrust::stable_sort_by_key(RandomAccessIterator1, RandomAccessIterator1,
↳ RandomAccessIterator2)
```

Template Function `thrust::stable_sort_by_key(RandomAccessIterator1, RandomAccessIterator1, RandomAccessIterator2, StrictWeakOrdering)`

Function Documentation

```
template<typename RandomAccessIterator1, typename RandomAccessIterator2, typename StrictWeakOrdering>
void thrust::stable_sort_by_key(RandomAccessIterator1 keys_first, RandomAccessIterator1
keys_last, RandomAccessIterator2 values_first, StrictWeakOrdering comp)
```

`stable_sort_by_key` performs a key-value sort. That is, `stable_sort_by_key` sorts the elements in `[keys_first, keys_last)` and `[values_first, values_first + (keys_last - keys_first))` into ascending key order, meaning that if `i` and `j` are any two valid iterators in `[keys_first, keys_last)` such that `i` precedes `j`, and `p` and `q` are iterators in `[values_first, values_first + (keys_last - keys_first))` corresponding to `i` and `j` respectively, then `*j` is not less than `*i`.

As the name suggests, `stable_sort_by_key` is stable: it preserves the relative ordering of equivalent elements. That is, if `x` and `y` are elements in `[keys_first, keys_last)` such that `x` precedes `y`, and if the two elements are equivalent (neither `x < y` nor `y < x`) then a postcondition of `stable_sort_by_key` is that `x` still precedes `y`.

This version of `stable_sort_by_key` compares key objects using the function object `comp`.

The following code snippet demonstrates how to use `sort_by_key` to sort an array of character values using integers as sorting keys. The keys are sorted in descending order using the `greater<int>` comparison operator.

Pre The range `[keys_first, keys_last)` shall not overlap the range `[values_first, values_first + (keys_last - keys_first))`.

Parameters

- `keys_first`: The beginning of the key sequence.
- `keys_last`: The end of the key sequence.
- `values_first`: The beginning of the value sequence.
- `comp`: Comparison operator.

Template Parameters

- `RandomAccessIterator1`: is a model of `Random Access Iterator`, `RandomAccessIterator1` is mutable, and `RandomAccessIterator1`'s `value_type` is convertible to `StrictWeakOrdering`'s `first_argument_type` and `second_argument_type`.
- `RandomAccessIterator2`: is a model of `Random Access Iterator`, and `RandomAccessIterator2` is mutable.
- `StrictWeakOrdering`: is a model of `Strict Weak Ordering`.

```
#include <thrust/sort.h>
...
const int N = 6;
int keys[N] = { 1, 4, 2, 8, 5, 7};
char values[N] = {'a', 'b', 'c', 'd', 'e', 'f'};
thrust::stable_sort_by_key(keys, keys + N, values, thrust::greater<int>());
// keys is now { 8, 7, 5, 4, 2, 1}
// values is now {'d', 'f', 'e', 'b', 'c', 'a'}
```

See <http://www.sgi.com/tech/stl/sort.html>

See `sort_by_key`

See `stable_sort`

Template Function `thrust::swap(device_reference<T>&, device_reference<T>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::swap`” with arguments `(device_reference<T>&, device_reference<T>&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename Assignable1, typename Assignable2>__host__ __device__ void_
  ↳ thrust::swap(Assignable1 &, Assignable2 &)
- template<typename T0, typename T1, typename T2, typename T3, typename T4,
  ↳ typename T5, typename T6, typename T7, typename T8, typename T9, typename U0,
  ↳ typename U1, typename U2, typename U3, typename U4, typename U5, typename U6,
  ↳ typename U7, typename U8, typename U9>__host__ __device__ void thrust::swap(tuple
  ↳ < T0, T1, T2, T3, T4, T5, T6, T7, T8, T9 > &, tuple < U0, U1, U2, U3, U4, U5, U6,
  ↳ U7, U8, U9 > &)
- template<typename T1, typename T2>__host__ __device__ void thrust::swap(pair < T1,
  ↳ T2 > &, pair < T1, T2 > &)
- template<typename T>__host__ __device__ void thrust::swap(device_reference < T > &
  ↳ , device_reference < T > &)
```

Template Function `thrust::swap(pair<T1, T2>&, pair<T1, T2>&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::swap” with arguments (pair<T1, T2>&, pair<T1, T2>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename Assignable1, typename Assignable2>__host__ __device__ void
  ↳thrust::swap(Assignable1 &, Assignable2 &)
- template<typename T0, typename T1, typename T2, typename T3, typename T4,
  ↳typename T5, typename T6, typename T7, typename T8, typename T9, typename U0,
  ↳typename U1, typename U2, typename U3, typename U4, typename U5, typename U6,
  ↳typename U7, typename U8, typename U9>__host__ __device__ void thrust::swap(tuple
  ↳< T0, T1, T2, T3, T4, T5, T6, T7, T8, T9 > &, tuple < U0, U1, U2, U3, U4, U5, U6,
  ↳U7, U8, U9 > &)
- template<typename T1, typename T2>__host__ __device__ void thrust::swap(pair < T1,
  ↳T2 > &, pair < T1, T2 > &)
- template<typename T>__host__ __device__ void thrust::swap(device_reference < T > &
  ↳, device_reference < T > &)
```

Template Function `thrust::swap(Assignable1&, Assignable2&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::swap” with arguments (Assignable1&, Assignable2&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename Assignable1, typename Assignable2>__host__ __device__ void
  ↳thrust::swap(Assignable1 &, Assignable2 &)
- template<typename T0, typename T1, typename T2, typename T3, typename T4,
  ↳typename T5, typename T6, typename T7, typename T8, typename T9, typename U0,
  ↳typename U1, typename U2, typename U3, typename U4, typename U5, typename U6,
  ↳typename U7, typename U8, typename U9>__host__ __device__ void thrust::swap(tuple
  ↳< T0, T1, T2, T3, T4, T5, T6, T7, T8, T9 > &, tuple < U0, U1, U2, U3, U4, U5, U6,
  ↳U7, U8, U9 > &)
- template<typename T1, typename T2>__host__ __device__ void thrust::swap(pair < T1,
  ↳T2 > &, pair < T1, T2 > &)
- template<typename T>__host__ __device__ void thrust::swap(device_reference < T > &
  ↳, device_reference < T > &)
```

Template Function `thrust::swap(tuple<T0, T1, T2, T3, T4, T5, T6, T7, T8, T9>&, tuple<U0, U1, U2, U3, U4, U5, U6, U7, U8, U9>&)`

- Defined in file `_thrust_tuple.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::swap” with arguments (tuple<T0, T1, T2, T3, T4, T5, T6, T7, T8, T9>&, tuple<U0, U1, U2, U3, U4, U5, U6, U7, U8, U9>&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename Assignable1, typename Assignable2>__host__ __device__ void
→thrust::swap(Assignable1 &, Assignable2 &)
- template<typename T0, typename T1, typename T2, typename T3, typename T4,
→typename T5, typename T6, typename T7, typename T8, typename T9, typename U0,
→typename U1, typename U2, typename U3, typename U4, typename U5, typename U6,
→typename U7, typename U8, typename U9>__host__ __device__ void thrust::swap(tuple
→< T0, T1, T2, T3, T4, T5, T6, T7, T8, T9 > &, tuple < U0, U1, U2, U3, U4, U5, U6,
→U7, U8, U9 > &)
- template<typename T1, typename T2>__host__ __device__ void thrust::swap(pair < T1,
→ T2 > &, pair < T1, T2 > &)
- template<typename T>__host__ __device__ void thrust::swap(device_reference < T > &
→, device_reference < T > &)
```

Template Function thrust::swap_ranges(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator1, ForwardIterator1, ForwardIterator2)

- Defined in file_thrust_swap.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::swap_ranges” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator1, ForwardIterator1, ForwardIterator2) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator1, typename
→ForwardIterator2>__host__ __device__ ForwardIterator2 thrust::swap_ranges(const
→thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator1,
→ForwardIterator1, ForwardIterator2)
- template<typename ForwardIterator1, typename ForwardIterator2>
ForwardIterator2 thrust::swap_ranges(ForwardIterator1, ForwardIterator1,
→ForwardIterator2)
```

Template Function thrust::swap_ranges(ForwardIterator1, ForwardIterator1, ForwardIterator2)

Function Documentation

template<typename **ForwardIterator1**, typename **ForwardIterator2**>

ForwardIterator2 thrust::swap_ranges(*ForwardIterator1 first1*, *ForwardIterator1 last1*, *ForwardIterator2 first2*)

swap_ranges swaps each of the elements in the range [first1, last1) with the corresponding element in the range [first2, first2 + (last1 - first1)). That is, for each integer n such that 0 <= n < (last1 - first1), it swaps *(first1 + n) and *(first2 + n). The return value is first2 + (last1 - first1).

The following code snippet demonstrates how to use `swap_ranges` to swap the contents of two `thrust::device_vectors`.

Return An iterator pointing to one position past the last element of the second sequence to swap.

Pre `first1` may equal `first2`, but the range `[first1, last1)` shall not overlap the range `[first2, first2 + (last1 - first1))` otherwise.

Parameters

- `first1`: The beginning of the first sequence to swap.
- `last1`: One position past the last element of the first sequence to swap.
- `first2`: The beginning of the second sequence to swap.

Template Parameters

- `ForwardIterator1`: is a model of [Forward Iterator](#), and `ForwardIterator1`'s `value_type` must be convertible to `ForwardIterator2`'s `value_type`.
- `ForwardIterator2`: is a model of [Forward Iterator](#), and `ForwardIterator2`'s `value_type` must be convertible to `ForwardIterator1`'s `value_type`.

```
#include <thrust/swap.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> v1(2), v2(2);
v1[0] = 1;
v1[1] = 2;
v2[0] = 3;
v2[1] = 4;

thrust::swap_ranges(v1.begin(), v1.end(), v2.begin());

// v1[0] == 3, v1[1] == 4, v2[0] == 1, v2[1] == 2
```

See http://www.sgi.com/tech/stl/swap_ranges.html

See `swap`

Template Function `thrust::tabulate(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, UnaryOperation)`

- Defined in file `_thrust_tabulate.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::tabulate`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, UnaryOperation)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename
  ↳UnaryOperation>__host__ __device__ void thrust::tabulate(const
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ForwardIterator, UnaryOperation)
- template<typename ForwardIterator, typename UnaryOperation>
  void thrust::tabulate(ForwardIterator, ForwardIterator, UnaryOperation)
```

Template Function `thrust::tabulate(ForwardIterator, ForwardIterator, UnaryOperation)`

Function Documentation

template<typename **ForwardIterator**, typename **UnaryOperation**>

void `thrust::tabulate` (*ForwardIterator first, ForwardIterator last, UnaryOperation unary_op*)

`tabulate` fills the range `[first, last)` with the value of a function applied to each element's index.

For each iterator `i` in the range `[first, last)`, `tabulate` performs the assignment `*i = unary_op(i - first)`.

The following code snippet demonstrates how to use `tabulate` to generate the first `n` non-positive integers:

Parameters

- `first`: The beginning of the range.
- `last`: The end of the range.
- `unary_op`: The unary operation to apply.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator` is mutable, and if `x` and `y` are objects of `ForwardIterator`'s `value_type`, then `x + y` is defined, and if `T` is `ForwardIterator`'s `value_type`, then `T(0)` is defined.
- `UnaryOperation`: is a model of [Unary Function](#) and `UnaryFunction`'s `result_type` is convertible to `OutputIterator`'s `value_type`.

```
#include <thrust/tabulate.h>
#include <thrust/functional.h>
...
const int N = 10;
int A[N];
thrust::tabulate(A, A + 10, thrust::negate<int>());
// A is now {0, -1, -2, -3, -4, -5, -6, -7, -8, -9}
```

See `thrust::fill`

See `thrust::generate`

See `thrust::sequence`

Template Function `thrust::tan`

- Defined in `file_thrust_complex.h`

Function Documentation

template<typename **T**> __host__ __device__ complex<**T**> `thrust::tan`(const complex < **T** > & **z**)

Returns the complex tangent of a complex number.

Parameters

- `z`: The complex argument.

Template Function `thrust::tanh`

- Defined in file `thrust_complex.h`

Function Documentation

template<typename T>__host__ __device__ complex<T> thrust::tanh(const complex < T > & z)
Returns the complex hyperbolic tangent of a complex number.

Parameters

- `z`: The complex argument.

Template Function `thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T&, const T&, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::THRUST_PREVENT_MACRO_SUBSTITUTION`” with arguments (`const T&`, `const T&`, `BinaryPredicate`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T, typename BinaryPredicate>__host__ __device__ T max_  
→thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T &, const T &, BinaryPredicate)  
- template<typename T, typename BinaryPredicate>__host__ __device__ T min_  
→thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T &, const T &, BinaryPredicate)  
- template<typename T>__host__ __device__ T max thrust::THRUST_PREVENT_MACRO_  
→SUBSTITUTION(const T &, const T &)  
- template<typename T>__host__ __device__ T min thrust::THRUST_PREVENT_MACRO_  
→SUBSTITUTION(const T &, const T &)
```

Template Function `thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T&, const T&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::THRUST_PREVENT_MACRO_SUBSTITUTION`” with arguments (`const T&`, `const T&`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T, typename BinaryPredicate>__host__ __device__ T max_  
→thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T &, const T &, BinaryPredicate)  
- template<typename T, typename BinaryPredicate>__host__ __device__ T min_  
→thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T &, const T &, BinaryPredicate)  
- template<typename T>__host__ __device__ T max thrust::THRUST_PREVENT_MACRO_  
→SUBSTITUTION(const T &, const T &)  
- template<typename T>__host__ __device__ T min thrust::THRUST_PREVENT_MACRO_  
→SUBSTITUTION(const T &, const T &)
```

Template Function `thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T&, const T&, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::THRUST_PREVENT_MACRO_SUBSTITUTION`” with arguments `(const T&, const T&, BinaryPredicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T, typename BinaryPredicate>__host__ __device__ T max_  
→thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T &, const T &, BinaryPredicate)  
- template<typename T, typename BinaryPredicate>__host__ __device__ T min_  
→thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T &, const T &, BinaryPredicate)  
- template<typename T>__host__ __device__ T max thrust::THRUST_PREVENT_MACRO_  
→SUBSTITUTION(const T &, const T &)  
- template<typename T>__host__ __device__ T min thrust::THRUST_PREVENT_MACRO_  
→SUBSTITUTION(const T &, const T &)
```

