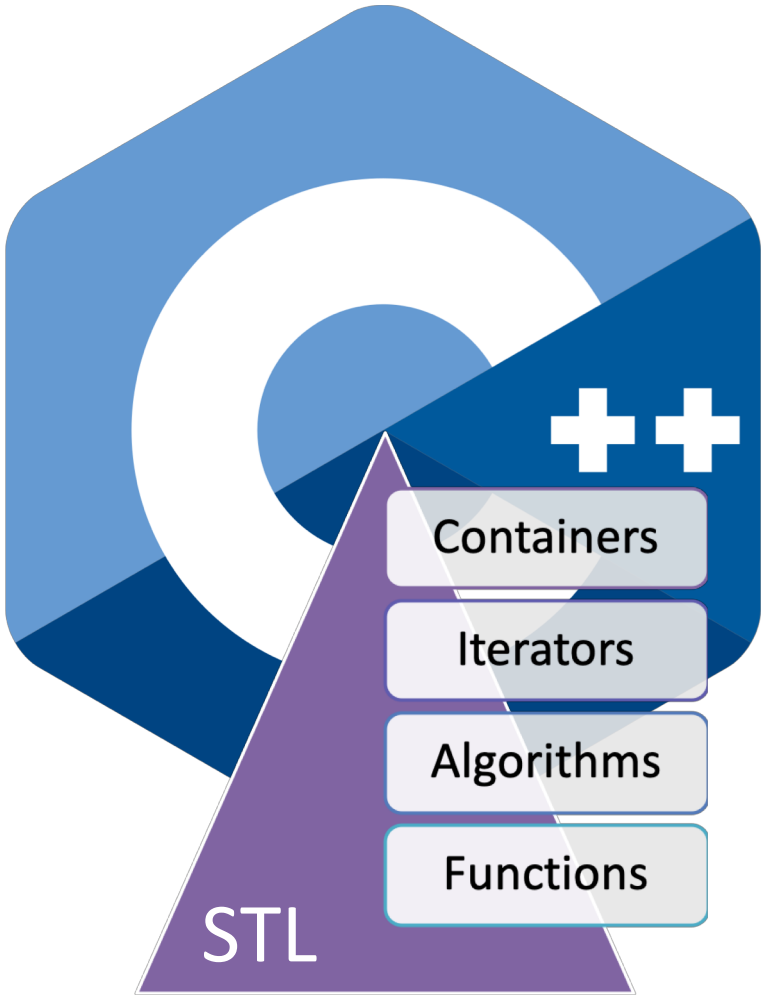
Dipping into C++17 Parallel Algorithms with Intel's Parallel STL

Armin Sobhani
asobhani@sharcnet.ca

SHARCNET

University of Ontario Institute of Technology (UOIT)

