

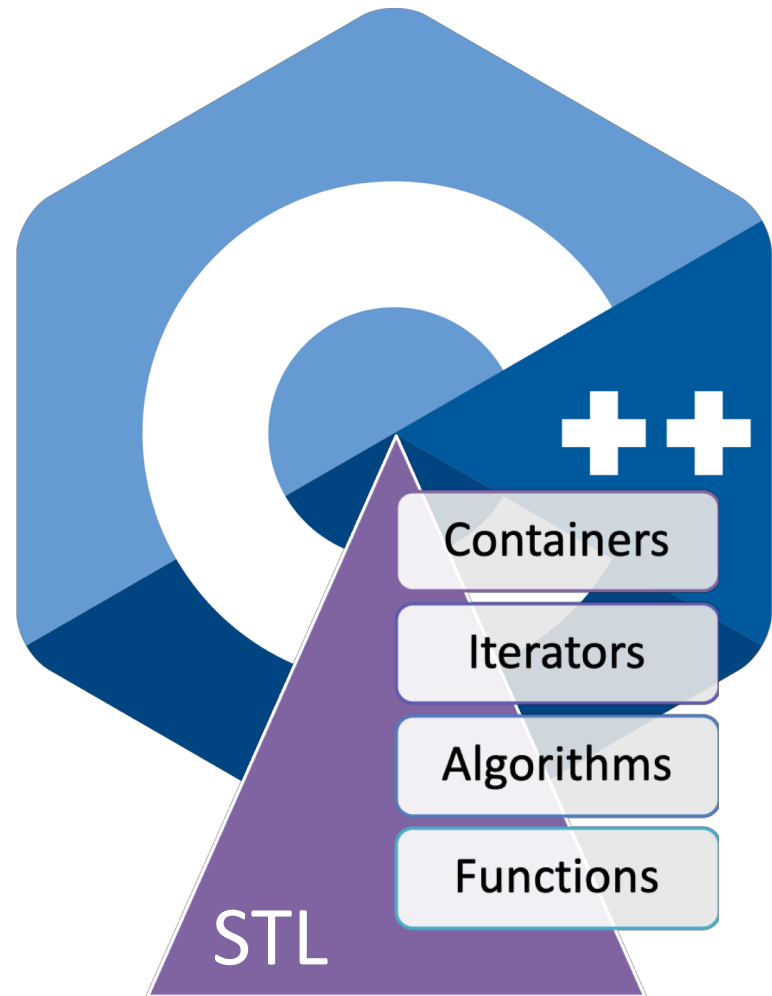
Dipping into C++17 Parallel Algorithms with Intel's Parallel STL

Armin Sobhani
asobhani@sharcnet.ca

SHARCNET

University of Ontario Institute
of Technology (UOIT)

February 27, 2019



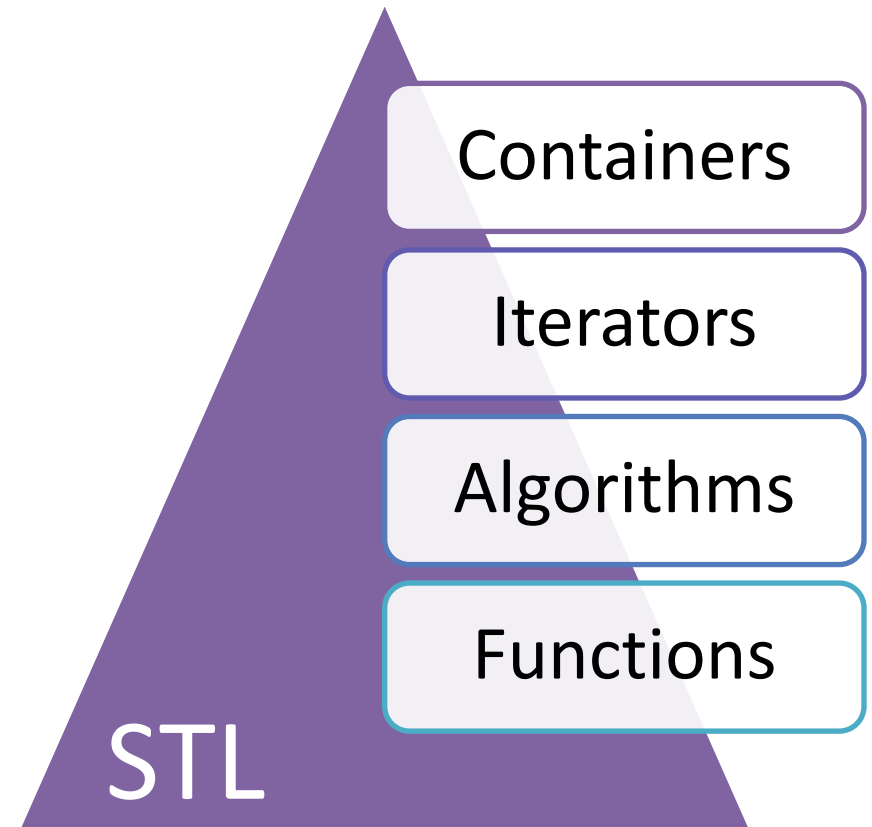
Outline

- An overview of the C++17 execution policies
- How to use the Intel's Parallel STL library
- A tutorial/benchmarks for exploring Parallel STL library on SHARCNET clusters:

[https://git.sharcnet.ca/asobhani/parallelstl tutorial](https://git.sharcnet.ca/asobhani/parallelstl_tutorial)

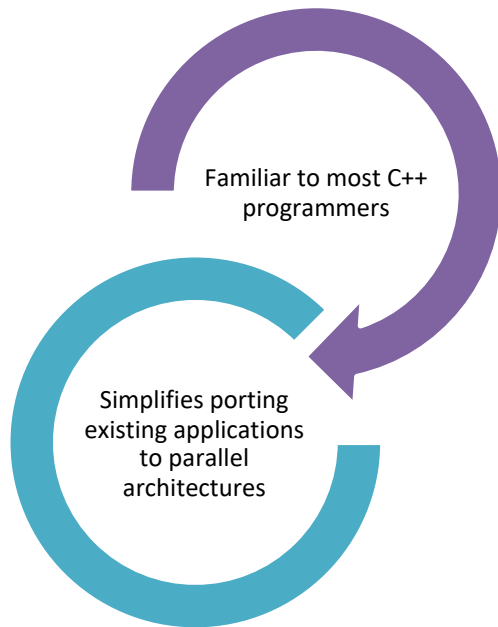
Standard Template Library (STL)

- Software library for the C++
- Influenced many parts of the C++ Standard Library
- Consisting of 4 components:



Parallel STL

Why?