Template Function `thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T&, const T&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::THRUST_PREVENT_MACRO_SUBSTITUTION`” with arguments `(const T&, const T&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T, typename BinaryPredicate>__host__ __device__ T max_  
→thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T &, const T &, BinaryPredicate)  
- template<typename T, typename BinaryPredicate>__host__ __device__ T min_  
→thrust::THRUST_PREVENT_MACRO_SUBSTITUTION(const T &, const T &, BinaryPredicate)  
- template<typename T>__host__ __device__ T max thrust::THRUST_PREVENT_MACRO_  
→SUBSTITUTION(const T &, const T &)  
- template<typename T>__host__ __device__ T min thrust::THRUST_PREVENT_MACRO_  
→SUBSTITUTION(const T &, const T &)
```

Template Function `thrust::tie(T0&)`

- Defined in file `_thrust_tuple.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::tie`” with arguments `(T0&)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ tuple<T0&, T1&>_  
→thrust::tie(T0 &, T1 &)  
- template<typename T0>__host__ __device__ tuple<T0&> thrust::tie(T0 &)
```


Template Function `thrust::tie(T0&, T1&)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::tie” with arguments (T0&, T1&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename T0, typename T1>__host__ __device__ tuple<T0&,T1&>
↳thrust::tie(T0 &, T1 &)
- template<typename T0>__host__ __device__ tuple<T0&> thrust::tie(T0 &)
```

Template Function `thrust::transform(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, UnaryFunction)`

- Defined in file `_thrust_transform.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::transform” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, UnaryFunction) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
↳typename UnaryFunction>__host__ __device__ OutputIterator thrust::transform(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
↳InputIterator, OutputIterator, UnaryFunction)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
↳typename OutputIterator, typename BinaryFunction>__host__ __device__
↳OutputIterator thrust::transform(const thrust::detail::execution_policy_base<
↳DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2, OutputIterator,
↳BinaryFunction)
- template<typename InputIterator, typename OutputIterator, typename UnaryFunction>
↳OutputIterator thrust::transform(InputIterator, InputIterator, OutputIterator,
↳UnaryFunction)
- template<typename InputIterator1, typename InputIterator2, typename
↳OutputIterator, typename BinaryFunction>
↳OutputIterator thrust::transform(InputIterator1, InputIterator1, InputIterator2,
↳OutputIterator, BinaryFunction)
```

Template Function `thrust::transform(InputIterator, InputIterator, OutputIterator, UnaryFunction)`

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**, typename **UnaryFunction**>
OutputIterator thrust::transform(*InputIterator first*, *InputIterator last*, *OutputIterator result*, *Unary-*
Function op)

This version of `transform` applies a unary function to each element of an input sequence and stores the result in the corresponding position in an output sequence. Specifically, for each iterator `i` in the range `[first, last)` the operation `op(*i)` is performed and the result is assigned to `*o`, where `o` is the corresponding output

iterator in the range `[result, result + (last - first))`. The input and output sequences may coincide, resulting in an in-place transformation.

The following code snippet demonstrates how to use `transform`

Return The end of the output sequence.

Pre `first` may equal `result`, but the range `[first, last)` shall not overlap the range `[result, result + (last - first))` otherwise.

Parameters

- `first`: The beginning of the input sequence.
- `last`: The end of the input sequence.
- `result`: The beginning of the output sequence.
- `op`: The transformation operation.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#) and `InputIterator`'s `value_type` is convertible to `UnaryFunction`'s `argument_type`.
- `OutputIterator`: is a model of [Output Iterator](#).
- `UnaryFunction`: is a model of [Unary Function](#) and `UnaryFunction`'s `result_type` is convertible to `OutputIterator`'s `value_type`.

```
#include <thrust/transform.h>
#include <thrust/functional.h>

int data[10] = {-5, 0, 2, -3, 2, 4, 0, -1, 2, 8};

thrust::negate<int> op;

thrust::transform(data, data + 10, data, op); // in-place transformation

// data is now {5, 0, -2, 3, -2, -4, 0, 1, -2, -8};
```

See <http://www.sgi.com/tech/stl/transform.html>

Template Function `thrust::transform(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryFunction)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::transform” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryFunction) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
  ↳ typename UnaryFunction>__host__ __device__ OutputIterator thrust::transform(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳ InputIterator, OutputIterator, UnaryFunction)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator, typename BinaryFunction>__host__ __device__
  ↳ OutputIterator thrust::transform(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2, OutputIterator,
  ↳ BinaryFunction)
- template<typename InputIterator, typename OutputIterator, typename UnaryFunction>
  ↳ OutputIterator thrust::transform(InputIterator, InputIterator, OutputIterator,
  ↳ UnaryFunction)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator, typename BinaryFunction>
  ↳ OutputIterator thrust::transform(InputIterator1, InputIterator1, InputIterator2,
  ↳ OutputIterator, BinaryFunction)

```

Template Function `thrust::transform(InputIterator1, InputIterator1, InputIterator2, OutputIterator, BinaryFunction)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator**, typename **BinaryFunction**>
OutputIterator thrust::transform(*InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, Out-*
putIterator result, BinaryFunction op)

This version of `transform` applies a binary function to each pair of elements from two input sequences and stores the result in the corresponding position in an output sequence. Specifically, for each iterator i in the range $[first1, last1)$ and $j = first + (i - first1)$ in the range $[first2, last2)$ the operation $op(*i, *j)$ is performed and the result is assigned to $*o$, where o is the corresponding output iterator in the range $[result, result + (last - first))$. The input and output sequences may coincide, resulting in an in-place transformation.

The following code snippet demonstrates how to use `transform`

Return The end of the output sequence.

Pre `first1` may equal `result`, but the range $[first1, last1)$ shall not overlap the range $[result, result + (last1 - first1))$ otherwise.

Pre `first2` may equal `result`, but the range $[first2, first2 + (last1 - first1))$ shall not overlap the range $[result, result + (last1 - first1))$ otherwise.

Parameters

- `first1`: The beginning of the first input sequence.
- `last1`: The end of the first input sequence.
- `first2`: The beginning of the second input sequence.
- `result`: The beginning of the output sequence.
- `op`: The transformation operation.

Template Parameters

- `InputIterator1`: is a model of `Input Iterator` and `InputIterator1`'s `value_type` is convertible to `BinaryFunction`'s `first_argument_type`.

- InputIterator2: is a model of [Input Iterator](#) and InputIterator2's value_type is convertible to BinaryFunction's second_argument_type.
- OutputIterator: is a model of [Output Iterator](#).
- BinaryFunction: is a model of [Binary Function](#) and BinaryFunction's result_type is convertible to OutputIterator's value_type.

```
#include <thrust/transform.h>
#include <thrust/functional.h>

int input1[6] = {-5, 0, 2, 3, 2, 4};
int input2[6] = { 3, 6, -2, 1, 2, 3};
int output[6];

thrust::plus<int> op;

thrust::transform(input1, input1 + 6, input2, output, op);

// output is now {-2, 6, 0, 4, 4, 7};
```

See <http://www.sgi.com/tech/stl/transform.html>

Template Function thrust::transform_exclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>& InputIterator, InputIterator, OutputIterator, UnaryFunction, T, AssociativeOperator)

- Defined in file_thrust_transform_scan.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::transform_exclusive_scan” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, UnaryFunction, T, AssociativeOperator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
  ↳typename UnaryFunction, typename T, typename AssociativeOperator>__host__ __
  ↳device__ OutputIterator thrust::transform_exclusive_scan(const
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳InputIterator, OutputIterator, UnaryFunction, T, AssociativeOperator)
- template<typename InputIterator, typename OutputIterator, typename UnaryFunction,
  ↳typename T, typename AssociativeOperator>
  OutputIterator thrust::transform_exclusive_scan(InputIterator, InputIterator,
  ↳OutputIterator, UnaryFunction, T, AssociativeOperator)
```

Template Function thrust::transform_exclusive_scan(InputIterator, InputIterator, OutputIterator, UnaryFunction, T, AssociativeOperator)

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**, typename **UnaryFunction**, typename **T**, typename **Assoc**

OutputIterator thrust::transform_exclusive_scan(*InputIterator* first, *InputIterator* last, *OutputIterator* result, *UnaryFunction* unary_op, *T* init, *AssociativeOperator* binary_op)

transform_exclusive_scan fuses the transform and exclusive_scan operations. transform_exclusive_scan is equivalent to performing a transformation defined by unary_op into a temporary sequence and then performing an exclusive_scan on the transformed sequence. In most cases, fusing these two operations together is more efficient, since fewer memory reads and writes are required. In transform_exclusive_scan, init is assigned to *result and the result of binary_op(init, unary_op(*first)) is assigned to *(result + 1), and so on. The transform scan operation is permitted to be in-place.

The following code snippet demonstrates how to use transform_exclusive_scan

Return The end of the output sequence.

Pre first may equal result, but the range [first, last) and the range [result, result + (last - first)) shall not overlap otherwise.

Parameters

- first: The beginning of the input sequence.
- last: The end of the input sequence.
- result: The beginning of the output sequence.
- unary_op: The function used to transform the input sequence.
- init: The initial value of the exclusive_scan
- binary_op: The associative operator used to 'sum' transformed values.

Template Parameters

- InputIterator: is a model of [Input Iterator](#) and InputIterator's value_type is convertible to unary_op's input type.
- OutputIterator: is a model of [Output Iterator](#).
- UnaryFunction: is a model of [Unary Function](#) and accepts inputs of InputIterator's value_type. UnaryFunction's result_type is convertible to OutputIterator's value_type.
- T: is convertible to OutputIterator's value_type.
- AssociativeOperator: is a model of [Binary Function](#) and AssociativeOperator's result_type is convertible to OutputIterator's value_type.

```
#include <thrust/transform_scan.h>

int data[6] = {1, 0, 2, 2, 1, 3};

thrust::negate<int> unary_op;
thrust::plus<int> binary_op;

thrust::transform_exclusive_scan(data, data + 6, data, unary_op, 4, binary_op); //
↳ in-place scan

// data is now {4, 3, 3, 1, -1, -2}
```

See transform

See exclusive_scan

Template Function `thrust::transform_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, ForwardIterator, UnaryFunction, Predicate)`

- Defined in file `_thrust_transform.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::transform_if`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, ForwardIterator, UnaryFunction, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename ForwardIterator,
  ↳ typename UnaryFunction, typename Predicate>__host__ __device__ ForwardIterator
  ↳ thrust::transform_if(const thrust::detail::execution_policy_base< DerivedPolicy >
  ↳ &, InputIterator, InputIterator, ForwardIterator, UnaryFunction, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename ForwardIterator, typename UnaryFunction, typename Predicate>__host__ __
  ↳ device__ ForwardIterator thrust::transform_if(const thrust::detail::execution_
  ↳ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ ForwardIterator, UnaryFunction, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename InputIterator3, typename ForwardIterator, typename BinaryFunction,
  ↳ typename Predicate>__host__ __device__ ForwardIterator thrust::transform_if(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, InputIterator3, ForwardIterator, BinaryFunction,
  ↳ Predicate)
- template<typename InputIterator, typename ForwardIterator, typename UnaryFunction,
  ↳ typename Predicate>
  ↳ ForwardIterator thrust::transform_if(InputIterator, InputIterator,
  ↳ ForwardIterator, UnaryFunction, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ ForwardIterator, typename UnaryFunction, typename Predicate>
  ↳ ForwardIterator thrust::transform_if(InputIterator1, InputIterator1,
  ↳ InputIterator2, ForwardIterator, UnaryFunction, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ InputIterator3, typename ForwardIterator, typename BinaryFunction, typename
  ↳ Predicate>
  ↳ ForwardIterator thrust::transform_if(InputIterator1, InputIterator1,
  ↳ InputIterator2, InputIterator3, ForwardIterator, BinaryFunction, Predicate)
```

Template Function `thrust::transform_if(InputIterator, InputIterator, ForwardIterator, UnaryFunction, Predicate)`

Function Documentation

```
template<typename InputIterator, typename ForwardIterator, typename UnaryFunction, typename Predicate>
ForwardIterator thrust::transform_if(InputIterator first, InputIterator last, ForwardIterator result,
                                   UnaryFunction op, Predicate pred)
```

This version of `transform_if` conditionally applies a unary function to each element of an input sequence and stores the result in the corresponding position in an output sequence if the corresponding position in the input sequence satisfies a predicate. Otherwise, the corresponding position in the output sequence is not modified.

Specifically, for each iterator *i* in the range `[first, last)` the predicate `pred(*i)` is evaluated. If this predicate evaluates to `true`, the result of `op(*i)` is assigned to `*o`, where *o* is the corresponding output iterator in the range `[result, result + (last - first))`. Otherwise, `op(*i)` is not evaluated and no assignment occurs. The input and output sequences may coincide, resulting in an in-place transformation.

The following code snippet demonstrates how to use `transform_if`:

Return The end of the output sequence.

Pre `first` may equal `result`, but the range `[first, last)` shall not overlap the range `[result, result + (last - first))` otherwise.

Parameters

- `first`: The beginning of the input sequence.
- `last`: The end of the input sequence.
- `result`: The beginning of the output sequence.
- `op`: The transformation operation.
- `pred`: The predicate operation.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is convertible to `Predicate`'s `argument_type`, and `InputIterator`'s `value_type` is convertible to `UnaryFunction`'s `argument_type`.
- `ForwardIterator`: is a model of [Forward Iterator](#).
- `UnaryFunction`: is a model of [Unary Function](#) and `UnaryFunction`'s `result_type` is convertible to `OutputIterator`'s `value_type`.
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/transform.h>
#include <thrust/functional.h>

int data[10] = {-5, 0, 2, -3, 2, 4, 0, -1, 2, 8};

struct is_odd
{
    __host__ __device__
    bool operator() (int x)
    {
        return x % 2;
    }
};

thrust::negate<int> op;
thrust::identity<int> identity;

// negate odd elements
thrust::transform_if(data, data + 10, data, op, is_odd()); // in-place
↳ transformation

// data is now {5, 0, 2, 3, 2, 4, 0, 1, 2, 8};
```