February 27, 2019



SHARCNET™

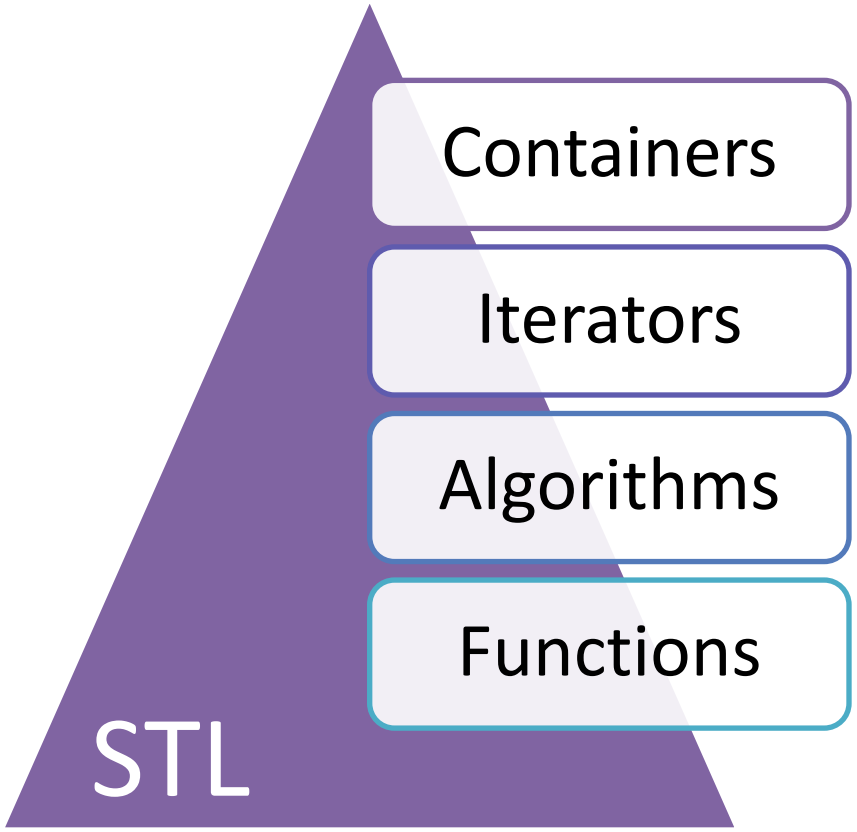
Outline

- 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

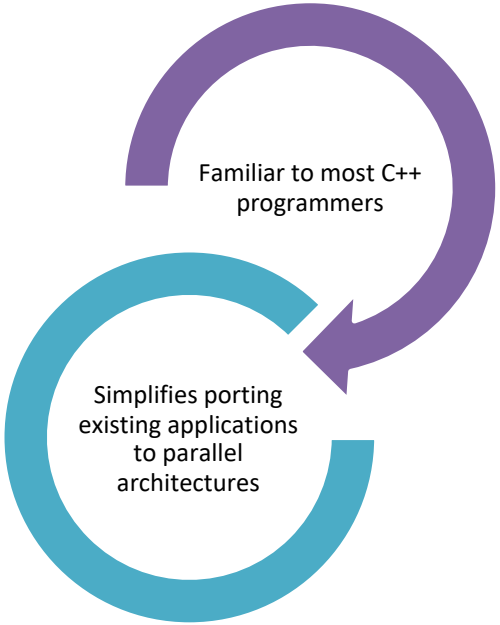
Standard Template Library (STL)

- 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

SHARCNET™

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

- 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()

STL Algorithms – No Execution Policy

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()

STL Algorithms – Parallel by Nature

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

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

header-only template library

free (https://github.com/intel/parallelstl)