Available Implementations

C++17 Parallel Algorithms

- Microsoft Visual Studio 2017 15.5
- Intel's open source Parallel STL
- STE | | AR Group's HPX library
- KhronosGroup's SYCL Parallel STL

Third-Party C++ Libraries

- Boost.Compute
- Nvidia's Thrust
- AMD's Bolt

C++17 Execution Policy 101

C++17 Execution Policy

- Most algorithms have overloads that accept execution policies as the first argument:
 - **std::execution::seq** (since C++17)
sequential execution, implements **sequenced_policytype**
 - **std::execution::unseq** (since C++20)
unsequenced execution, implements **unsequenced_policytype**
 - **std::execution::par** (since C++17)
parallel execution, implements **parallel_policy** type
 - **std::execution::par_unseq** (since C++17)
parallel and unsequenced execution,
implements **parallel_unsequenced_policytype**
- Execution policies are permissions, not obligations
- They require at least forward iterators

STL Algorithms – Overview

Algorithms

<code>accumulate()</code>	<code>includes()</code>	<code>partition()</code>	<code>stable_partition()</code>
<code>adjacent_difference()</code>	<code>inclusive_scan()</code>	<code>partition_copy()</code>	<code>stable_sort()</code>
<code>adjacent_find()</code>	<code>inner_product()</code>	<code>partition_point()</code>	<code>swap_ranges()</code>
<code>all_of()</code>	<code>inplace_merge()</code>	<code>prev_permutation()</code>	<code>transform()</code>
<code>any_of()</code>	<code>iota()</code>	<code>random_shuffle()</code>	<code>transform_exclusive_scan()</code>
<code>binary_search()</code>	<code>is_heap()</code>	<code>reduce()</code>	<code>transform_inclusive_scan()</code>
<code>copy()</code>	<code>is_heap_until()</code>	<code>remove()</code>	<code>transform_reduce()</code>
<code>copy_if()</code>	<code>is_partitioned()</code>	<code>remove_copy()</code>	<code>uninitialized_copy()</code>
<code>copy_n()</code>	<code>is_permutation()</code>	<code>remove_copy_if()</code>	<code>uninitialized_copy_n()</code>
<code>count()</code>	<code>is_sorted()</code>	<code>remove_if()</code>	<code>uninitialized_default_construct()</code>
<code>count_if()</code>	<code>is_sorted_until()</code>	<code>replace()</code>	<code>uninitialized_default_construct_n()</code>
<code>destroy()</code>	<code>lower_bound()</code>	<code>replace_if()</code>	<code>uninitialized_fill()</code>
<code>destroy_n()</code>	<code>lexicographical_compare()</code>	<code>replace_copy()</code>	<code>uninitialized_fill_n()</code>
<code>equal()</code>	<code>max_element()</code>	<code>replace_copy_if()</code>	<code>uninitialized_move()</code>
<code>equal_range()</code>	<code>merge()</code>	<code>reverse()</code>	<code>uninitialized_move_n()</code>
<code>exclusive_scan()</code>	<code>min_element()</code>	<code>reverse_copy()</code>	<code>uninitialized_value_construct()</code>
<code>fill()</code>	<code>minmax_element()</code>	<code>rotate()</code>	<code>uninitialized_value_construct_n()</code>
<code>fill_n()</code>	<code>mismatch()</code>	<code>rotate_copy()</code>	<code>unique()</code>
<code>find()</code>	<code>next_permutation()</code>	<code>search()</code>	<code>unique_copy()</code>
<code>find_end()</code>	<code>move()</code>	<code>search_n()</code>	<code>upper_bound()</code>
<code>find_if()</code>	<code>none_of()</code>	<code>set_difference()</code>	
<code>find_if_not()</code>	<code>nth_element()</code>	<code>set_intersection()</code>	
<code>for_each()</code>	<code>partial_sum()</code>	<code>set_symmetric_difference()</code>	
<code>for_each_n()</code>	<code>partial_sort()</code>	<code>set_union()</code>	
<code>generate()</code>	<code>partial_sort_copy()</code>	<code>sort()</code>	
<code>generate_n()</code>			