See `thrust::transform`

Template Function `thrust::transform_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, ForwardIterator, UnaryFunction, Predicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::transform_if`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, ForwardIterator, UnaryFunction, Predicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename ForwardIterator,
  ↳ typename UnaryFunction, typename Predicate>__host__ __device__ ForwardIterator
  ↳ thrust::transform_if(const thrust::detail::execution_policy_base< DerivedPolicy >
  ↳ &, InputIterator, InputIterator, ForwardIterator, UnaryFunction, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename ForwardIterator, typename UnaryFunction, typename Predicate>__host__ __
  ↳ device__ ForwardIterator thrust::transform_if(const thrust::detail::execution_
  ↳ policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
  ↳ ForwardIterator, UnaryFunction, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename InputIterator3, typename ForwardIterator, typename BinaryFunction,
  ↳ typename Predicate>__host__ __device__ ForwardIterator thrust::transform_if(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, InputIterator3, ForwardIterator, BinaryFunction,
  ↳ Predicate)
- template<typename InputIterator, typename ForwardIterator, typename UnaryFunction,
  ↳ typename Predicate>
  ↳ ForwardIterator thrust::transform_if(InputIterator, InputIterator,
  ↳ ForwardIterator, UnaryFunction, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ ForwardIterator, typename UnaryFunction, typename Predicate>
  ↳ ForwardIterator thrust::transform_if(InputIterator1, InputIterator1,
  ↳ InputIterator2, ForwardIterator, UnaryFunction, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ InputIterator3, typename ForwardIterator, typename BinaryFunction, typename
  ↳ Predicate>
  ↳ ForwardIterator thrust::transform_if(InputIterator1, InputIterator1,
  ↳ InputIterator2, InputIterator3, ForwardIterator, BinaryFunction, Predicate)
```

Template Function `thrust::transform_if(InputIterator1, InputIterator1, InputIterator2, ForwardIterator, UnaryFunction, Predicate)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **ForwardIterator**, typename **UnaryFunction**, typename **ForwardIterator** `thrust::transform_if` (*InputIterator1 first, InputIterator1 last, InputIterator2 stencil,*

ForwardIterator result, UnaryFunction op, Predicate pred)

This version of `transform_if` conditionally applies a unary function to each element of an input sequence and stores the result in the corresponding position in an output sequence if the corresponding position in a stencil sequence satisfies a predicate. Otherwise, the corresponding position in the output sequence is not modified.

Specifically, for each iterator `i` in the range `[first, last)` the predicate `pred(*s)` is evaluated, where `s` is the corresponding input iterator in the range `[stencil, stencil + (last - first))`. If this predicate evaluates to `true`, the result of `op(*i)` is assigned to `*o`, where `o` is the corresponding output

iterator in the range `[result, result + (last - first))`. Otherwise, `op(*i)` is not evaluated and no assignment occurs. The input and output sequences may coincide, resulting in an in-place transformation.

The following code snippet demonstrates how to use `transform_if`:

Return The end of the output sequence.

Pre `first` may equal `result`, but the range `[first, last)` shall not overlap the range `[result, result + (last - first))` otherwise.

Pre `stencil` may equal `result`, but the range `[stencil, stencil + (last - first))` shall not overlap the range `[result, result + (last - first))` otherwise.

Parameters

- `first`: The beginning of the input sequence.
- `last`: The end of the input sequence.
- `stencil`: The beginning of the stencil sequence.
- `result`: The beginning of the output sequence.
- `op`: The transformation operation.
- `pred`: The predicate operation.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#) and `InputIterator1`'s `value_type` is convertible to `UnaryFunction`'s `argument_type`.
- `InputIterator2`: is a model of [Input Iterator](#) and `InputIterator2`'s `value_type` is convertible to `Predicate`'s `argument_type`.
- `ForwardIterator`: is a model of [Forward Iterator](#).
- `UnaryFunction`: is a model of [Unary Function](#) and `UnaryFunction`'s `result_type` is convertible to `OutputIterator`'s `value_type`.
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/transform.h>
#include <thrust/functional.h>

int data[10]    = {-5, 0, 2, -3, 2, 4, 0, -1, 2, 8};
int stencil[10] = { 1, 0, 1, 0, 1, 0, 1, 0, 1, 0};

thrust::negate<int> op;
thrust::identity<int> identity;

thrust::transform_if(data, data + 10, stencil, data, op, identity); // in-place_
↪transformation

// data is now {5, 0, -2, -3, -2, 4, 0, -1, -2, 8};
```

See `thrust::transform`

Template Function `thrust::transform_if(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator3, ForwardIterator, BinaryFunction, Predicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::transform_if” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, InputIterator3, ForwardIterator, BinaryFunction, Predicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename ForwardIterator,
→ typename UnaryFunction, typename Predicate>__host__ __device__ ForwardIterator
→thrust::transform_if(const thrust::detail::execution_policy_base< DerivedPolicy >
→&, InputIterator, InputIterator, ForwardIterator, UnaryFunction, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
→ typename ForwardIterator, typename UnaryFunction, typename Predicate>__host__ __
→device__ ForwardIterator thrust::transform_if(const thrust::detail::execution_
→policy_base< DerivedPolicy > &, InputIterator1, InputIterator1, InputIterator2,
→ForwardIterator, UnaryFunction, Predicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
→ typename InputIterator3, typename ForwardIterator, typename BinaryFunction,
→typename Predicate>__host__ __device__ ForwardIterator thrust::transform_if(const
→thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
→InputIterator1, InputIterator2, InputIterator3, ForwardIterator, BinaryFunction,
→Predicate)
- template<typename InputIterator, typename ForwardIterator, typename UnaryFunction,
→ typename Predicate>
  ForwardIterator thrust::transform_if(InputIterator, InputIterator,
→ForwardIterator, UnaryFunction, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
→ForwardIterator, typename UnaryFunction, typename Predicate>
  ForwardIterator thrust::transform_if(InputIterator1, InputIterator1,
→InputIterator2, ForwardIterator, UnaryFunction, Predicate)
- template<typename InputIterator1, typename InputIterator2, typename
→InputIterator3, typename ForwardIterator, typename BinaryFunction, typename
→Predicate>
  ForwardIterator thrust::transform_if(InputIterator1, InputIterator1,
→InputIterator2, InputIterator3, ForwardIterator, BinaryFunction, Predicate)
```

Template Function thrust::transform_if(InputIterator1, InputIterator1, InputIterator2, InputIterator3, ForwardIterator, BinaryFunction, Predicate)

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename InputIterator3, typename ForwardIterator>
ForwardIterator thrust::transform_if (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2,
                                     InputIterator3 stencil, ForwardIterator result, BinaryFunction
                                     binary_op, Predicate pred)
```

This version of `transform_if` conditionally applies a binary function to each pair of elements from two input sequences and stores the result in the corresponding position in an output sequence if the corresponding position in a stencil sequence satisfies a predicate. Otherwise, the corresponding position in the output sequence is not modified.

Specifically, for each iterator i in the range $[first1, last1)$ and $j = first2 + (i - first1)$ in the range $[first2, first2 + (last1 - first1))$, the predicate `pred(*s)` is evaluated, where s is the corresponding input iterator in the range $[stencil, stencil + (last1 - first1))$. If this predicate evaluates to `true`, the result of `binary_op(*i,*j)` is assigned to `*o`, where o is the

corresponding output iterator in the range `[result, result + (last1 - first1))`. Otherwise, `binary_op(*i, *j)` is not evaluated and no assignment occurs. The input and output sequences may coincide, resulting in an in-place transformation.

The following code snippet demonstrates how to use `transform_if`:

Return The end of the output sequence.

Pre `first1` may equal `result`, but the range `[first1, last1)` shall not overlap the range `[result, result + (last1 - first1))` otherwise.

Pre `first2` may equal `result`, but the range `[first2, first2 + (last1 - first1))` shall not overlap the range `[result, result + (last1 - first1))` otherwise.

Pre `stencil` may equal `result`, but the range `[stencil, stencil + (last1 - first1))` shall not overlap the range `[result, result + (last1 - first1))` otherwise.

Parameters

- `first1`: The beginning of the first input sequence.
- `last1`: The end of the first input sequence.
- `first2`: The beginning of the second input sequence.
- `stencil`: The beginning of the stencil sequence.
- `result`: The beginning of the output sequence.
- `binary_op`: The transformation operation.
- `pred`: The predicate operation.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#) and `InputIterator1`'s `value_type` is convertible to `BinaryFunction`'s `first_argument_type`.
- `InputIterator2`: is a model of [Input Iterator](#) and `InputIterator2`'s `value_type` is convertible to `BinaryFunction`'s `second_argument_type`.
- `ForwardIterator`: is a model of [Forward Iterator](#).
- `BinaryFunction`: is a model of [Binary Function](#) and `BinaryFunction`'s `result_type` is convertible to `OutputIterator`'s `value_type`.
- `Predicate`: is a model of [Predicate](#).

```
#include <thrust/transform.h>
#include <thrust/functional.h>

int input1[6] = {-5, 0, 2, 3, 2, 4};
int input2[6] = { 3, 6, -2, 1, 2, 3};
int stencil[8] = { 1, 0, 1, 0, 1, 0};
int output[6];

thrust::plus<int> op;
thrust::identity<int> identity;

thrust::transform_if(input1, input1 + 6, input2, stencil, output, op, identity);

// output is now {-2, 0, 0, 3, 4, 4};
```

See `thrust::transform`

Template Function `thrust::transform_inclusive_scan(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, UnaryFunction, AssociativeOperator)`

- Defined in file `_thrust_transform_scan.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::transform_inclusive_scan`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `InputIterator`, `InputIterator`, `OutputIterator`, `UnaryFunction`, `AssociativeOperator`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
→typename UnaryFunction, typename AssociativeOperator>__host__ __device__
→OutputIterator thrust::transform_inclusive_scan(const thrust::detail::execution_
→policy_base< DerivedPolicy > &, InputIterator, InputIterator, OutputIterator,
→UnaryFunction, AssociativeOperator)
- template<typename InputIterator, typename OutputIterator, typename UnaryFunction,
→typename AssociativeOperator>
  OutputIterator thrust::transform_inclusive_scan(InputIterator, InputIterator,
→OutputIterator, UnaryFunction, AssociativeOperator)
```

Template Function `thrust::transform_inclusive_scan(InputIterator, InputIterator, OutputIterator, UnaryFunction, AssociativeOperator)`

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**, typename **UnaryFunction**, typename **AssociativeOperator**>
OutputIterator thrust::transform_inclusive_scan (*InputIterator* first, *InputIterator* last, *OutputIterator* result, *UnaryFunction* unary_op, *AssociativeOperator* binary_op)

`transform_inclusive_scan` fuses the `transform` and `inclusive_scan` operations. `transform_inclusive_scan` is equivalent to performing a transformation defined by `unary_op` into a temporary sequence and then performing an `inclusive_scan` on the transformed sequence. In most cases, fusing these two operations together is more efficient, since fewer memory reads and writes are required. In `transform_inclusive_scan`, `unary_op(*first)` is assigned to `*result` and the result of `binary_op(unary_op(*first), unary_op(*(first + 1)))` is assigned to `*(result + 1)`, and so on. The transform scan operation is permitted to be in-place.

The following code snippet demonstrates how to use `transform_inclusive_scan`

Return The end of the output sequence.

Pre `first` may equal `result`, but the range `[first, last)` and the range `[result, result + (last - first))` shall not overlap otherwise.

Parameters

- `first`: The beginning of the input sequence.
- `last`: The end of the input sequence.
- `result`: The beginning of the output sequence.
- `unary_op`: The function used to transform the input sequence.

- `binary_op`: The associative operator used to ‘sum’ transformed values.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#) and `InputIterator`'s `value_type` is convertible to `unary_op`'s input type.
- `OutputIterator`: is a model of [Output Iterator](#).
- `UnaryFunction`: is a model of [Unary Function](#) and accepts inputs of `InputIterator`'s `value_type`. `UnaryFunction`'s `result_type` is convertible to `OutputIterator`'s `value_type`.
- `AssociativeOperator`: is a model of [Binary Function](#) and `AssociativeOperator`'s `result_type` is convertible to `OutputIterator`'s `value_type`.

```
#include <thrust/transform_scan.h>

int data[6] = {1, 0, 2, 2, 1, 3};

thrust::negate<int> unary_op;
thrust::plus<int> binary_op;

thrust::transform_inclusive_scan(data, data + 6, data, unary_op, binary_op); //
↳ in-place scan

// data is now {-1, -1, -3, -5, -6, -9}
```

See `transform`

See `inclusive_scan`

Template Function `thrust::transform_reduce(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, UnaryFunction, OutputType, BinaryFunction)`

- Defined in `file_thrust_transform_reduce.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::transform_reduce`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `InputIterator`, `InputIterator`, `UnaryFunction`, `OutputType`, `BinaryFunction`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename UnaryFunction,
↳ typename OutputType, typename BinaryFunction>__host__ __device__ OutputType
↳ thrust::transform_reduce(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, InputIterator, InputIterator, UnaryFunction, OutputType,
↳ BinaryFunction)
- template<typename InputIterator, typename UnaryFunction, typename OutputType,
↳ typename BinaryFunction>
  OutputType thrust::transform_reduce(InputIterator, InputIterator, UnaryFunction,
↳ OutputType, BinaryFunction)
```

Template Function `thrust::transform_reduce(InputIterator, InputIterator, UnaryFunction, OutputType, BinaryFunction)`

Function Documentation

```
template<typename InputIterator, typename UnaryFunction, typename OutputType, typename BinaryFunction>
OutputType thrust::transform_reduce(InputIterator first, InputIterator last, UnaryFunction
                                   unary_op, OutputType init, BinaryFunction binary_op)
```

`transform_reduce` fuses the transform and reduce operations. `transform_reduce` is equivalent to performing a transformation defined by `unary_op` into a temporary sequence and then performing reduce on the transformed sequence. In most cases, fusing these two operations together is more efficient, since fewer memory reads and writes are required.

`transform_reduce` performs a reduction on the transformation of the sequence `[first, last)` according to `unary_op`. Specifically, `unary_op` is applied to each element of the sequence and then the result is reduced to a single value with `binary_op` using the initial value `init`. Note that the transformation `unary_op` is not applied to the initial value `init`. The order of reduction is not specified, so `binary_op` must be both commutative and associative.

The following code snippet demonstrates how to use `transform_reduce` to compute the maximum value of the absolute value of the elements of a range.

Return The result of the transformed reduction.

Parameters

- `first`: The beginning of the sequence.
- `last`: The end of the sequence.
- `unary_op`: The function to apply to each element of the input sequence.
- `init`: The result is initialized to this value.
- `binary_op`: The reduction operation.