depends on Intel TBB

will be used for implementation on GCC and Clang

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

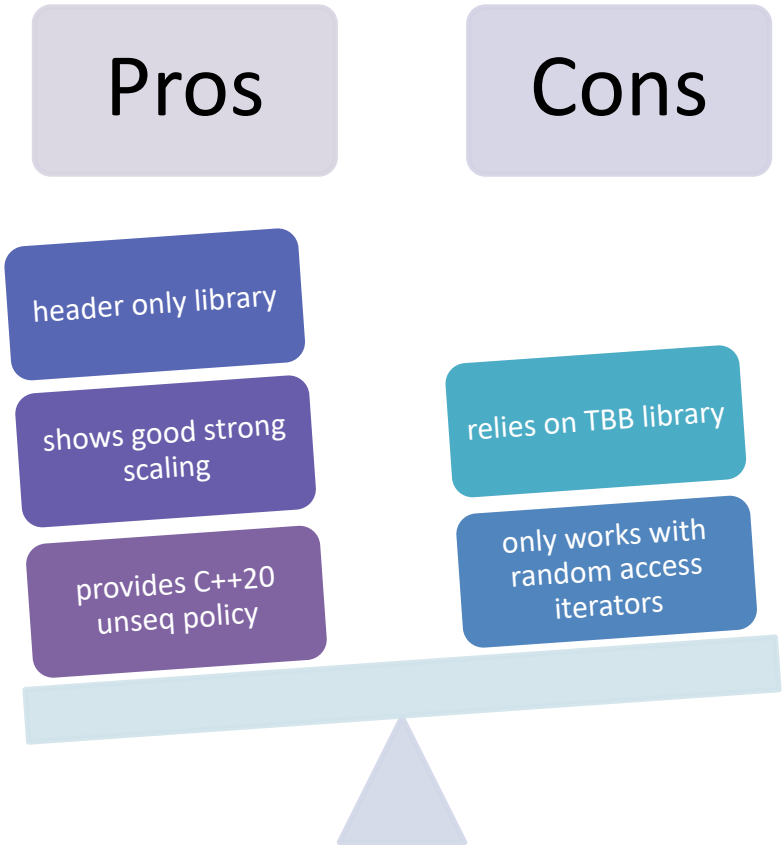
```
#include <vector>
#include <pstl/algorithm>
#include <pstl/execution>