STL Algorithms – No Execution Policy

Algorithms

<code>accumulate()</code>	<code>includes()</code>	<code>partition()</code>	<code>stable_partition()</code>
<code>adjacent_difference()</code>	<code>inclusive_scan()</code>	<code>partition_copy()</code>	<code>stable_sort()</code>
<code>adjacent_find()</code>	<code>inner_product()</code>	<code>partition_point()</code>	<code>swap_ranges()</code>
<code>all_of()</code>	<code>inplace_merge()</code>	<code>prev_permutation()</code>	<code>transform()</code>
<code>any_of()</code>	<code>iota()</code>	<code>random_shuffle()</code>	<code>transform_exclusive_scan()</code>
<code>binary_search()</code>	<code>is_heap()</code>	<code>reduce()</code>	<code>transform_inclusive_scan()</code>
<code>copy()</code>	<code>is_heap_until()</code>	<code>remove()</code>	<code>transform_reduce()</code>
<code>copy_if()</code>	<code>is_partitioned()</code>	<code>remove_copy()</code>	<code>uninitialized_copy()</code>
<code>copy_n()</code>	<code>is_permutation()</code>	<code>remove_copy_if()</code>	<code>uninitialized_copy_n()</code>
<code>count()</code>	<code>is_sorted()</code>	<code>remove_if()</code>	<code>uninitialized_default_construct()</code>
<code>count_if()</code>	<code>is_sorted_until()</code>	<code>replace()</code>	<code>uninitialized_default_construct_n()</code>
<code>destroy()</code>	<code>lower_bound()</code>	<code>replace_if()</code>	<code>uninitialized_fill()</code>
<code>destroy_n()</code>	<code>lexicographical_compare()</code>	<code>replace_copy()</code>	<code>uninitialized_fill_n()</code>
<code>equal()</code>	<code>max_element()</code>	<code>replace_copy_if()</code>	<code>uninitialized_move()</code>
<code>equal_range()</code>	<code>merge()</code>	<code>reverse()</code>	<code>uninitialized_move_n()</code>
<code>exclusive_scan()</code>	<code>min_element()</code>	<code>reverse_copy()</code>	<code>uninitialized_value_construct()</code>
<code>fill()</code>	<code>minmax_element()</code>	<code>rotate()</code>	<code>uninitialized_value_construct_n()</code>
<code>fill_n()</code>	<code>mismatch()</code>	<code>rotate_copy()</code>	<code>unique()</code>
<code>find()</code>	<code>next_permutation()</code>	<code>search()</code>	<code>unique_copy()</code>
<code>find_end()</code>	<code>move()</code>	<code>search_n()</code>	<code>upper_bound()</code>
<code>find_if()</code>	<code>none_of()</code>	<code>set_difference()</code>	
<code>find_if_not()</code>	<code>nth_element()</code>	<code>set_intersection()</code>	
<code>for_each()</code>	<code>partial_sum()</code>	<code>set_symmetric_difference()</code>	
<code>for_each_n()</code>	<code>partial_sort()</code>	<code>set_union()</code>	
<code>generate()</code>	<code>partial_sort_copy()</code>	<code>sort()</code>	
<code>generate_n()</code>			

STL Algorithms – Parallel by Nature

Algorithms

<code>accumulate()</code>	<code>includes()</code>	<code>partition()</code>	<code>stable_partition()</code>
<code>adjacent_difference()</code>	<code>inclusive_scan()</code>	<code>partition_copy()</code>	<code>stable_sort()</code>
<code>adjacent_find()</code>	<code>inner_product()</code>	<code>partition_point()</code>	<code>swap_ranges()</code>
<code>all_of()</code>	<code>inplace_merge()</code>	<code>prev_permutation()</code>	<code>transform()</code>
<code>any_of()</code>	<code>iota()</code>	<code>random_shuffle()</code>	<code>transform_exclusive_scan()</code>
<code>binary_search()</code>	<code>is_heap()</code>	<code>reduce()</code>	<code>transform_inclusive_scan()</code>
<code>copy()</code>	<code>is_heap_until()</code>	<code>remove()</code>	<code>transform_reduce()</code>
<code>copy_if()</code>	<code>is_partitioned()</code>	<code>remove_copy()</code>	<code>uninitialized_copy()</code>
<code>copy_n()</code>	<code>is_permutation()</code>	<code>remove_copy_if()</code>	<code>uninitialized_copy_n()</code>
<code>count()</code>	<code>is_sorted()</code>	<code>remove_if()</code>	<code>uninitialized_default_construct()</code>
<code>count_if()</code>	<code>is_sorted_until()</code>	<code>replace()</code>	<code>uninitialized_default_construct_n()</code>
<code>destroy()</code>	<code>lower_bound()</code>	<code>replace_if()</code>	<code>uninitialized_fill()</code>
<code>destroy_n()</code>	<code>lexicographical_compare()</code>	<code>replace_copy()</code>	<code>uninitialized_fill_n()</code>
<code>equal()</code>	<code>max_element()</code>	<code>replace_copy_if()</code>	<code>uninitialized_move()</code>
<code>equal_range()</code>	<code>merge()</code>	<code>reverse()</code>	<code>uninitialized_move_n()</code>
<code>exclusive_scan()</code>	<code>min_element()</code>	<code>reverse_copy()</code>	<code>uninitialized_value_construct()</code>
<code>fill()</code>	<code>minmax_element()</code>	<code>rotate()</code>	<code>uninitialized_value_construct_n()</code>
<code>fill_n()</code>	<code>mismatch()</code>	<code>rotate_copy()</code>	<code>unique()</code>
<code>find()</code>	<code>next_permutation()</code>	<code>search()</code>	<code>unique_copy()</code>
<code>find_end()</code>	<code>move()</code>	<code>search_n()</code>	<code>upper_bound()</code>
<code>find_if()</code>	<code>none_of()</code>	<code>set_difference()</code>	
<code>find_if_not()</code>	<code>nth_element()</code>	<code>set_intersection()</code>	
<code>for_each()</code>	<code>partial_sum()</code>	<code>set_symmetric_difference()</code>	
<code>for_each_n()</code>	<code>partial_sort()</code>	<code>set_union()</code>	
<code>generate()</code>	<code>partial_sort_copy()</code>	<code>sort()</code>	
<code>generate_n()</code>			

Sort Example

STL

```
#include <vector>
#include <algorithm>

// fill the vector with some data
std::vector<int> v{...};

std::sort(v.begin(), v.end());
```

C++17

```
#include <vector>
#include <algorithm>
#include <execution>

// fill the vector with some data
std::vector<int> v{...};

std::sort(std::execution::par, v.begin(), v.end());
```

Sort Example

STL

```
#include <vector>
#include <algorithm>

// fill the vector with some data
std::vector<int> v{...};

std::sort(v.begin(), v.end());
```