Template Parameters

- `InputIterator`: is a model of `Input Iterator`, and `InputIterator`'s `value_type` is convertible to `UnaryFunction`'s `argument_type`.
- `UnaryFunction`: is a model of `Unary Function`, and `UnaryFunction`'s `result_type` is convertible to `OutputType`.
- `OutputType`: is a model of `Assignable`, and is convertible to `BinaryFunction`'s `first_argument_type` and `second_argument_type`.
- `BinaryFunction`: is a model of `Binary Function`, and `BinaryFunction`'s `result_type` is convertible to `OutputType`.

```
#include <thrust/transform_reduce.h>
#include <thrust/functional.h>

template<typename T>
struct absolute_value : public unary_function<T,T>
{
    __host__ __device__ T operator()(const T &x) const
    {
        return x < T(0) ? -x : x;
    }
};
```

(continues on next page)

(continued from previous page)

```
...

int data[6] = {-1, 0, -2, -2, 1, -3};
int result = thrust::transform_reduce(data, data + 6,
                                     absolute_value<int>(),
                                     0,
                                     thrust::maximum<int>());

// result == 3
```

See `transform`

See `reduce`

Template Function `thrust::uninitialized_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, ForwardIterator)`

- Defined in file `thrust_uninitialized_copy.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::uninitialized_copy” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, ForwardIterator) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename ForwardIterator>
  ↳ __host__ __device__ ForwardIterator thrust::uninitialized_copy(const_
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
  ↳ InputIterator, ForwardIterator)
- template<typename InputIterator, typename ForwardIterator>
  ForwardIterator thrust::uninitialized_copy(InputIterator, InputIterator,
  ↳ ForwardIterator)
```

Template Function `thrust::uninitialized_copy(InputIterator, InputIterator, ForwardIterator)`

Function Documentation

`template<typename InputIterator, typename ForwardIterator>`

`ForwardIterator thrust::uninitialized_copy (InputIterator first, InputIterator last, ForwardIterator result)`

In thrust, the function `thrust::device_new` allocates memory for an object and then creates an object at that location by calling a constructor. Occasionally, however, it is useful to separate those two operations. If each iterator in the range `[result, result + (last - first))` points to uninitialized memory, then `uninitialized_copy` creates a copy of `[first, last)` in that range. That is, for each iterator `i` in the input, `uninitialized_copy` creates a copy of `*i` in the location pointed to by the corresponding iterator in the output range by `ForwardIterator`'s `value_type`'s copy constructor with `*i` as its argument.

The following code snippet demonstrates how to use `uninitialized_copy` to initialize a range of uninitialized memory.

Return An iterator pointing to the last element of the output range.

Pre first may equal result, but the range [first, last) and the range [result, result + (last - first)) shall not overlap otherwise.

Parameters

- first: The first element of the input range to copy from.
- last: The last element of the input range to copy from.
- result: The first element of the output range to copy to.

Template Parameters

- InputIterator: is a model of [Input Iterator](#).
- ForwardIterator: is a model of [Forward Iterator](#), ForwardIterator is mutable, and ForwardIterator's value_type has a constructor that takes a single argument whose type is InputIterator's value_type.

```
#include <thrust/uninitialized_copy.h>
#include <thrust/device_malloc.h>
#include <thrust/device_vector.h>

struct Int
{
    __host__ __device__
    Int(int x) : val(x) {}
    int val;
};

...
const int N = 137;

Int val(46);
thrust::device_vector<Int> input(N, val);
thrust::device_ptr<Int> array = thrust::device_malloc<Int>(N);
thrust::uninitialized_copy(input.begin(), input.end(), array);

// Int x = array[i];
// x.val == 46 for all 0 <= i < N
```

See http://www.sgi.com/tech/stl/uninitialized_copy.html

See [copy](#)

See [uninitialized_fill](#)

See [device_new](#)

See [device_malloc](#)

Template Function `thrust::uninitialized_copy_n(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, Size, ForwardIterator)`

- Defined in file `thrust_uninitialized_copy.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::uninitialized_copy_n” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, Size, ForwardIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename Size, typename
↪ForwardIterator>__host__ __device__ ForwardIterator thrust::uninitialized_copy_
↪n(const thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
↪Size, ForwardIterator)
- template<typename InputIterator, typename Size, typename ForwardIterator>
  ForwardIterator thrust::uninitialized_copy_n(InputIterator, Size, ForwardIterator)
```

Template Function thrust::uninitialized_copy_n(InputIterator, Size, ForwardIterator)

Function Documentation

template<typename **InputIterator**, typename **Size**, typename **ForwardIterator**>

ForwardIterator thrust::uninitialized_copy_n(*InputIterator first*, *Size n*, *ForwardIterator result*)

In thrust, the function `thrust::device_new` allocates memory for an object and then creates an object at that location by calling a constructor. Occasionally, however, it is useful to separate those two operations. If each iterator in the range `[result, result + n)` points to uninitialized memory, then `uninitialized_copy_n` creates a copy of `[first, first + n)` in that range. That is, for each iterator `i` in the input, `uninitialized_copy_n` creates a copy of `*i` in the location pointed to by the corresponding iterator in the output range by `InputIterator`'s `value_type`'s copy constructor with `*i` as its argument.

The following code snippet demonstrates how to use `uninitialized_copy` to initialize a range of uninitialized memory.

Return An iterator pointing to the last element of the output range.

Pre `first` may equal `result`, but the range `[first, first + n)` and the range `[result, result + n)` shall not overlap otherwise.

Parameters

- `first`: The first element of the input range to copy from.
- `n`: The number of elements to copy.
- `result`: The first element of the output range to copy to.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#).
- `Size`: is an integral type.
- `ForwardIterator`: is a model of [Forward Iterator](#), `ForwardIterator` is mutable, and `ForwardIterator`'s `value_type` has a constructor that takes a single argument whose type is `InputIterator`'s `value_type`.

```
#include <thrust/uninitialized_copy.h>
#include <thrust/device_malloc.h>
#include <thrust/device_vector.h>
```

(continues on next page)

(continued from previous page)

```

struct Int
{
    __host__ __device__
    Int(int x) : val(x) {}
    int val;
};
...
const int N = 137;

Int val(46);
thrust::device_vector<Int> input(N, val);
thrust::device_ptr<Int> array = thrust::device_malloc<Int>(N);
thrust::uninitialized_copy_n(input.begin(), N, array);

// Int x = array[i];
// x.val == 46 for all 0 <= i < N

```

See http://www.sgi.com/tech/stl/uninitialized_copy.html

See `uninitialized_copy`

See `copy`

See `uninitialized_fill`

See `device_new`

See `device_malloc`

Template Function `thrust::uninitialized_fill(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&)`

- Defined in `file_thrust_uninitialized_fill.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::uninitialized_fill” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename DerivedPolicy, typename ForwardIterator, typename T>__host__ __
  ↪device__ void thrust::uninitialized_fill(const thrust::detail::execution_policy_
  ↪base< DerivedPolicy > &, ForwardIterator, ForwardIterator, const T &)
- template<typename ForwardIterator, typename T>
  void thrust::uninitialized_fill(ForwardIterator, ForwardIterator, const T&)

```

Template Function `thrust::uninitialized_fill(ForwardIterator, ForwardIterator, const T&)`

Function Documentation

`template<typename ForwardIterator, typename T>`

void thrust::uninitialized_fill (ForwardIterator first, ForwardIterator last, const T &x)

In thrust, the function `thrust::device_new` allocates memory for an object and then creates an object at that location by calling a constructor. Occasionally, however, it is useful to separate those two operations. If each iterator in the range `[first, last)` points to uninitialized memory, then `uninitialized_fill` creates copies of `x` in that range. That is, for each iterator `i` in the range `[first, last)`, `uninitialized_fill` creates a copy of `x` in the location pointed to `i` by calling `ForwardIterator`'s `value_type`'s copy constructor.

The following code snippet demonstrates how to use `uninitialized_fill` to initialize a range of uninitialized memory.

Parameters

- `first`: The first element of the range of interest.
- `last`: The last element of the range of interest.
- `x`: The value to use as the exemplar of the copy constructor.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), `ForwardIterator` is mutable, and `ForwardIterator`'s `value_type` has a constructor that takes a single argument of type `T`.

```
#include <thrust/uninitialized_fill.h>
#include <thrust/device_malloc.h>

struct Int
{
    __host__ __device__
    Int(int x) : val(x) {}
    int val;
};
...
const int N = 137;

Int val(46);
thrust::device_ptr<Int> array = thrust::device_malloc<Int>(N);
thrust::uninitialized_fill(array, array + N, val);

// Int x = array[i];
// x.val == 46 for all 0 <= i < N
```

See http://www.sgi.com/tech/stl/uninitialized_fill.html

See `uninitialized_fill_n`

See `fill`

See `uninitialized_copy`

See `device_new`

See `device_malloc`

Template Function `thrust::uninitialized_fill_n(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, Size, const T&)`

- Defined in `file_thrust_uninitialized_fill.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::uninitialized_fill_n” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, Size, const T&) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename Size,
↳typename T>__host__ __device__ ForwardIterator thrust::uninitialized_fill_n(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator, Size,
↳const T &)
- template<typename ForwardIterator, typename Size, typename T>
ForwardIterator thrust::uninitialized_fill_n(ForwardIterator, Size, const T&)
```

Template Function thrust::uninitialized_fill_n(ForwardIterator, Size, const T&)

Function Documentation

template<typename **ForwardIterator**, typename **Size**, typename **T**>

ForwardIterator thrust::uninitialized_fill_n(*ForwardIterator* first, *Size* n, const *T* &x)

In thrust, the function *thrust::device_new* allocates memory for an object and then creates an object at that location by calling a constructor. Occasionally, however, it is useful to separate those two operations. If each iterator in the range [first, first+n) points to uninitialized memory, then *uninitialized_fill* creates copies of *x* in that range. That is, for each iterator *i* in the range [first, first+n), *uninitialized_fill* creates a copy of *x* in the location pointed to *i* by calling *ForwardIterator*'s *value_type*'s copy constructor.

The following code snippet demonstrates how to use *uninitialized_fill* to initialize a range of uninitialized memory.

Return first+n

Parameters

- first: The first element of the range of interest.
- n: The size of the range of interest.
- x: The value to use as the exemplar of the copy constructor.

Template Parameters

- *ForwardIterator*: is a model of *Forward Iterator*, *ForwardIterator* is mutable, and *ForwardIterator*'s *value_type* has a constructor that takes a single argument of type *T*.

```
#include <thrust/uninitialized_fill.h>
#include <thrust/device_malloc.h>

struct Int
{
    __host__ __device__
    Int(int x) : val(x) {}
    int val;
};
...
const int N = 137;
```

(continues on next page)

(continued from previous page)

```

Int val(46);
thrust::device_ptr<Int> array = thrust::device_malloc<Int>(N);
thrust::uninitialized_fill_n(array, N, val);

// Int x = array[i];
// x.val == 46 for all 0 <= i < N

```

See http://www.sgi.com/tech/stl/uninitialized_fill.html

See `uninitialized_fill`

See `fill`

See `uninitialized_copy_n`

See `device_new`

See `device_malloc`

Template Function `thrust::unique(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator)`

- Defined in file `thrust_unique.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::unique” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<typename DerivedPolicy, typename ForwardIterator, typename
  ↳ BinaryPredicate>__host__ __device__ ForwardIterator thrust::unique(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
  ↳ ForwardIterator thrust::unique(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, ForwardIterator, ForwardIterator)
- template<typename ForwardIterator, typename BinaryPredicate>
  ↳ ForwardIterator thrust::unique(ForwardIterator, ForwardIterator, BinaryPredicate)
- template<typename ForwardIterator>
  ↳ ForwardIterator thrust::unique(ForwardIterator, ForwardIterator)

```

Template Function `thrust::unique(ForwardIterator, ForwardIterator)`

Function Documentation

template<typename **ForwardIterator**>

ForwardIterator `thrust::unique` (*ForwardIterator first*, *ForwardIterator last*)

For each group of consecutive elements in the range `[first, last)` with the same value, `unique` removes all but the first element of the group. The return value is an iterator `new_last` such that no two consecutive elements in the range `[first, new_last)` are equal. The iterators in the range `[new_last, last)` are

all still dereferenceable, but the elements that they point to are unspecified. `unique` is stable, meaning that the relative order of elements that are not removed is unchanged.

This version of `unique` uses `operator==` to test for equality.

The following code snippet demonstrates how to use `unique` to compact a sequence of numbers to remove consecutive duplicates.

Return The end of the unique range `[first, new_last)`.

Parameters

- `first`: The beginning of the input range.
- `last`: The end of the input range.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#), and `ForwardIterator` is mutable, and `ForwardIterator`'s `value_type` is a model of [Equality Comparable](#).

```
#include <thrust/unique.h>
...
const int N = 7;
int A[N] = {1, 3, 3, 3, 2, 2, 1};
int *new_end = thrust::unique(A, A + N);
// The first four values of A are now {1, 3, 2, 1}
// Values beyond new_end are unspecified.
```

See <http://www.sgi.com/tech/stl/unique.html>

See `unique_copy`

Template Function `thrust::unique(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::unique`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, BinaryPredicate)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator, typename
  ↳ BinaryPredicate>__host__ __device__ ForwardIterator thrust::unique(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename ForwardIterator>__host__ __device__
  ↳ ForwardIterator thrust::unique(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, ForwardIterator, ForwardIterator)
- template<typename ForwardIterator, typename BinaryPredicate>
  ↳ ForwardIterator thrust::unique(ForwardIterator, ForwardIterator, BinaryPredicate)
- template<typename ForwardIterator>
  ↳ ForwardIterator thrust::unique(ForwardIterator, ForwardIterator)
```

Template Function `thrust::unique(ForwardIterator, ForwardIterator, BinaryPredicate)`

Function Documentation

template<typename **ForwardIterator**, typename **BinaryPredicate**>

ForwardIterator thrust::unique(*ForwardIterator* first, *ForwardIterator* last, *BinaryPredicate* binary_pred)

For each group of consecutive elements in the range [first, last) with the same value, unique removes all but the first element of the group. The return value is an iterator new_last such that no two consecutive elements in the range [first, new_last) are equal. The iterators in the range [new_last, last) are all still dereferenceable, but the elements that they point to are unspecified. unique is stable, meaning that the relative order of elements that are not removed is unchanged.

This version of unique uses the function object binary_pred to test for equality.

The following code snippet demonstrates how to use unique to compact a sequence of numbers to remove consecutive duplicates.

Return The end of the unique range [first, new_last)

Parameters

- first: The beginning of the input range.
- last: The end of the input range.
- binary_pred: The binary predicate used to determine equality.