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

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

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

Intel's Parallel STL – Pros & Cons



Parallel STL Tutorial

SHARCNET™

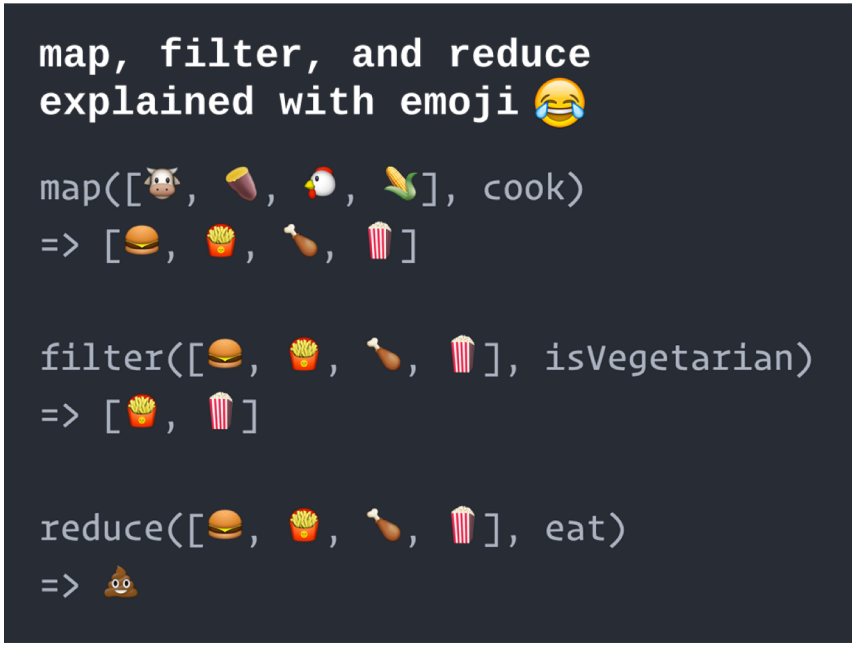
Available on SHARCNET GitLab

https://git.sharcnet.ca/asobhani/parallelstl_tutorial

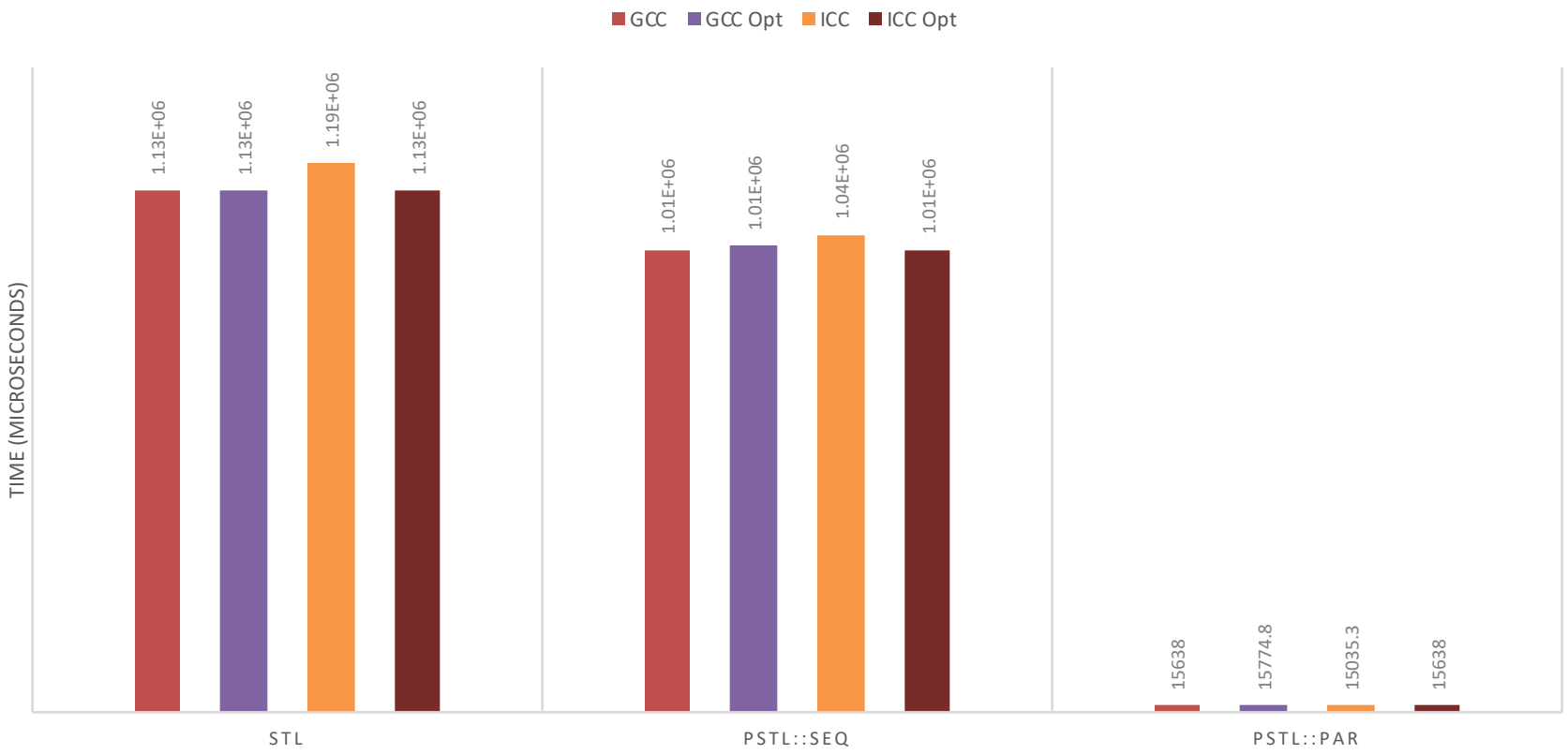
Benchmark Results

Algorithm Selection

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