C++17

```
#include <vector>
#include <algorithm>
{ #include <execution>

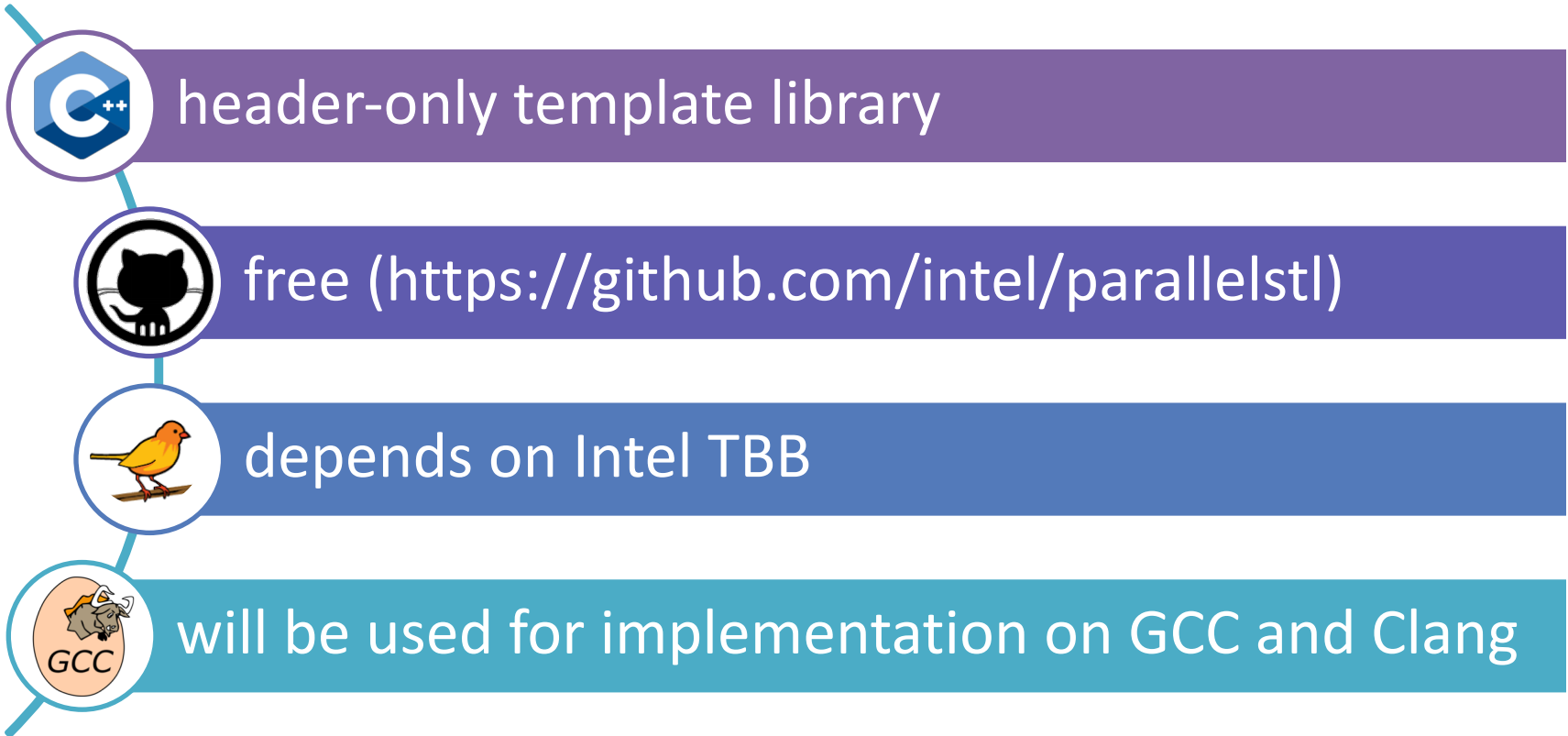
// fill the vector with some data
std::vector<int> v{...};

std::sort(std::execution::par, v.begin(), v.end());

std::execution::seq      (since C++17)
std::execution::unseq    (since C++20)
std::execution::par      (since C++17)
std::execution::par_unseq (since C++17)
```


Intel's Parallel STL

Intel's Parallel STL



Parallel STL Algorithms – Overview

■ not provided

■ parallel version not implemented

■ only parallel

■ parallel and unsequenced

Algorithms

accumulate()
adjacent_difference()
adjacent_find()
all_of()
any_of()
binary_search()
copy()
copy_if()
copy_n()
count()
count_if()
destroy()
destroy_n()
equal()
equal_range()
exclusive_scan()
fill()
fill_n()
find()
find_end()
find_if()
find_if_not()
for_each()
for_each_n()
generate()
generate_n()

includes()
inclusive_scan()
inner_product()
inplace_merge()
iota()
is_heap()
is_heap_until()
is_partitioned()
is_permutation()
is_sorted()
is_sorted_until()
lower_bound()
lexicographical_
compare()
max_element()
merge()
min_element()
minmax_element()
mismatch()
next_permutation()
move()
none_of()
nth_element()
partial_sum()
partial_sort()
partial_sort_copy()

partition()
partition_copy()
partition_point()
prev_permutation()
random_shuffle()
reduce()
remove()
remove_copy()
remove_copy_if()
remove_if()
replace()
replace_if()
replace_copy()
replace_copy_if()
reverse()
reverse_copy()
rotate()
rotate_copy()
search()
search_n()
set_difference()
set_intersection()
set_symmetric_
difference()
set_union()
sort()

stable_partition()
stable_sort()
swap_ranges()
transform()
transform_exclusive_
scan()
transform_inclusive_
scan()
transform_reduce()
uninitialized_copy()
uninitialized_copy_n()
uninitialized_default_
construct()
uninitialized_default_
construct_n()
uninitialized_fill()
uninitialized_fill_n()
uninitialized_move()
uninitialized_move_n()
uninitialized_value_
construct()
uninitialized_value_
construct_n()
unique()
unique_copy()
upper_bound()

Sort Example

C++17

```
#include <vector>
#include <algorithm>
#include <execution>

// fill the vector with some data
std::vector<int> v{...};

std::sort(std::execution::par, v.begin(), v.end());
```