Template Parameters

- ForwardIterator: is a model of [Forward Iterator](#), and ForwardIterator is mutable, and ForwardIterator's value_type is convertible to BinaryPredicate's first_argument_type and to BinaryPredicate's second_argument_type.
- BinaryPredicate: is a model of [Binary Predicate](#).

```
#include <thrust/unique.h>
...
const int N = 7;
int A[N] = {1, 3, 3, 3, 2, 2, 1};
int *new_end = thrust::unique(A, A + N, thrust::equal_to<int>());
// The first four values of A are now {1, 3, 2, 1}
// Values beyond new_end are unspecified.
```

See <http://www.sgi.com/tech/stl/unique.html>

See unique_copy

Template Function thrust::unique_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator1, ForwardIterator1, ForwardIterator2)

- Defined in file_thrust_unique.h

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::unique_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator1, ForwardIterator1, ForwardIterator2) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<typename DerivedPolicy, typename ForwardIterator1, typename
↳ForwardIterator2, typename BinaryPredicate>__host__ __device__ thrust::pair
↳<ForwardIterator1,ForwardIterator2> thrust::unique_by_key(const
↳thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator1,
↳ForwardIterator1, ForwardIterator2, BinaryPredicate)
- template<typename DerivedPolicy, typename ForwardIterator1, typename
↳ForwardIterator2>__host__ __device__ thrust::pair<ForwardIterator1,
↳ForwardIterator2> thrust::unique_by_key(const thrust::detail::execution_policy_
↳base< DerivedPolicy > &, ForwardIterator1, ForwardIterator1, ForwardIterator2)
- template<typename ForwardIterator1, typename ForwardIterator2, typename
↳BinaryPredicate>
  thrust::pair<ForwardIterator1, ForwardIterator2> thrust::unique_by_
↳key(ForwardIterator1, ForwardIterator1, ForwardIterator2, BinaryPredicate)
- template<typename ForwardIterator1, typename ForwardIterator2>
  thrust::pair<ForwardIterator1, ForwardIterator2> thrust::unique_by_
↳key(ForwardIterator1, ForwardIterator1, ForwardIterator2)

```

Template Function `thrust::unique_by_key(ForwardIterator1, ForwardIterator1, ForwardIterator2)`

Function Documentation

template<typename **ForwardIterator1**, typename **ForwardIterator2**>

thrust::pair<*ForwardIterator1*, *ForwardIterator2*> thrust::unique_by_key(*ForwardIterator1*
keys_first, *ForwardIterator1*
keys_last, *ForwardIterator2*
values_first)

`unique_by_key` is a generalization of `unique` to key-value pairs. For each group of consecutive keys in the range `[keys_first, keys_last)` that are equal, `unique_by_key` removes all but the first element of the group. Similarly, the corresponding values in the range `[values_first, values_first + (keys_last - keys_first))` are also removed.

The return value is a pair of iterators (`new_keys_last`, `new_values_last`) such that no two consecutive elements in the range `[keys_first, new_keys_last)` are equal.

This version of `unique_by_key` uses `operator==` to test for equality and *project1st* to reduce values with equal keys.

The following code snippet demonstrates how to use `unique_by_key` to compact a sequence of key/value pairs to remove consecutive duplicates.

Return A pair of iterators at end of the ranges `[key_first, keys_new_last)` and `[values_first, values_new_last)`.

Pre The range `[keys_first, keys_last)` and the range `[values_first, values_first + (keys_last - keys_first))` shall not overlap.

Parameters

- `keys_first`: The beginning of the key range.
- `keys_last`: The end of the key range.
- `values_first`: The beginning of the value range.

Template Parameters

- `ForwardIterator1`: is a model of [Forward Iterator](#), and `ForwardIterator1` is mutable, and `ForwardIterator`'s `value_type` is a model of [Equality Comparable](#).

- ForwardIterator2: is a model of [Forward Iterator](#), and ForwardIterator2 is mutable.

```
#include <thrust/unique.h>
...
const int N = 7;
int A[N] = {1, 3, 3, 3, 2, 2, 1}; // keys
int B[N] = {9, 8, 7, 6, 5, 4, 3}; // values

thrust::pair<int*,int*> new_end;
new_end = thrust::unique_by_key(A, A + N, B);

// The first four keys in A are now {1, 3, 2, 1} and new_end.first - A is 4.
// The first four values in B are now {9, 8, 5, 3} and new_end.second - B is 4.
```

See [unique](#)

See [unique_by_key_copy](#)

See [reduce_by_key](#)

Template Function `thrust::unique_by_key(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator1, ForwardIterator1, ForwardIterator2, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::unique_by_key” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator1, ForwardIterator1, ForwardIterator2, BinaryPredicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename ForwardIterator1, typename
  ↳ForwardIterator2, typename BinaryPredicate>__host__ __device__ thrust::pair
  ↳<ForwardIterator1,ForwardIterator2> thrust::unique_by_key(const
  ↳thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator1,
  ↳ForwardIterator1, ForwardIterator2, BinaryPredicate)
- template<typename DerivedPolicy, typename ForwardIterator1, typename
  ↳ForwardIterator2>__host__ __device__ thrust::pair<ForwardIterator1,
  ↳ForwardIterator2> thrust::unique_by_key(const thrust::detail::execution_policy_
  ↳base< DerivedPolicy > &, ForwardIterator1, ForwardIterator1, ForwardIterator2)
- template<typename ForwardIterator1, typename ForwardIterator2, typename
  ↳BinaryPredicate>
  thrust::pair<ForwardIterator1, ForwardIterator2> thrust::unique_by_
  ↳key(ForwardIterator1, ForwardIterator1, ForwardIterator2, BinaryPredicate)
- template<typename ForwardIterator1, typename ForwardIterator2>
  thrust::pair<ForwardIterator1, ForwardIterator2> thrust::unique_by_
  ↳key(ForwardIterator1, ForwardIterator1, ForwardIterator2)
```

Template Function `thrust::unique_by_key(ForwardIterator1, ForwardIterator1, ForwardIterator2, BinaryPredicate)`

Function Documentation

template<typename **ForwardIterator1**, typename **ForwardIterator2**, typename **BinaryPredicate**>

```
thrust::pair<ForwardIterator1, ForwardIterator2> thrust::unique_by_key(ForwardIterator1
                                                                    keys_first, ForwardIt-
                                                                    erator1 keys_last, For-
                                                                    wardIterator2 values_first,
                                                                    BinaryPredicate      bi-
                                                                    nary_pred)
```

`unique_by_key` is a generalization of `unique` to key-value pairs. For each group of consecutive keys in the range `[keys_first, keys_last)` that are equal, `unique_by_key` removes all but the first element of the group. Similarly, the corresponding values in the range `[values_first, values_first + (keys_last - keys_first))` are also removed.

This version of `unique_by_key` uses the function object `binary_pred` to test for equality and *project1st* to reduce values with equal keys.

The following code snippet demonstrates how to use `unique_by_key` to compact a sequence of key/value pairs to remove consecutive duplicates.

Return The end of the unique range `[first, new_last)`.

Pre The range `[keys_first, keys_last)` and the range `[values_first, values_first + (keys_last - keys_first))` shall not overlap.

Parameters

- `keys_first`: The beginning of the key range.
- `keys_last`: The end of the key range.
- `values_first`: The beginning of the value range.
- `binary_pred`: The binary predicate used to determine equality.

Template Parameters

- `ForwardIterator1`: is a model of [Forward Iterator](#), and `ForwardIterator1` is mutable, and `ForwardIterator`'s `value_type` is a model of [Equality Comparable](#).
- `ForwardIterator2`: is a model of [Forward Iterator](#), and `ForwardIterator2` is mutable.
- `BinaryPredicate`: is a model of [Binary Predicate](#).

```
#include <thrust/unique.h>
...
const int N = 7;
int A[N] = {1, 3, 3, 3, 2, 2, 1}; // keys
int B[N] = {9, 8, 7, 6, 5, 4, 3}; // values

thrust::pair<int*, int*> new_end;
thrust::equal_to<int> binary_pred;
new_end = thrust::unique_by_key(keys, keys + N, values, binary_pred);

// The first four keys in A are now {1, 3, 2, 1} and new_end.first - A is 4.
// The first four values in B are now {9, 8, 5, 3} and new_end.second - B is 4.
```

See `unique`

See `unique_by_key_copy`

See `reduce_by_key`

Template Function `thrust::unique_by_key_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)`

- Defined in file `_thrust_unique.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::unique_by_key_copy`” with arguments `(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)` in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename OutputIterator1, typename OutputIterator2, typename BinaryPredicate>__
    ↪host__ __device__ thrust::pair<OutputIterator1,OutputIterator2> thrust::unique_by_
    ↪key_copy(const thrust::detail::execution_policy_base< DerivedPolicy > &,
    ↪InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2,
    ↪BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
    ↪ typename OutputIterator1, typename OutputIterator2>__host__ __device__
    ↪thrust::pair<OutputIterator1,OutputIterator2> thrust::unique_by_key_copy(const
    ↪thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
    ↪InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪OutputIterator1, typename OutputIterator2, typename BinaryPredicate>
    ↪thrust::pair<OutputIterator1, OutputIterator2> thrust::unique_by_key_
    ↪copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
    ↪OutputIterator2, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2, typename
    ↪OutputIterator1, typename OutputIterator2>
    ↪thrust::pair<OutputIterator1, OutputIterator2> thrust::unique_by_key_
    ↪copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
    ↪OutputIterator2)
```

Template Function `thrust::unique_by_key_copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)`

Function Documentation

```
template<typename InputIterator1, typename InputIterator2, typename OutputIterator1, typename OutputIterator2>
thrust::pair<OutputIterator1, OutputIterator2> thrust::unique_by_key_copy (InputIterator1
                                                                    keys_first,    InputIt-
                                                                    erator1    keys_last,
                                                                    InputIterator2    val-
                                                                    ues_first,    OutputIt-
                                                                    erator1    keys_result,
                                                                    OutputIterator2    val-
                                                                    ues_result)
```

`unique_by_key_copy` is a generalization of `unique_copy` to key-value pairs. For each group of consecutive keys in the range `[keys_first, keys_last)` that are equal, `unique_by_key_copy` copies the first element of the group to a range beginning with `keys_result` and the corresponding values from the

range `[values_first, values_first + (keys_last - keys_first))` are copied to a range beginning with `values_result`.

This version of `unique_by_key_copy` uses `operator==` to test for equality and *project1st* to reduce values with equal keys.

The following code snippet demonstrates how to use `unique_by_key_copy` to compact a sequence of key/value pairs and with equal keys.

Return A pair of iterators at end of the ranges `[keys_result, keys_result_last)` and `[values_result, values_result_last)`.

Pre The input ranges shall not overlap either output range.

Parameters

- `keys_first`: The beginning of the input key range.
- `keys_last`: The end of the input key range.
- `values_first`: The beginning of the input value range.
- `keys_result`: The beginning of the output key range.
- `values_result`: The beginning of the output value range.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#),
- `InputIterator2`: is a model of [Input Iterator](#),
- `OutputIterator1`: is a model of [Output Iterator](#) and `InputIterator1`'s `value_type` is convertible to `OutputIterator1`'s `value_type`.
- `OutputIterator2`: is a model of [Output Iterator](#) and `InputIterator2`'s `value_type` is convertible to `OutputIterator2`'s `value_type`.

```
#include <thrust/unique.h>
...
const int N = 7;
int A[N] = {1, 3, 3, 3, 2, 2, 1}; // input keys
int B[N] = {9, 8, 7, 6, 5, 4, 3}; // input values
int C[N]; // output keys
int D[N]; // output values

thrust::pair<int*, int*> new_end;
new_end = thrust::unique_by_key_copy(A, A + N, B, C, D);

// The first four keys in C are now {1, 3, 2, 1} and new_end.first - C is 4.
// The first four values in D are now {9, 8, 5, 3} and new_end.second - D is 4.
```

See `unique_copy`

See `unique_by_key`

See `reduce_by_key`

Template Function `thrust::unique_by_key_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::unique_by_key_copy” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator1, typename OutputIterator2, typename BinaryPredicate>__
  ↳ host__ __device__ thrust::pair<OutputIterator1,OutputIterator2> thrust::unique_by_
  ↳ key_copy(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2,
  ↳ BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator1, typename InputIterator2,
  ↳ typename OutputIterator1, typename OutputIterator2>__host__ __device__
  ↳ thrust::pair<OutputIterator1,OutputIterator2> thrust::unique_by_key_copy(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator1,
  ↳ InputIterator1, InputIterator2, OutputIterator1, OutputIterator2)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator1, typename OutputIterator2, typename BinaryPredicate>
  ↳ thrust::pair<OutputIterator1, OutputIterator2> thrust::unique_by_key_
  ↳ copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
  ↳ OutputIterator2, BinaryPredicate)
- template<typename InputIterator1, typename InputIterator2, typename
  ↳ OutputIterator1, typename OutputIterator2>
  ↳ thrust::pair<OutputIterator1, OutputIterator2> thrust::unique_by_key_
  ↳ copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1,
  ↳ OutputIterator2)
```

Template Function `thrust::unique_by_key_copy(InputIterator1, InputIterator1, InputIterator2, OutputIterator1, OutputIterator2, BinaryPredicate)`

Function Documentation

template<typename **InputIterator1**, typename **InputIterator2**, typename **OutputIterator1**, typename **OutputIterator2**, typename **BinaryPredicate**>
 thrust::pair<*OutputIterator1*, *OutputIterator2*> thrust::unique_by_key_copy(*InputIterator1*

keys_first, *InputIterator1* *keys_last*, *InputIterator2* *values_first*,
OutputIterator1 *keys_result*, *OutputIterator2* *values_result*,
BinaryPredicate *binary_pred*)

`unique_by_key_copy` is a generalization of `unique_copy` to key-value pairs. For each group of consecutive keys in the range `[keys_first, keys_last)` that are equal, `unique_by_key_copy` copies the first element of the group to a range beginning with `keys_result` and the corresponding values from the range `[values_first, values_first + (keys_last - keys_first))` are copied to a range beginning with `values_result`.

This version of `unique_by_key_copy` uses the function object `binary_pred` to test for equality and `project1st` to reduce values with equal keys.

The following code snippet demonstrates how to use `unique_by_key_copy` to compact a sequence of key/value pairs and with equal keys.

Return A pair of iterators at end of the ranges `[keys_result, keys_result_last)` and `[values_result, values_result_last)`.

Pre The input ranges shall not overlap either output range.

Parameters

- `keys_first`: The beginning of the input key range.
- `keys_last`: The end of the input key range.
- `values_first`: The beginning of the input value range.
- `keys_result`: The beginning of the output key range.
- `values_result`: The beginning of the output value range.
- `binary_pred`: The binary predicate used to determine equality.

Template Parameters

- `InputIterator1`: is a model of [Input Iterator](#),
- `InputIterator2`: is a model of [Input Iterator](#),
- `OutputIterator1`: is a model of [Output Iterator](#) and `InputIterator1`'s `value_type` is convertible to `OutputIterator1`'s `value_type`.
- `OutputIterator2`: is a model of [Output Iterator](#) and `InputIterator2`'s `value_type` is convertible to `OutputIterator2`'s `value_type`.
- `BinaryPredicate`: is a model of [Binary Predicate](#).

```
#include <thrust/unique.h>
...
const int N = 7;
int A[N] = {1, 3, 3, 3, 2, 2, 1}; // input keys
int B[N] = {9, 8, 7, 6, 5, 4, 3}; // input values
int C[N]; // output keys
int D[N]; // output values

thrust::pair<int*, int*> new_end;
thrust::equal_to<int> binary_pred;
new_end = thrust::unique_by_key_copy(A, A + N, B, C, D, binary_pred);

// The first four keys in C are now {1, 3, 2, 1} and new_end.first - C is 4.
// The first four values in D are now {9, 8, 5, 3} and new_end.second - D is 4.
```

See `unique_copy`

See `unique_by_key`

See `reduce_by_key`

Template Function `thrust::unique_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator)`

- Defined in `file_thrust_unique.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::unique_copy” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
↳ typename BinaryPredicate>__host__ __device__ OutputIterator thrust::unique_
↳ copy(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ InputIterator, InputIterator, OutputIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator>_
↳ __host__ __device__ OutputIterator thrust::unique_copy(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
↳ InputIterator, OutputIterator)
- template<typename InputIterator, typename OutputIterator, typename
↳ BinaryPredicate>
  OutputIterator thrust::unique_copy(InputIterator, InputIterator, OutputIterator,
↳ BinaryPredicate)
- template<typename InputIterator, typename OutputIterator>
  OutputIterator thrust::unique_copy(InputIterator, InputIterator, OutputIterator)
```

Template Function thrust::unique_copy(InputIterator, InputIterator, OutputIterator)

Function Documentation

template<typename **InputIterator**, typename **OutputIterator**>

OutputIterator thrust::unique_copy (*InputIterator* first, *InputIterator* last, *OutputIterator* result)

unique_copy copies elements from the range [first, last) to a range beginning with result, except that in a consecutive group of duplicate elements only the first one is copied. The return value is the end of the range to which the elements are copied.

The reason there are two different versions of unique_copy is that there are two different definitions of what it means for a consecutive group of elements to be duplicates. In the first version, the test is simple equality: the elements in a range [f, l) are duplicates if, for every iterator i in the range, either i == f or else *i == *(i-1). In the second, the test is an arbitrary BinaryPredicate binary_pred: the elements in [f, l) are duplicates if, for every iterator i in the range, either i == f or else binary_pred(*i, *(i-1)) is true.

This version of unique_copy uses operator== to test for equality.

The following code snippet demonstrates how to use unique_copy to compact a sequence of numbers to remove consecutive duplicates.

Return The end of the unique range [result, result_end).

Pre The range [first,last) and the range [result, result + (last - first)) shall not overlap.

Parameters

- first: The beginning of the input range.
- last: The end of the input range.
- result: The beginning of the output range.

Template Parameters

- InputIterator: is a model of [Input Iterator](#), and InputIterator's value_type is a model of [Equality Comparable](#).
- OutputIterator: is a model of [Output Iterator](#) and InputIterator's value_type is convertible to OutputIterator's value_type.