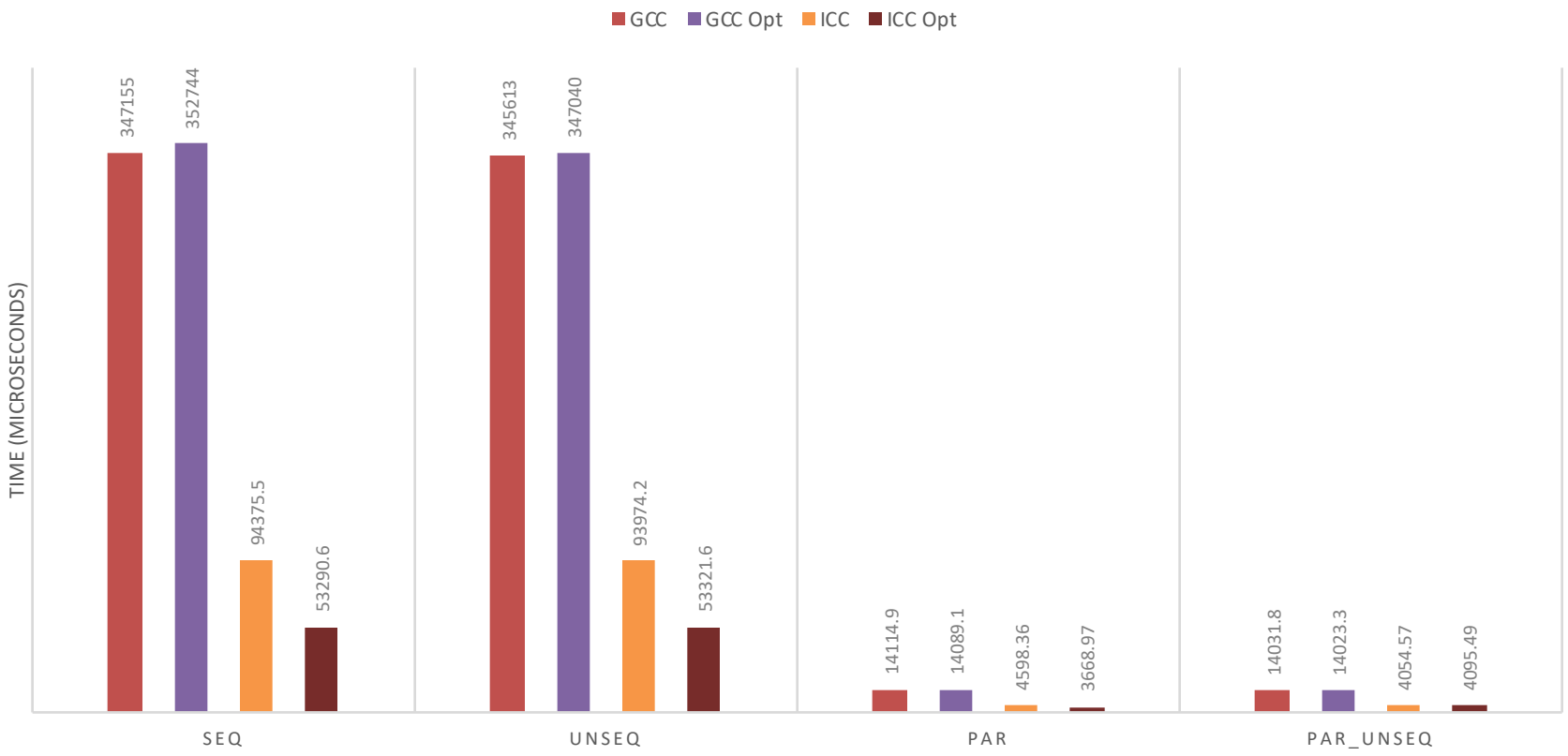


sort()
32 x 3 GHz Xeon – 10M float



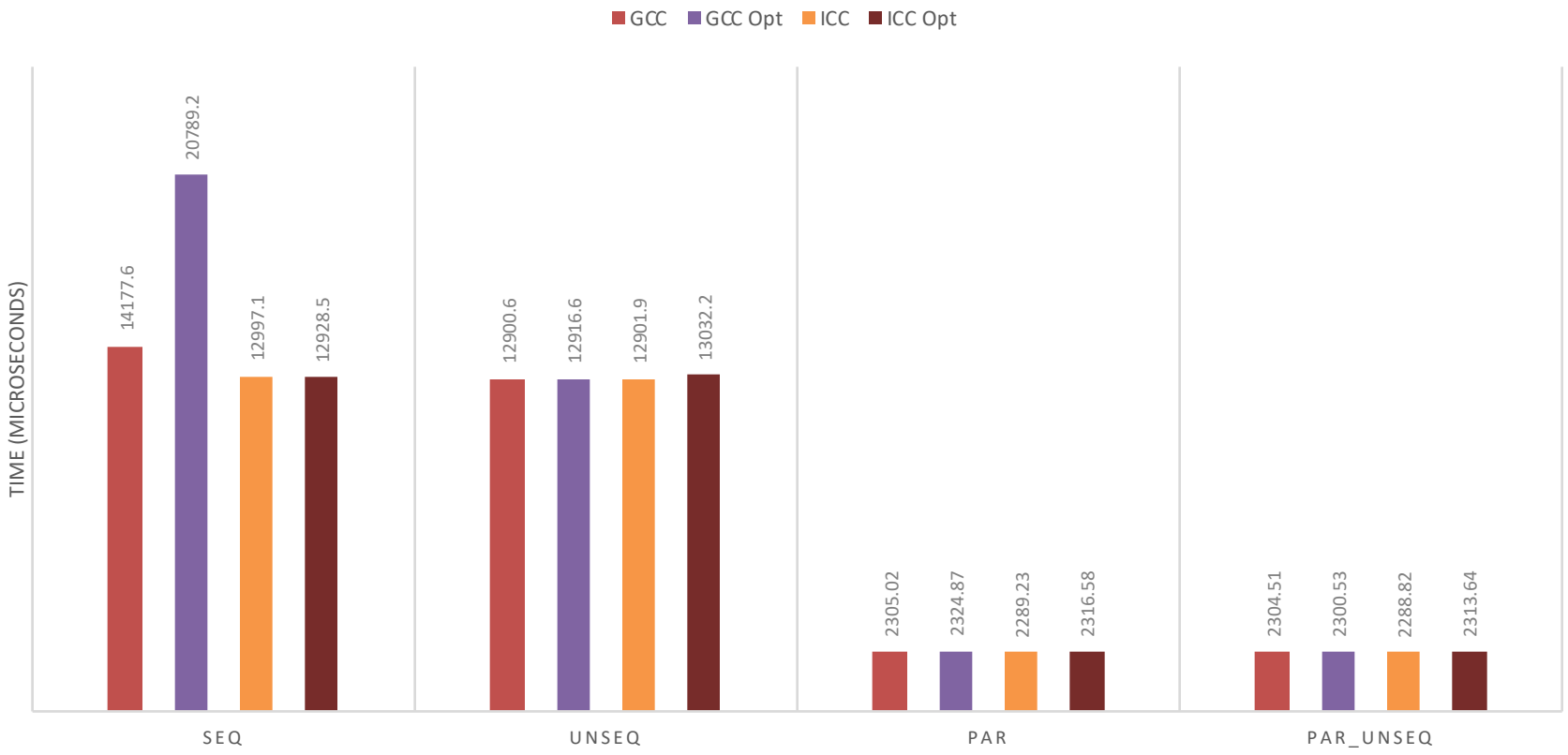
Parallel Scaling: GCC=64 ICC=70

transform()
32 x 3 GHz Xeon – 10M double float



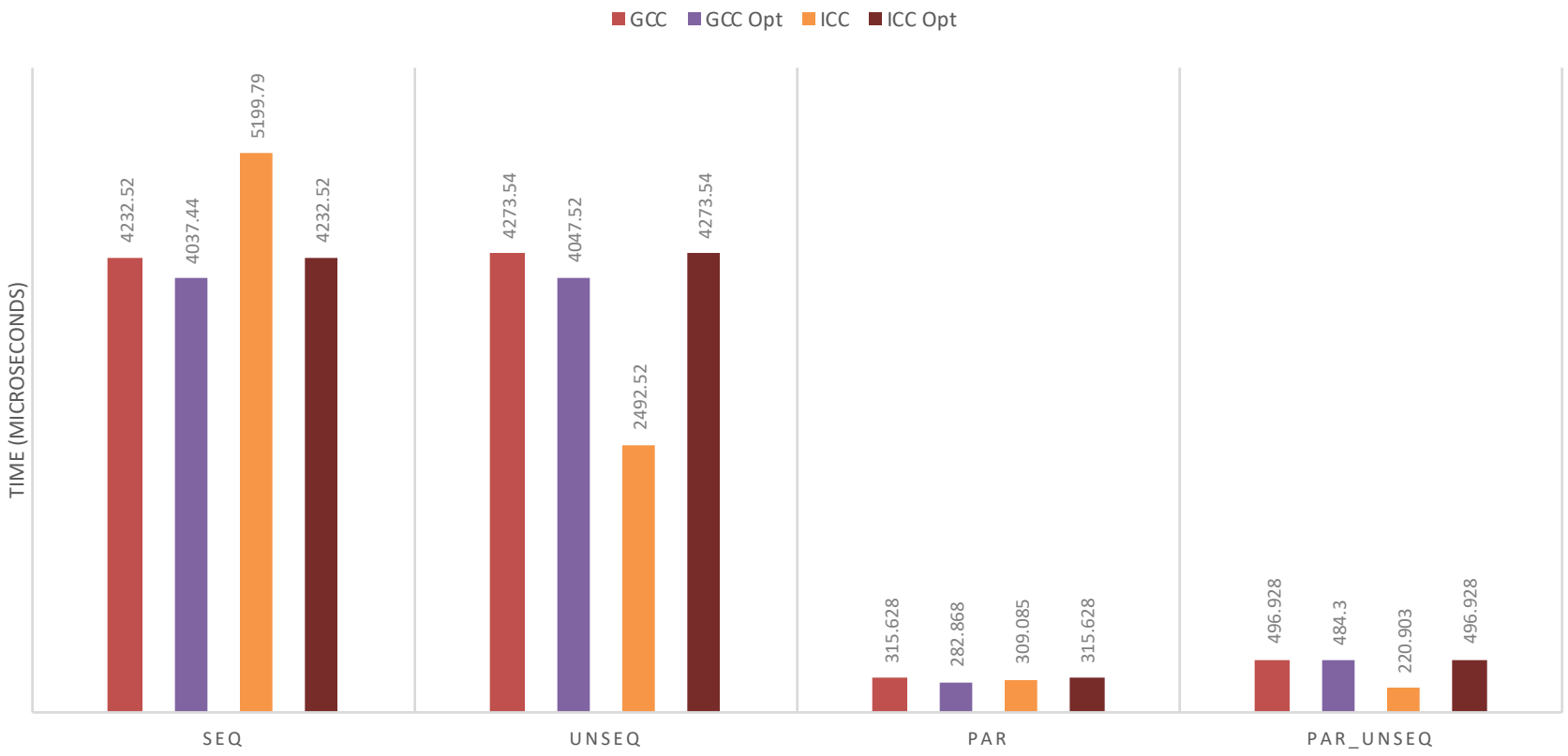