Intel's Parallel STL

```
#include <vector>
#include <pstl/algorithm>
#include <pstl/execution>

// fill the vector with some data
std::vector<int> v{...};

std::sort(pstl::execution::par, v.begin(), v.end());
```

Sort Example

C++17

```
#include <vector>
#include <algorithm>
#include <execution>

// fill the vector with some data
std::vector<int> v{...};

std::sort(std::execution::par, v.begin(), v.end());
```

Intel's Parallel STL

```
#include <vector>
#include <pstl/algorithm>
#include <pstl/execution>

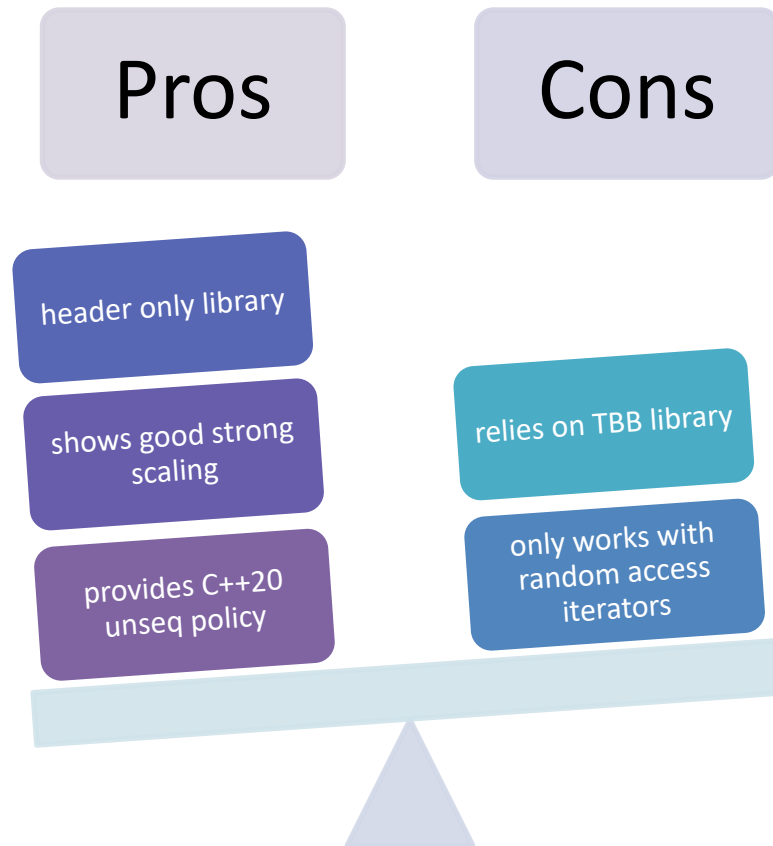
// fill the vector with some data
std::vector<int> v{...};

std::sort(pstl::execution::par, v.begin(), v.end());
```

Execution policies for Intel's Parallel STL:

- pstl::execution::seq
- pstl::execution::unseq
- pstl::execution::par
- pstl::execution::par_unseq

Intel's Parallel STL – Pros & Cons



Parallel STL Tutorial

Available on SHARCNET GitLab

[https://git.sharcnet.ca/asobhani/parallelstl tutorial](https://git.sharcnet.ca/asobhani/parallelstl_tutorial)

Benchmark Results

Algorithm Selection

- `sort()`
- Functional programming (Data Science)
 - `map` -> `transform()`, `transform_reduce()`
 - `filter` -> `remove_if()`
 - `reduce` -> `accumulate()`, `reduce()`

map, filter, and reduce explained with emoji 🤔

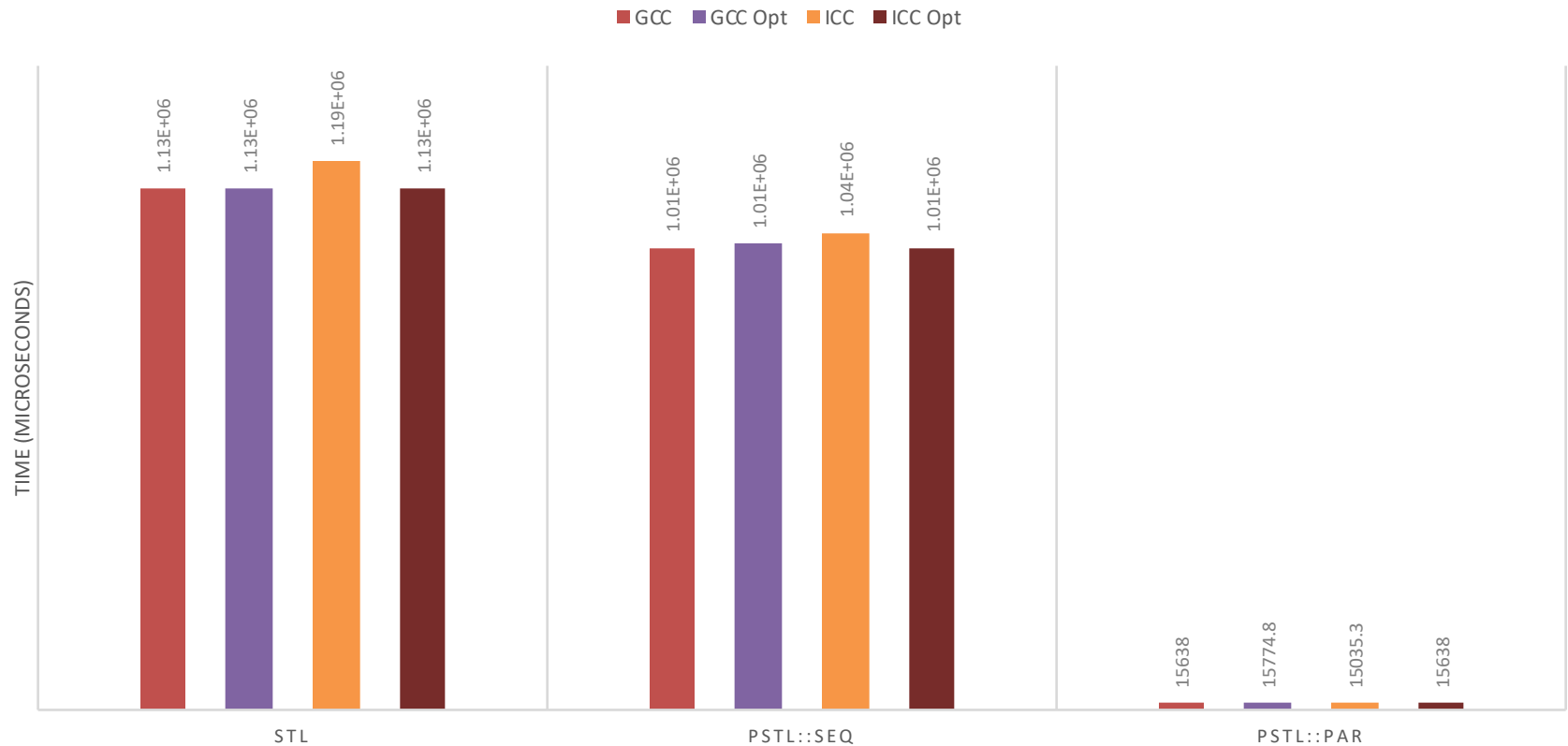
```
map([🐮, 🍌, 🐔, 🌽], cook)  
=> [🍔, 🍟, 🍗, 🍿]
```

```
filter([🍔, 🍟, 🍗, 🍿], isVegetarian)  
=> [🍟, 🍿]
```

```
reduce([🍔, 🍟, 🍗, 🍿], eat)  
=> 🤩
```