```
#include <thrust/unique.h>
...
const int N = 7;
int A[N] = {1, 3, 3, 3, 2, 2, 1};
int B[N];
int *result_end = thrust::unique_copy(A, A + N, B);
// The first four values of B are now {1, 3, 2, 1} and (result_end - B) is 4
// Values beyond result_end are unspecified
```

See [unique](#)

See http://www.sgi.com/tech/stl/unique_copy.html

Template Function `thrust::unique_copy(const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, BinaryPredicate)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::unique_copy” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, InputIterator, InputIterator, OutputIterator, BinaryPredicate) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator,
→typename BinaryPredicate>__host__ __device__ OutputIterator thrust::unique_
→copy(const thrust::detail::execution_policy_base< DerivedPolicy > &,
→InputIterator, InputIterator, OutputIterator, BinaryPredicate)
- template<typename DerivedPolicy, typename InputIterator, typename OutputIterator>_
→__host__ __device__ OutputIterator thrust::unique_copy(const_
→thrust::detail::execution_policy_base< DerivedPolicy > &, InputIterator,
→InputIterator, OutputIterator)
- template<typename InputIterator, typename OutputIterator, typename_
→BinaryPredicate>
  OutputIterator thrust::unique_copy(InputIterator, InputIterator, OutputIterator,
→BinaryPredicate)
- template<typename InputIterator, typename OutputIterator>
  OutputIterator thrust::unique_copy(InputIterator, InputIterator, OutputIterator)
```

Template Function `thrust::unique_copy(InputIterator, InputIterator, OutputIterator, BinaryPredicate)`

Function Documentation

`template<typename InputIterator, typename OutputIterator, typename BinaryPredicate>`
`OutputIterator thrust::unique_copy (InputIterator first, InputIterator last, OutputIterator result, Bina-`
`ryPredicate binary_pred)`

`unique_copy` copies elements from the range `[first, last)` to a range beginning with `result`, except that in a consecutive group of duplicate elements only the first one is copied. The return value is the end of the range to which the elements are copied.

This version of `unique_copy` uses the function object `binary_pred` to test for equality.

The following code snippet demonstrates how to use `unique_copy` to compact a sequence of numbers to remove consecutive duplicates.

Return The end of the unique range `[result, result_end)`.

Pre The range `[first, last)` and the range `[result, result + (last - first))` shall not overlap.

Parameters

- `first`: The beginning of the input range.
- `last`: The end of the input range.
- `result`: The beginning of the output range.
- `binary_pred`: The binary predicate used to determine equality.

Template Parameters

- `InputIterator`: is a model of [Input Iterator](#), and `InputIterator`'s `value_type` is a model of [Equality Comparable](#).
- `OutputIterator`: is a model of [Output Iterator](#) and `InputIterator`'s `value_type` is convertible to `OutputIterator`'s `value_type`.
- `BinaryPredicate`: is a model of [Binary Predicate](#).

```
#include <thrust/unique.h>
...
const int N = 7;
int A[N] = {1, 3, 3, 3, 2, 2, 1};
int B[N];
int *result_end = thrust::unique_copy(A, A + N, B, thrust::equal_to<int>());
// The first four values of B are now {1, 3, 2, 1} and (result_end - B) is 4
// Values beyond result_end are unspecified.
```

See `unique`

See http://www.sgi.com/tech/stl/unique_copy.html

Template Function `thrust::upper_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const LessThanComparable&)`

- Defined in `file_thrust_binary_search.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “`thrust::upper_bound`” with arguments (`const thrust::detail::execution_policy_base<DerivedPolicy>&`, `ForwardIterator`, `ForwardIterator`, `const LessThanComparable&`) in doxygen xml output for project “rocThrust” from directory: `./docBin/xml`. Potential matches:

```

- template<class ForwardIterator, class InputIterator, class OutputIterator, class
↳ StrictWeakOrdering>
  OutputIterator thrust::upper_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)
- template<class ForwardIterator, class InputIterator, class OutputIterator>
  OutputIterator thrust::upper_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator)
- template<class ForwardIterator, class LessThanComparable>
  ForwardIterator thrust::upper_bound(ForwardIterator, ForwardIterator, const
↳ LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  ForwardIterator thrust::upper_bound(ForwardIterator, ForwardIterator, const T&,
↳ StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator, typename StrictWeakOrdering>__host__ __device__
↳ OutputIterator thrust::upper_bound(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, ForwardIterator, ForwardIterator, InputIterator, InputIterator,
↳ OutputIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::upper_
↳ bound(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳ LessThanComparable>__host__ __device__ ForwardIterator thrust::upper_bound(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
↳ StrictWeakOrdering>__host__ __device__ ForwardIterator thrust::upper_bound(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const T &, StrictWeakOrdering)

```

Template Function `thrust::upper_bound(ForwardIterator, ForwardIterator, const LessThanComparable&)`

Function Documentation

template<class **ForwardIterator**, class **LessThanComparable**>

ForwardIterator thrust::upper_bound(*ForwardIterator first*, *ForwardIterator last*, const *LessThanComparable &value*)

`upper_bound` is a version of binary search: it attempts to find the element value in an ordered range [*first*, *last*). Specifically, it returns the last position where value could be inserted without violating the ordering. This version of `upper_bound` uses `operator<` for comparison and returns the furthestmost iterator *i* in [*first*, *last*) such that, for every iterator *j* in [*first*, *i*), `value < *j` is false.

The following code snippet demonstrates how to use `upper_bound` to search for values in a ordered range.

Return The furthestmost iterator *i*, such that `value < *i` is false.

Parameters

- *first*: The beginning of the ordered sequence.
- *last*: The end of the ordered sequence.
- *value*: The value to be searched.

Template Parameters

- ForwardIterator: is a model of Forward Iterator.
- LessThanComparable: is a model of LessThanComparable.

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::upper_bound(input.begin(), input.end(), 0); // returns input.begin() + 1
thrust::upper_bound(input.begin(), input.end(), 1); // returns input.begin() + 1
thrust::upper_bound(input.begin(), input.end(), 2); // returns input.begin() + 2
thrust::upper_bound(input.begin(), input.end(), 3); // returns input.begin() + 2
thrust::upper_bound(input.begin(), input.end(), 8); // returns input.end()
thrust::upper_bound(input.begin(), input.end(), 9); // returns input.end()
```

See http://www.sgi.com/tech/stl/upper_bound.html

See `lower_bound`

See `equal_range`

See `binary_search`

Template Function `thrust::upper_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::upper_bound” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```
- template<class ForwardIterator, class InputIterator, class OutputIterator, class
↳ StrictWeakOrdering>
  OutputIterator thrust::upper_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)
- template<class ForwardIterator, class InputIterator, class OutputIterator>
  OutputIterator thrust::upper_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator)
- template<class ForwardIterator, class LessThanComparable>
  ForwardIterator thrust::upper_bound(ForwardIterator, ForwardIterator, const
↳ LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  ForwardIterator thrust::upper_bound(ForwardIterator, ForwardIterator, const T&,
↳ StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator, typename StrictWeakOrdering>__host__ __device__
↳ OutputIterator thrust::upper_bound(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, ForwardIterator, ForwardIterator, InputIterator, InputIterator,
↳ OutputIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::upper_
↳ bound(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳ LessThanComparable>__host__ __device__ ForwardIterator thrust::upper_bound(const
```

Template Function `thrust::upper_bound(ForwardIterator, ForwardIterator, const T&, StrictWeakOrdering)`

Function Documentation

template<class **ForwardIterator**, class **T**, class **StrictWeakOrdering**>

ForwardIterator thrust::upper_bound(*ForwardIterator* first, *ForwardIterator* last, const *T* &value, *StrictWeakOrdering* comp)

`upper_bound` is a version of binary search: it attempts to find the element value in an ordered range `[first, last)`. Specifically, it returns the last position where value could be inserted without violating the ordering. This version of `upper_bound` uses function object `comp` for comparison and returns the furthestmost iterator `i` in `[first, last)` such that, for every iterator `j` in `[first, i)`, `comp(value, *j)` is false.

The following code snippet demonstrates how to use `upper_bound` to search for values in a ordered range.

Return The furthestmost iterator `i`, such that `comp(value, *i)` is false.

Parameters

- `first`: The beginning of the ordered sequence.
- `last`: The end of the ordered sequence.
- `value`: The value to be searched.
- `comp`: The comparison operator.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#).
- `T`: is comparable to `ForwardIterator`'s `value_type`.
- `StrictWeakOrdering`: is a model of [Strict Weak Ordering](#).

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
#include <thrust/functional.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::upper_bound(input.begin(), input.end(), 0, thrust::less<int>()); //
↳ returns input.begin() + 1
thrust::upper_bound(input.begin(), input.end(), 1, thrust::less<int>()); //
↳ returns input.begin() + 1
thrust::upper_bound(input.begin(), input.end(), 2, thrust::less<int>()); //
↳ returns input.begin() + 2
thrust::upper_bound(input.begin(), input.end(), 3, thrust::less<int>()); //
↳ returns input.begin() + 2
```

(continues on next page)

(continued from previous page)

```

thrust::upper_bound(input.begin(), input.end(), 8, thrust::less<int>()); //
↳ returns input.end()
thrust::upper_bound(input.begin(), input.end(), 9, thrust::less<int>()); //
↳ returns input.end()

```

See http://www.sgi.com/tech/stl/upper_bound.html

See `lower_bound`

See `equal_range`

See `binary_search`

Template Function `thrust::upper_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::upper_bound” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<class ForwardIterator, class InputIterator, class OutputIterator, class
↳ StrictWeakOrdering>
  OutputIterator thrust::upper_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)
- template<class ForwardIterator, class InputIterator, class OutputIterator>
  OutputIterator thrust::upper_bound(ForwardIterator, ForwardIterator,
↳ InputIterator, InputIterator, OutputIterator)
- template<class ForwardIterator, class LessThanComparable>
  ForwardIterator thrust::upper_bound(ForwardIterator, ForwardIterator, const
↳ LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  ForwardIterator thrust::upper_bound(ForwardIterator, ForwardIterator, const T&,
↳ StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator, typename StrictWeakOrdering>__host__ __device__
↳ OutputIterator thrust::upper_bound(const thrust::detail::execution_policy_base<
↳ DerivedPolicy > &, ForwardIterator, ForwardIterator, InputIterator, InputIterator,
↳ OutputIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::upper_
↳ bound(const thrust::detail::execution_policy_base< DerivedPolicy > &,
↳ ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)
- template<typename DerivedPolicy, typename ForwardIterator, typename
↳ LessThanComparable>__host__ __device__ ForwardIterator thrust::upper_bound(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
↳ StrictWeakOrdering>__host__ __device__ ForwardIterator thrust::upper_bound(const
↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
↳ ForwardIterator, const T &, StrictWeakOrdering)

```

Template Function `thrust::upper_bound(ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)`

Function Documentation

template<class **ForwardIterator**, class **InputIterator**, class **OutputIterator**>

OutputIterator `thrust::upper_bound(ForwardIterator first, ForwardIterator last, InputIterator values_first, InputIterator values_last, OutputIterator result)`

`upper_bound` is a vectorized version of binary search: for each iterator `v` in `[values_first, values_last)` it attempts to find the value `*v` in an ordered range `[first, last)`. Specifically, it returns the index of last position where value could be inserted without violating the ordering.

The following code snippet demonstrates how to use `upper_bound` to search for multiple values in a ordered range.

Pre The ranges `[first, last)` and `[result, result + (last - first))` shall not overlap.

Parameters

- `first`: The beginning of the ordered sequence.
- `last`: The end of the ordered sequence.
- `values_first`: The beginning of the search values sequence.
- `values_last`: The end of the search values sequence.
- `result`: The beginning of the output sequence.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#).
- `InputIterator`: is a model of [Input Iterator](#). and `InputIterator`'s `value_type` is [LessThanComparable](#).
- `OutputIterator`: is a model of [Output Iterator](#). and `ForwardIterator`'s `difference_type` is convertible to `OutputIterator`'s `value_type`.