Parallel Scaling: GCC=25 ICC=15

transform_reduce() (Dot Product)
32 x 3 GHz Xeon – 10M double float



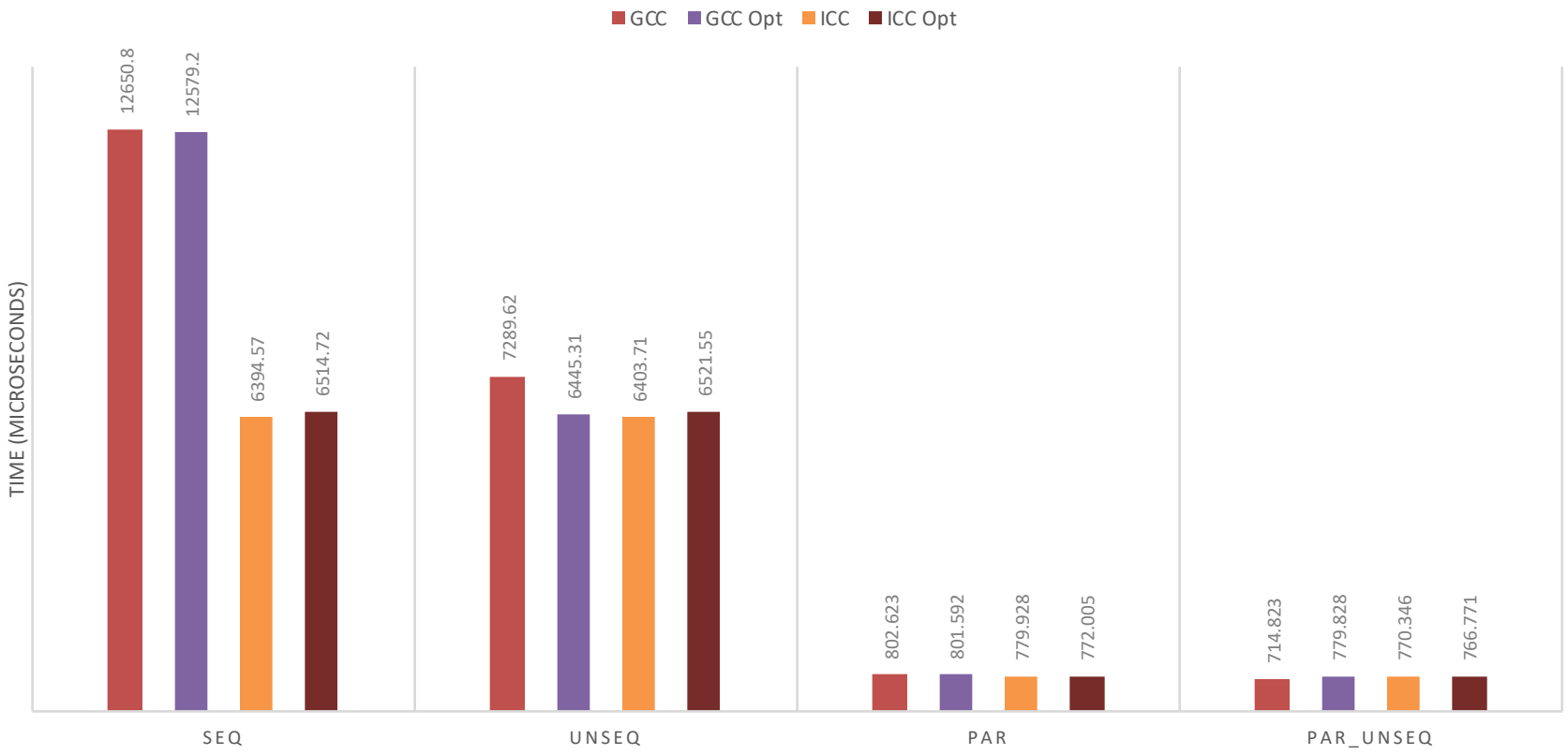
Parallel Scaling: GCC=6 ICC=6

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