sort()

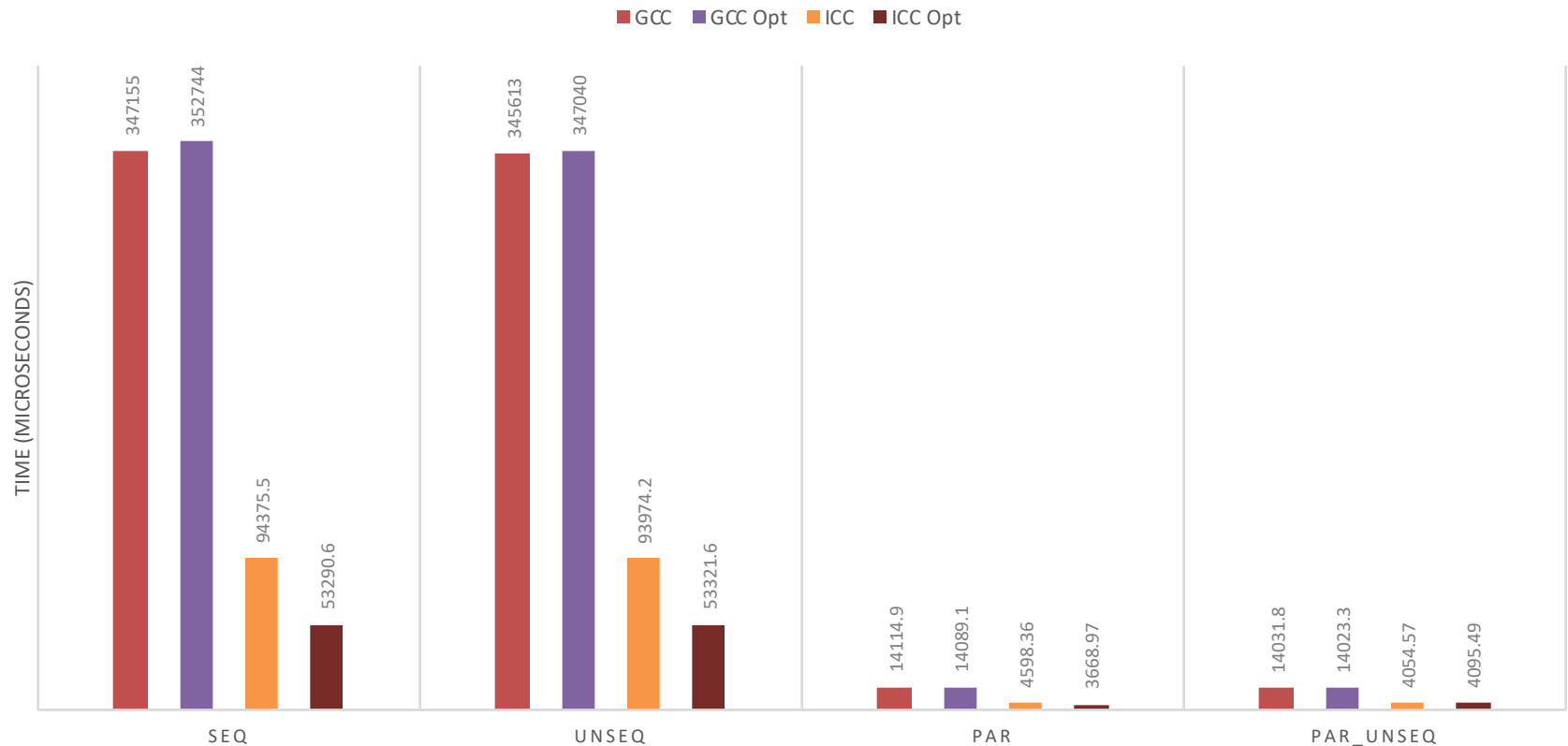
32 x 3 GHz Xeon – 10M float



Parallel Scaling: GCC=64 ICC=70

transform()