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::device_vector<int> values(6);
values[0] = 0;
values[1] = 1;
values[2] = 2;
values[3] = 3;
values[4] = 8;
values[5] = 9;

thrust::device_vector<unsigned int> output(6);

thrust::upper_bound(input.begin(), input.end(),
```

(continues on next page)

(continued from previous page)

```

        values.begin(), values.end(),
        output.begin());

// output is now [1, 1, 2, 2, 5, 5]

```

See http://www.sgi.com/tech/stl/upper_bound.html

See `upper_bound`

See `equal_range`

See `binary_search`

Template Function `thrust::upper_bound(const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)`

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “thrust::upper_bound” with arguments (const thrust::detail::execution_policy_base<DerivedPolicy>&, ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering) in doxygen xml output for project “rocThrust” from directory: ./docBin/xml. Potential matches:

```

- template<class ForwardIterator, class InputIterator, class OutputIterator, class
  ↳ StrictWeakOrdering>
  OutputIterator thrust::upper_bound(ForwardIterator, ForwardIterator,
  ↳ InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)
- template<class ForwardIterator, class InputIterator, class OutputIterator>
  OutputIterator thrust::upper_bound(ForwardIterator, ForwardIterator,
  ↳ InputIterator, InputIterator, OutputIterator)
- template<class ForwardIterator, class LessThanComparable>
  ForwardIterator thrust::upper_bound(ForwardIterator, ForwardIterator, const
  ↳ LessThanComparable&)
- template<class ForwardIterator, class T, class StrictWeakOrdering>
  ForwardIterator thrust::upper_bound(ForwardIterator, ForwardIterator, const T&,
  ↳ StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
  ↳ typename OutputIterator, typename StrictWeakOrdering>__host__ __device__
  ↳ OutputIterator thrust::upper_bound(const thrust::detail::execution_policy_base<
  ↳ DerivedPolicy > &, ForwardIterator, ForwardIterator, InputIterator, InputIterator,
  ↳ OutputIterator, StrictWeakOrdering)
- template<typename DerivedPolicy, typename ForwardIterator, typename InputIterator,
  ↳ typename OutputIterator>__host__ __device__ OutputIterator thrust::upper_
  ↳ bound(const thrust::detail::execution_policy_base< DerivedPolicy > &,
  ↳ ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator)
- template<typename DerivedPolicy, typename ForwardIterator, typename
  ↳ LessThanComparable>__host__ __device__ ForwardIterator thrust::upper_bound(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator, const LessThanComparable &)
- template<typename DerivedPolicy, typename ForwardIterator, typename T, typename
  ↳ StrictWeakOrdering>__host__ __device__ ForwardIterator thrust::upper_bound(const
  ↳ thrust::detail::execution_policy_base< DerivedPolicy > &, ForwardIterator,
  ↳ ForwardIterator, const T &, StrictWeakOrdering)

```

Template Function `thrust::upper_bound(ForwardIterator, ForwardIterator, InputIterator, InputIterator, OutputIterator, StrictWeakOrdering)`

Function Documentation

```
template<class ForwardIterator, class InputIterator, class OutputIterator, class StrictWeakOrdering>
OutputIterator thrust::upper_bound(ForwardIterator first, ForwardIterator last, InputIterator val-
                                   ues_first, InputIterator values_last, OutputIterator result,
                                   StrictWeakOrdering comp)
```

`upper_bound` is a vectorized version of binary search: for each iterator `v` in `[values_first, values_last)` it attempts to find the value `*v` in an ordered range `[first, last)`. Specifically, it returns the index of first position where value could be inserted without violating the ordering. This version of `upper_bound` uses function object `comp` for comparison.

The following code snippet demonstrates how to use `upper_bound` to search for multiple values in a ordered range.

Pre The ranges `[first, last)` and `[result, result + (last - first))` shall not overlap.

Parameters

- `first`: The beginning of the ordered sequence.
- `last`: The end of the ordered sequence.
- `values_first`: The beginning of the search values sequence.
- `values_last`: The end of the search values sequence.
- `result`: The beginning of the output sequence.
- `comp`: The comparison operator.

Template Parameters

- `ForwardIterator`: is a model of [Forward Iterator](#).
- `InputIterator`: is a model of [Input Iterator](#). and `InputIterator`'s `value_type` is comparable to `ForwardIterator`'s `value_type`.
- `OutputIterator`: is a model of [Output Iterator](#). and `ForwardIterator`'s `difference_type` is convertible to `OutputIterator`'s `value_type`.
- `StrictWeakOrdering`: is a model of [Strict Weak Ordering](#).

```
#include <thrust/binary_search.h>
#include <thrust/device_vector.h>
#include <thrust/functional.h>
...
thrust::device_vector<int> input(5);

input[0] = 0;
input[1] = 2;
input[2] = 5;
input[3] = 7;
input[4] = 8;

thrust::device_vector<int> values(6);
values[0] = 0;
values[1] = 1;
values[2] = 2;
```

(continues on next page)

(continued from previous page)

```

values[3] = 3;
values[4] = 8;
values[5] = 9;

thrust::device_vector<unsigned int> output(6);

thrust::upper_bound(input.begin(), input.end(),
                    values.begin(), values.end(),
                    output.begin(),
                    thrust::less<int>());

// output is now [1, 1, 2, 2, 5, 5]

```

See http://www.sgi.com/tech/stl/upper_bound.html

See `lower_bound`

See `equal_range`

See `binary_search`

1.3.4 Variables

Variable `thrust::device`

Variable Documentation

const detail::device_t thrust::device

`thrust::device` is the default parallel execution policy associated with Thrust's device backend system configured by the `THRUST_DEVICE_SYSTEM` macro.

Instead of relying on implicit algorithm dispatch through iterator system tags, users may directly target algorithm dispatch at Thrust's device system by providing `thrust::device` as an algorithm parameter.

Explicit dispatch can be useful in avoiding the introduction of data copies into containers such as `thrust::device_vector` or to avoid wrapping e.g. raw pointers allocated by the HIP API with types such as `thrust::device_ptr`.

The user must take care to guarantee that the iterators provided to an algorithm are compatible with the device backend system. For example, raw pointers allocated by `std::malloc` typically cannot be dereferenced by a GPU. For this reason, raw pointers allocated by host APIs should not be mixed with a `thrust::device` algorithm invocation when the device backend is HIP.

The type of `thrust::device` is implementation-defined.

The following code snippet demonstrates how to use `thrust::device` to explicitly dispatch an invocation of `thrust::for_each` to the device backend system:

```

#include <thrust/for_each.h>
#include <thrust/device_vector.h>
#include <thrust/execution_policy.h>
#include <cstdio>

struct printf_functor
{
    __host__ __device__

```

(continues on next page)

(continued from previous page)

```

void operator() (int x)
{
    printf("%d\n", x);
}
};
...
thrust::device_vector<int> vec(3);
vec[0] = 0; vec[1] = 1; vec[2] = 2;

thrust::for_each(thrust::device, vec.begin(), vec.end(), printf_functor());

// 0 1 2 is printed to standard output in some unspecified order

```

See *host_execution_policy*

See `thrust::device`

Variable `thrust::host`

Variable Documentation

`const detail::host_t thrust::host`

`thrust::host` is the default parallel execution policy associated with Thrust's host backend system configured by the `THRUST_HOST_SYSTEM` macro.

Instead of relying on implicit algorithm dispatch through iterator system tags, users may directly target algorithm dispatch at Thrust's host system by providing `thrust::host` as an algorithm parameter.

Explicit dispatch can be useful in avoiding the introduction of data copies into containers such as *`thrust::host_vector`*.

Note that even though `thrust::host` targets the host CPU, it is a parallel execution policy. That is, the order that an algorithm invokes functors or dereferences iterators is not defined.

The type of `thrust::host` is implementation-defined.

The following code snippet demonstrates how to use `thrust::host` to explicitly dispatch an invocation of `thrust::for_each` to the host backend system:

```

#include <thrust/for_each.h>
#include <thrust/execution_policy.h>
#include <cstdio>

struct printf_functor
{
    __host__ __device__
    void operator() (int x)
    {
        printf("%d\n", x);
    }
};
...
int vec(3);
vec[0] = 0; vec[1] = 1; vec[2] = 2;

thrust::for_each(thrust::host, vec.begin(), vec.end(), printf_functor());

```

(continues on next page)

(continued from previous page)

```
// 0 1 2 is printed to standard output in some unspecified order
```

See *host_execution_policy*

See `thrust::device`

Variable `thrust::placeholders::_1`

- Defined in `file_thrust_functional.h`

Variable Documentation

const `thrust::detail::functional::placeholder<0>::type thrust::placeholders::_1`
`thrust::placeholders::_1` is the placeholder for the first function parameter.

Variable `thrust::placeholders::_10`

- Defined in `file_thrust_functional.h`

Variable Documentation

const `thrust::detail::functional::placeholder<9>::type thrust::placeholders::_10`
`thrust::placeholders::_10` is the placeholder for the tenth function parameter.

Variable `thrust::placeholders::_2`

- Defined in `file_thrust_functional.h`

Variable Documentation

const `thrust::detail::functional::placeholder<1>::type thrust::placeholders::_2`
`thrust::placeholders::_2` is the placeholder for the second function parameter.

Variable `thrust::placeholders::_3`

- Defined in `file_thrust_functional.h`

Variable Documentation

const `thrust::detail::functional::placeholder<2>::type thrust::placeholders::_3`
`thrust::placeholders::_3` is the placeholder for the third function parameter.

Variable `thrust::placeholders::_4`

- Defined in `file_thrust_functional.h`

Variable Documentation

const thrust::detail::functional::placeholder<3>::type thrust::placeholders::_4
thrust::placeholders::_4 is the placeholder for the fourth function parameter.

Variable thrust::placeholders::_5

- Defined in file_thrust_functional.h

Variable Documentation

const thrust::detail::functional::placeholder<4>::type thrust::placeholders::_5
thrust::placeholders::_5 is the placeholder for the fifth function parameter.

Variable thrust::placeholders::_6

- Defined in file_thrust_functional.h

Variable Documentation

const thrust::detail::functional::placeholder<5>::type thrust::placeholders::_6
thrust::placeholders::_6 is the placeholder for the sixth function parameter.

Variable thrust::placeholders::_7

- Defined in file_thrust_functional.h

Variable Documentation

const thrust::detail::functional::placeholder<6>::type thrust::placeholders::_7
thrust::placeholders::_7 is the placeholder for the seventh function parameter.

Variable thrust::placeholders::_8

- Defined in file_thrust_functional.h

Variable Documentation

const thrust::detail::functional::placeholder<7>::type thrust::placeholders::_8
thrust::placeholders::_8 is the placeholder for the eighth function parameter.

Variable thrust::placeholders::_9

- Defined in file_thrust_functional.h

Variable Documentation

const thrust::detail::functional::placeholder<8>::type thrust::placeholders::_9
thrust::placeholders::_9 is the placeholder for the ninth function parameter.

1.3.5 Defines

Define `__THRUST_DEVICE_SYSTEM_EXECUTION_POLICY_HEADER`

- Defined in file_thrust_execution_policy.h

Define Documentation

`__THRUST_DEVICE_SYSTEM_EXECUTION_POLICY_HEADER`

Define `__THRUST_HOST_SYSTEM_EXECUTION_POLICY_HEADER`

- Defined in file_thrust_execution_policy.h

Define Documentation

`__THRUST_HOST_SYSTEM_EXECUTION_POLICY_HEADER`

Define `THRUST_MAJOR_VERSION`

- Defined in file_thrust_version.h

Define Documentation

`THRUST_MAJOR_VERSION`

The preprocessor macro `THRUST_MAJOR_VERSION` encodes the major version number of the Thrust library.

Define `THRUST_MINOR_VERSION`

- Defined in file_thrust_version.h

Define Documentation

`THRUST_MINOR_VERSION`

The preprocessor macro `THRUST_MINOR_VERSION` encodes the minor version number of the Thrust library.

Define `THRUST_PATCH_NUMBER`

- Defined in file_thrust_version.h

Define Documentation

THRUST_PATCH_NUMBER

The preprocessor macro `THRUST_PATCH_NUMBER` encodes the patch number of the Thrust library.

Define **THRUST_STD_COMPLEX_DEVICE**

- Defined in `file_thrust_complex.h`

Define Documentation

THRUST_STD_COMPLEX_DEVICE

Define **THRUST_STD_COMPLEX_IMAG**

- Defined in `file_thrust_complex.h`

Define Documentation

THRUST_STD_COMPLEX_IMAG (*z*)

Define **THRUST_STD_COMPLEX_REAL**

- Defined in `file_thrust_complex.h`

Define Documentation

THRUST_STD_COMPLEX_REAL (*z*)

Define **THRUST_SUBMINOR_VERSION**

- Defined in `file_thrust_version.h`

Define Documentation

THRUST_SUBMINOR_VERSION

The preprocessor macro `THRUST_SUBMINOR_VERSION` encodes the sub-minor version number of the Thrust library.

Define **THRUST_VERSION**

- Defined in `file_thrust_version.h`

Define Documentation

THRUST_VERSION

The preprocessor macro `THRUST_VERSION` encodes the version number of the Thrust library.

`THRUST_VERSION % 100` is the sub-minor version. `THRUST_VERSION / 100 % 1000` is the minor version. `THRUST_VERSION / 100000` is the major version.

1.3.6 Typedefs

Typedef `thrust::random::default_random_engine`

- Defined in `file_thrust_random.h`

Typedef Documentation

typedef `thrust::random::default_random_engine`

An implementation-defined “default” random number engine.

Note `default_random_engine` is currently an alias for `minstd_rand`, and may change in a future version.

Typedef `thrust::random::ranlux24`

- Defined in `file_thrust_random.h`

Typedef Documentation

typedef `thrust::random::ranlux24`

A random number engine with predefined parameters which implements the RANLUX level-3 random number generation algorithm.

Note The 10000th consecutive invocation of a default-constructed object of type `ranlux24` shall produce the value 9901578.

Typedef `thrust::random::ranlux48`

- Defined in `file_thrust_random.h`

Typedef Documentation

typedef `thrust::random::ranlux48`

A random number engine with predefined parameters which implements the RANLUX level-4 random number generation algorithm.

Note The 10000th consecutive invocation of a default-constructed object of type `ranlux48` shall produce the value 88229545517833.

Typedef `thrust::random::taus88`

- Defined in `file_thrust_random.h`

Typedef Documentation

typedef `thrust::random::taus88`

A random number engine with predefined parameters which implements L'Ecuyer's 1996 three-component Tausworthe random number generator.

Note The 10000th consecutive invocation of a default-constructed object of type `taus88` shall produce the value 3535848941 .

INDICES AND TABLES

- `genindex`
- `search`

Symbols

__THRUST_DEVICE_SYSTEM_EXECUTION_POLICY_HEADER
(*C macro*), 369

__THRUST_HOST_SYSTEM_EXECUTION_POLICY_HEADER
(*C macro*), 369

T

thrust::adjacent_difference (*C++ function*), 74

thrust::all_of (*C++ function*), 77

thrust::any_of (*C++ function*), 78

thrust::binary_function (*C++ class*), 20

thrust::binary_function::first_argument_type
(*C++ type*), 20

thrust::binary_function::result_type
(*C++ type*), 21

thrust::binary_function::second_argument_type
(*C++ type*), 20

thrust::binary_negate (*C++ class*), 21

thrust::binary_search (*C++ function*), 81, 82, 84, 86

thrust::binary_traits (*C++ class*), 21

thrust::bit_and (*C++ class*), 22

thrust::bit_and::first_argument_type
(*C++ type*), 22

thrust::bit_and::result_type (*C++ type*), 22

thrust::bit_and::second_argument_type
(*C++ type*), 22

thrust::bit_or (*C++ class*), 23

thrust::bit_or::first_argument_type
(*C++ type*), 23

thrust::bit_or::result_type (*C++ type*), 23

thrust::bit_or::second_argument_type
(*C++ type*), 23

thrust::bit_xor (*C++ class*), 24

thrust::bit_xor::first_argument_type
(*C++ type*), 24

thrust::bit_xor::result_type (*C++ type*), 24

thrust::bit_xor::second_argument_type
(*C++ type*), 24

thrust::complex (*C++ class*), 25

thrust::complex::value_type (*C++ type*), 25

thrust::copy (*C++ function*), 88

thrust::copy_if (*C++ function*), 90, 91

thrust::copy_n (*C++ function*), 93

thrust::count (*C++ function*), 95

thrust::count_if (*C++ function*), 96

thrust::device (*C++ member*), 365

thrust::device_allocator (*C++ class*), 55

thrust::device_allocator::const_pointer
(*C++ type*), 56

thrust::device_allocator::difference_type
(*C++ type*), 56

thrust::device_allocator::pointer (*C++ type*), 56

thrust::device_allocator::rebind (*C++ class*), 29, 55

thrust::device_allocator::size_type
(*C++ type*), 56

thrust::device_allocator::value_type
(*C++ type*), 56

thrust::device_allocator<T>::rebind::other
(*C++ type*), 29, 55

thrust::device_allocator<void> (*C++ class*), 56

thrust::device_allocator<void>::rebind
(*C++ class*), 29, 56

thrust::device_allocator<void>::rebind::other
(*C++ type*), 30, 56

thrust::device_delete (*C++ function*), 97

thrust::device_execution_policy (*C++ class*), 30

thrust::device_free (*C++ function*), 97

thrust::device_malloc (*C++ function*), 98

thrust::device_malloc_allocator (*C++ class*), 57

thrust::device_malloc_allocator::const_pointer
(*C++ type*), 57

thrust::device_malloc_allocator::const_reference
(*C++ type*), 57

thrust::device_malloc_allocator::difference_type
(*C++ type*), 57

thrust::device_malloc_allocator::max_size
(*C++ function*), 58

`thrust::device_malloc_allocator::pointer` (C++ type), 33
(C++ type), 57
`thrust::device_malloc_allocator::rebind` 33
(C++ class), 31, 58
`thrust::device_malloc_allocator::reference` (C++ type), 33
(C++ type), 57
`thrust::device_malloc_allocator::size_type` 109, 111
(C++ type), 57
`thrust::device_malloc_allocator::value_type` 109, 111
(C++ type), 57
`thrust::device_malloc_allocator<T>::rebind` 33
(C++ type), 31, 58
`thrust::device_new` (C++ function), 99
`thrust::device_new_allocator` (C++ class), 59
`thrust::device_new_allocator::const_pointer` (C++ type), 59
`thrust::device_new_allocator::const_reference` (C++ type), 59
`thrust::device_new_allocator::difference_type` (C++ type), 59
`thrust::device_new_allocator::pointer` (C++ type), 59
`thrust::device_new_allocator::rebind` (C++ class), 32, 60
`thrust::device_new_allocator::reference` (C++ type), 59
`thrust::device_new_allocator::size_type` (C++ type), 59
`thrust::device_new_allocator::value_type` (C++ type), 59
`thrust::device_new_allocator<T>::rebind` (C++ type), 32, 61
`thrust::device_ptr` (C++ class), 61
`thrust::device_reference` (C++ class), 62
`thrust::device_reference::pointer` (C++ type), 65
`thrust::device_reference::value_type` (C++ type), 65
`thrust::device_vector` (C++ class), 66
`thrust::device_vector::~~device_vector` (C++ function), 67
`thrust::device_vector::device_vector` (C++ function), 67, 68
`thrust::divides` (C++ class), 32
`thrust::divides::first_argument_type` (C++ type), 33
`thrust::divides::result_type` (C++ type), 33
`thrust::divides::second_argument_type` (C++ type), 33
`thrust::equal` (C++ function), 101, 103
`thrust::equal_range` (C++ function), 104, 106
`thrust::equal_to` (C++ class), 33
`thrust::equal_to::first_argument_type` (C++ type), 33
`thrust::equal_to::result_type` (C++ type), 33
`thrust::equal_to::second_argument_type` (C++ type), 33
`thrust::exclusive_scan` (C++ function), 108, 109, 111
`thrust::exclusive_scan_by_key` (C++ function), 113, 115, 117, 119
`thrust::find` (C++ function), 123
`thrust::find_if` (C++ function), 124
`thrust::find_if_not` (C++ function), 125
`thrust::for_each` (C++ function), 127
`thrust::for_each_n` (C++ function), 128
`thrust::gather` (C++ function), 130
`thrust::gather_if` (C++ function), 132, 134
`thrust::generate` (C++ function), 136
`thrust::generate_n` (C++ function), 137
`thrust::greater` (C++ class), 34
`thrust::greater::first_argument_type` (C++ type), 34
`thrust::greater::result_type` (C++ type), 34
`thrust::greater::second_argument_type` (C++ type), 34
`thrust::greater_equal` (C++ class), 35
`thrust::greater_equal::first_argument_type` (C++ type), 35
`thrust::greater_equal::result_type` (C++ type), 35
`thrust::greater_equal::second_argument_type` (C++ type), 35
`thrust::host` (C++ member), 366
`thrust::host_execution_policy` (C++ class), 35
`thrust::host_vector` (C++ class), 68
`thrust::host_vector::~~host_vector` (C++ function), 69
`thrust::host_vector::host_vector` (C++ function), 69, 70
`thrust::identity` (C++ class), 37
`thrust::identity::argument_type` (C++ type), 37
`thrust::identity::result_type` (C++ type), 37
`thrust::inclusive_scan` (C++ function), 140, 141
`thrust::inclusive_scan_by_key` (C++ function), 143, 145, 147
`thrust::inner_product` (C++ function), 149, 150
`thrust::is_partitioned` (C++ function), 152
`thrust::is_sorted` (C++ function), 153, 155
`thrust::is_sorted_until` (C++ function), 156, 157
`thrust::less` (C++ class), 37

[thrust::less::first_argument_type \(C++ type\), 38](#)
[thrust::less::result_type \(C++ type\), 38](#)
[thrust::less::second_argument_type \(C++ type\), 38](#)
[thrust::less_equal \(C++ class\), 38](#)
[thrust::less_equal::first_argument_type \(C++ type\), 38](#)
[thrust::less_equal::result_type \(C++ type\), 38](#)
[thrust::less_equal::second_argument_type \(C++ type\), 38](#)
[thrust::logical_and \(C++ class\), 39](#)
[thrust::logical_and::first_argument_type \(C++ type\), 39](#)
[thrust::logical_and::result_type \(C++ type\), 39](#)
[thrust::logical_and::second_argument_type \(C++ type\), 39](#)
[thrust::logical_not \(C++ class\), 39](#)
[thrust::logical_not::first_argument_type \(C++ type\), 40](#)
[thrust::logical_not::result_type \(C++ type\), 40](#)
[thrust::logical_not::second_argument_type \(C++ type\), 40](#)
[thrust::logical_or \(C++ class\), 40](#)
[thrust::logical_or::first_argument_type \(C++ type\), 41](#)
[thrust::logical_or::result_type \(C++ type\), 41](#)
[thrust::logical_or::second_argument_type \(C++ type\), 41](#)
[thrust::lower_bound \(C++ function\), 160, 161, 163, 165](#)
[thrust::max_element \(C++ function\), 169, 170](#)
[thrust::maximum \(C++ class\), 41](#)
[thrust::maximum::first_argument_type \(C++ type\), 41](#)
[thrust::maximum::result_type \(C++ type\), 41](#)
[thrust::maximum::second_argument_type \(C++ type\), 41](#)
[thrust::merge \(C++ function\), 172, 173](#)
[thrust::merge_by_key \(C++ function\), 175, 178](#)
[thrust::min_element \(C++ function\), 180, 182](#)
[thrust::minimum \(C++ class\), 42](#)
[thrust::minimum::first_argument_type \(C++ type\), 42](#)
[thrust::minimum::result_type \(C++ type\), 42](#)
[thrust::minimum::second_argument_type \(C++ type\), 42](#)
[thrust::minmax_element \(C++ function\), 183, 184](#)
[thrust::minus \(C++ class\), 43](#)
[thrust::minus::first_argument_type \(C++ type\), 43](#)
[thrust::minus::result_type \(C++ type\), 43](#)
[thrust::minus::second_argument_type \(C++ type\), 43](#)
[thrust::mismatch \(C++ function\), 186, 188](#)
[thrust::modulus \(C++ class\), 44](#)
[thrust::modulus::first_argument_type \(C++ type\), 44](#)
[thrust::modulus::result_type \(C++ type\), 44](#)
[thrust::modulus::second_argument_type \(C++ type\), 44](#)
[thrust::multiplies \(C++ class\), 45](#)
[thrust::multiplies::first_argument_type \(C++ type\), 45](#)
[thrust::multiplies::result_type \(C++ type\), 45](#)
[thrust::multiplies::second_argument_type \(C++ type\), 45](#)
[thrust::negate \(C++ class\), 46](#)
[thrust::negate::argument_type \(C++ type\), 46](#)
[thrust::negate::result_type \(C++ type\), 46](#)
[thrust::none_of \(C++ function\), 189](#)
[thrust::not_equal_to \(C++ class\), 47](#)
[thrust::not_equal_to::first_argument_type \(C++ type\), 47](#)
[thrust::not_equal_to::result_type \(C++ type\), 47](#)
[thrust::not_equal_to::second_argument_type \(C++ type\), 47](#)
[thrust::operator>> \(C++ function\), 204](#)
[thrust::operator<< \(C++ function\), 200](#)
[thrust::pair \(C++ class\), 47](#)
[thrust::pair::first \(C++ member\), 49](#)
[thrust::pair::first_type \(C++ type\), 48](#)
[thrust::pair::second \(C++ member\), 49](#)
[thrust::pair::second_type \(C++ type\), 48](#)
[thrust::partition \(C++ function\), 205, 206](#)
[thrust::partition_copy \(C++ function\), 208, 210](#)
[thrust::partition_point \(C++ function\), 212](#)
[thrust::placeholders::_1 \(C++ member\), 367](#)
[thrust::placeholders::_10 \(C++ member\), 367](#)
[thrust::placeholders::_2 \(C++ member\), 367](#)
[thrust::placeholders::_3 \(C++ member\), 367](#)
[thrust::placeholders::_4 \(C++ member\), 368](#)
[thrust::placeholders::_5 \(C++ member\), 368](#)
[thrust::placeholders::_6 \(C++ member\), 368](#)
[thrust::placeholders::_7 \(C++ member\), 368](#)
[thrust::placeholders::_8 \(C++ member\), 368](#)
[thrust::placeholders::_9 \(C++ member\), 369](#)
[thrust::plus \(C++ class\), 49](#)

`thrust::plus::first_argument_type` (C++ type), 49
`thrust::plus::result_type` (C++ type), 50
`thrust::plus::second_argument_type` (C++ type), 49
`thrust::project1st` (C++ class), 50
`thrust::project1st::first_argument_type` (C++ type), 50
`thrust::project1st::result_type` (C++ type), 50
`thrust::project1st::second_argument_type` (C++ type), 50
`thrust::project2nd` (C++ class), 51
`thrust::project2nd::first_argument_type` (C++ type), 51
`thrust::project2nd::result_type` (C++ type), 51
`thrust::project2nd::second_argument_type` (C++ type), 51
`thrust::random::default_random_engine` (C++ type), 371
`thrust::random::ranlux24` (C++ type), 371
`thrust::random::ranlux48` (C++ type), 371
`thrust::random::taus88` (C++ type), 372
`thrust::reduce` (C++ function), 216, 217, 219
`thrust::reduce_by_key` (C++ function), 220, 223, 225
`thrust::remove` (C++ function), 226
`thrust::remove_copy` (C++ function), 228
`thrust::remove_copy_if` (C++ function), 229, 231
`thrust::remove_if` (C++ function), 233, 234
`thrust::replace` (C++ function), 236
`thrust::replace_copy` (C++ function), 237
`thrust::replace_copy_if` (C++ function), 239, 241
`thrust::replace_if` (C++ function), 243, 244
`thrust::reverse` (C++ function), 247
`thrust::reverse_copy` (C++ function), 248
`thrust::scatter` (C++ function), 249
`thrust::scatter_if` (C++ function), 251, 253
`thrust::sequence` (C++ function), 254, 256, 257
`thrust::set_difference` (C++ function), 258, 260
`thrust::set_difference_by_key` (C++ function), 262, 265
`thrust::set_intersection` (C++ function), 268, 270
`thrust::set_intersection_by_key` (C++ function), 272, 275
`thrust::set_symmetric_difference` (C++ function), 278, 280
`thrust::set_symmetric_difference_by_key` (C++ function), 282, 286
`thrust::set_union` (C++ function), 290, 292
`thrust::set_union_by_key` (C++ function), 294, 297
`thrust::sort` (C++ function), 300
`thrust::sort_by_key` (C++ function), 302, 304
`thrust::stable_partition` (C++ function), 305, 307
`thrust::stable_partition_copy` (C++ function), 309, 311
`thrust::stable_sort` (C++ function), 313, 314
`thrust::stable_sort_by_key` (C++ function), 316, 317
`thrust::swap_ranges` (C++ function), 320
`thrust::tabulate` (C++ function), 322
`thrust::transform` (C++ function), 325, 327
`thrust::transform_exclusive_scan` (C++ function), 328
`thrust::transform_if` (C++ function), 330, 332, 334
`thrust::transform_inclusive_scan` (C++ function), 336
`thrust::transform_reduce` (C++ function), 338
`thrust::tuple` (C++ class), 70
`thrust::tuple_element` (C++ class), 51
`thrust::tuple_element::type` (C++ type), 52
`thrust::tuple_size` (C++ class), 52
`thrust::tuple_size::value` (C++ member), 52
`thrust::unary_function` (C++ class), 53
`thrust::unary_function::argument_type` (C++ type), 53
`thrust::unary_function::result_type` (C++ type), 53
`thrust::unary_negate` (C++ class), 53
`thrust::unary_traits` (C++ class), 54
`thrust::uninitialized_copy` (C++ function), 339
`thrust::uninitialized_copy_n` (C++ function), 341
`thrust::uninitialized_fill` (C++ function), 342
`thrust::uninitialized_fill_n` (C++ function), 344
`thrust::unique` (C++ function), 345, 347
`thrust::unique_by_key` (C++ function), 348, 349
`thrust::unique_by_key_copy` (C++ function), 351, 353
`thrust::unique_copy` (C++ function), 355, 356
`thrust::upper_bound` (C++ function), 358, 360, 362, 364
`THRUST_MAJOR_VERSION` (C macro), 369
`THRUST_MINOR_VERSION` (C macro), 369
`THRUST_PATCH_NUMBER` (C macro), 370
`THRUST_STD_COMPLEX_DEVICE` (C macro), 370
`THRUST_STD_COMPLEX_IMAG` (C macro), 370

THRUST_STD_COMPLEX_REAL (*C macro*), [370](#)
THRUST_SUBMINOR_VERSION (*C macro*), [370](#)
THRUST_VERSION (*C macro*), [371](#)