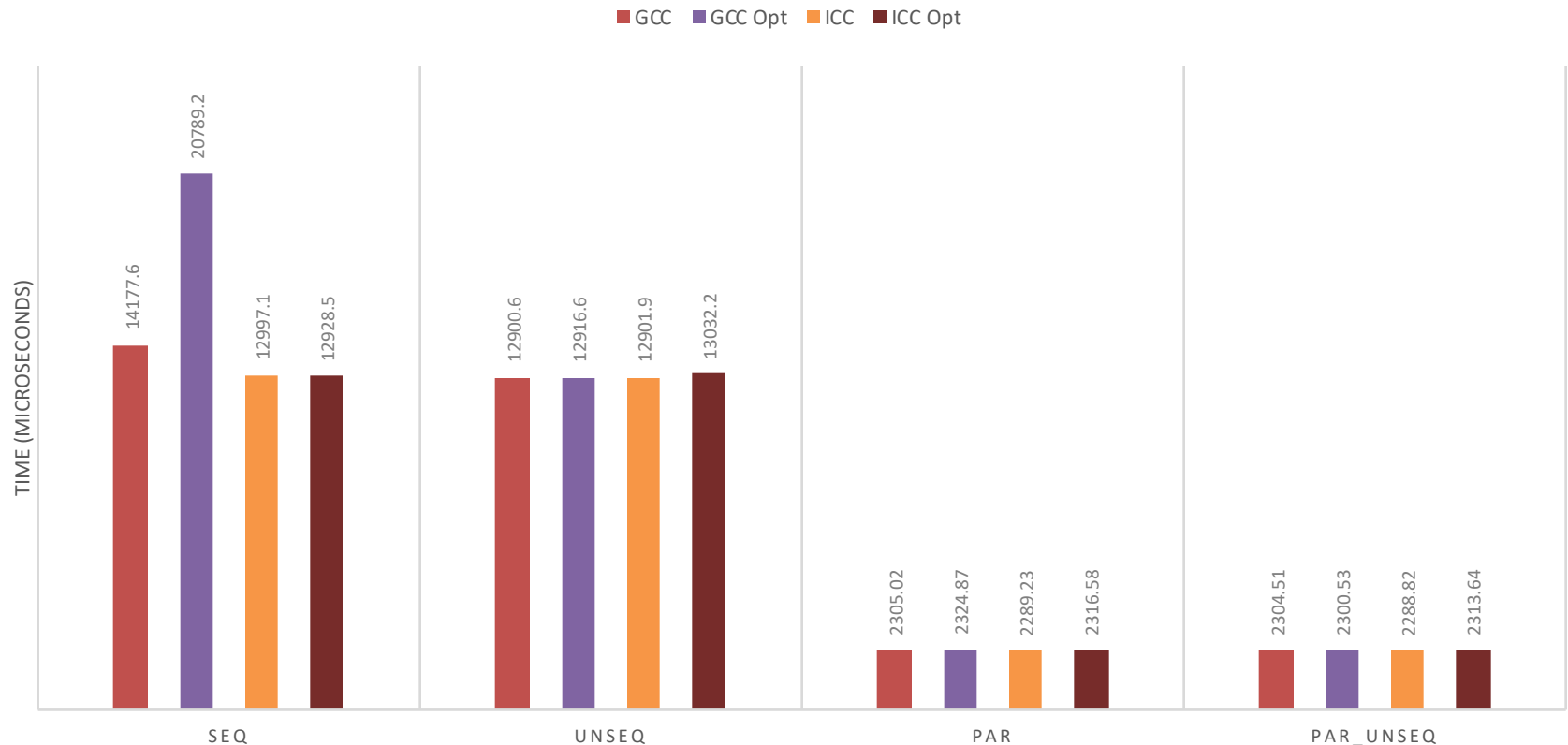
32 x 3 GHz Xeon – 10M double float



Parallel Scaling: GCC=25 ICC=15

transform_reduce() (Dot Product)

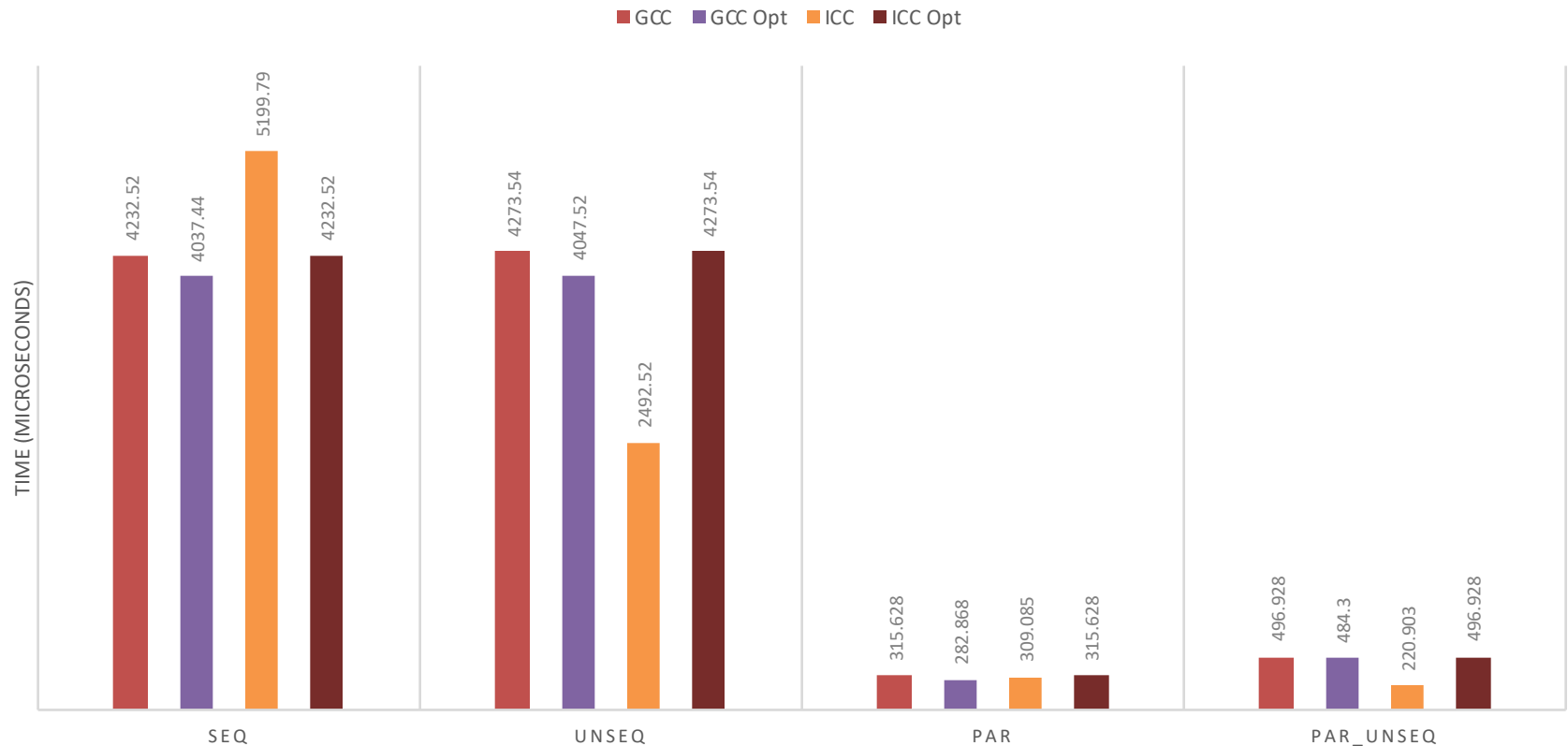
32 x 3 GHz Xeon – 10M double float



Parallel Scaling: GCC=6 ICC=6

remove_if()

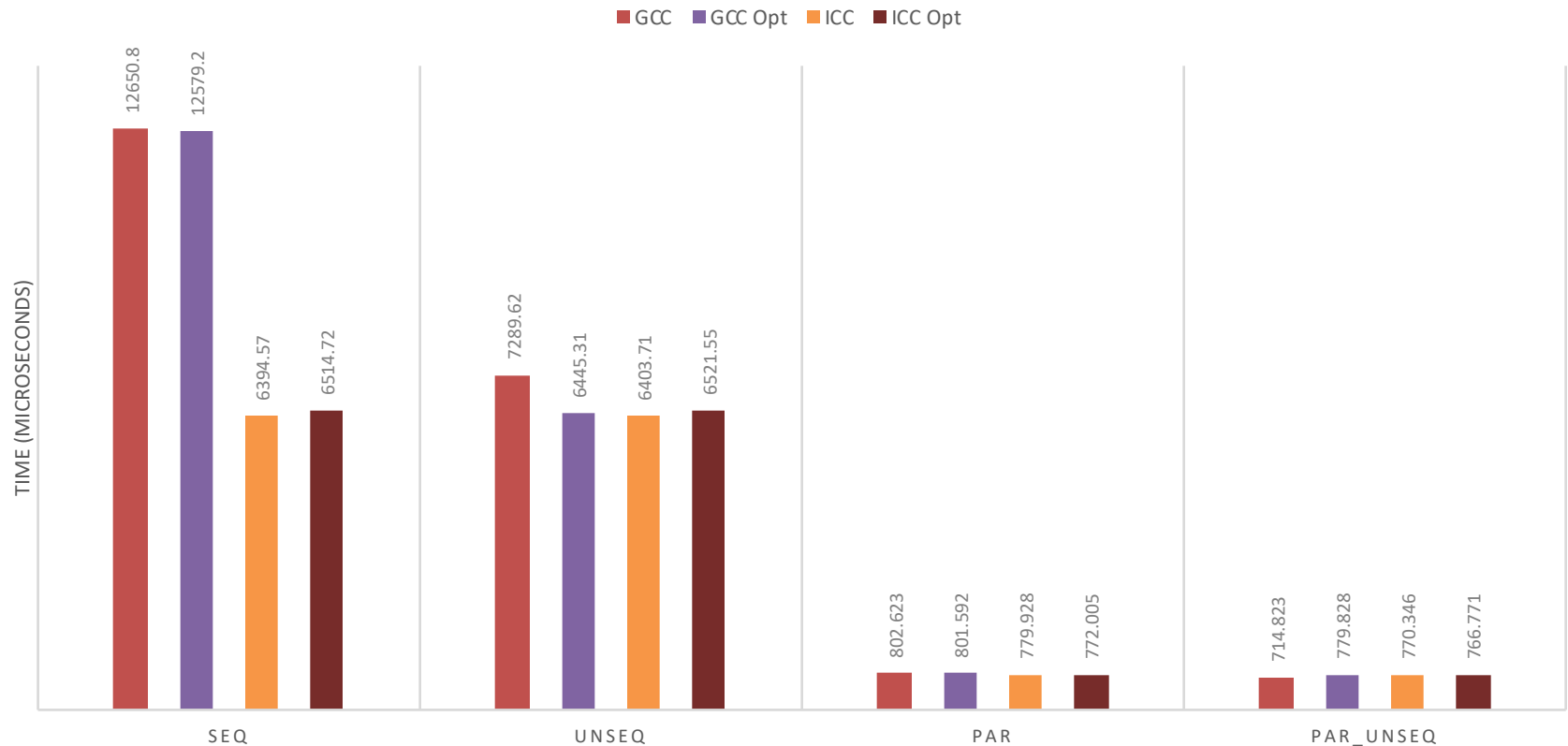
32 x 3 GHz Xeon – 10M integers



Parallel Scaling: GCC=14 ICC=9

reduce()

32 x 3 GHz Xeon – 10M double float



Parallel Scaling: GCC=16 ICC=8

More to Come...

- Adding benchmarks for all Parallel STL algorithms
- Including **strong scaling** results for all the Parallel STL algorithms
- Adding benchmarks for **STE||AR Group's HPX** library parallel algorithms