# CSCE 435 Group project

## 0. Group number:

6

## 1. Group members:

1. Yida Zou
2. Brian Chen
3. Sam Hollenbeck
4. Alex Pantazopol

## 2. Project topic

We chose to use the suggested topic idea:

Choose 3+ parallel sorting algorithms, implement in MPI and CUDA. Examine and compare performance in detail (computation time, communication time, how much data is sent) on a variety of inputs: sorted, random, reverse, sorted with 1% perturbed, etc. Strong scaling, weak scaling, GPU performance.

## 2. Project description

We will communicate via iMessage.

Algorithms we will implement:

- Sample Sort (MPI)
- Sample Sort (CUDA)
- Merge Sort (MPI)
- Merge Sort (CUDA)
- Bitonic Sort (MPI)
- Bitonic Sort (CUDA)

We will compare the performance of these three algorithms with the metrics stated in the project topic.

To vary our algorithms, we will apply the following communication and parallelization strategies:

- fork/join parallelism
- point-to-point communication

# 3. Pseudocode

```
function SampleSort(unsortedList, t):  //t = thread count
    //Divide input into samples depending on number of threads
    sampleSize = calculateSampleSize(unsortedList, t)
    sample = selectSample(unsortedList, sampleSize)

    //Distribute the sample to all processors
    distribute(sample)  //Using MPI

    //Each thread sorts the given sample locally
    sortedSample = sortLocally(sample)    //sort using cudaMemcpy

    //Gather sorted samples
    sortedSamples = communicate(sample)    //Using MPI

    //Merge and sort all samples together
    sortedLists = mergeSublists(sortedSamples)

    return sortedSublist


function void merge_sort(arr, numThreads):

    # Initialize MPI environment
    MPI_Init()

    # Get the MPI rank and size
    rank = MPI_Comm_rank(MPI_COMM_WORLD)
    size = MPI_Comm_size(MPI_COMM_WORLD)

    # Divide and distribute data across nodes
    local_data = distribute_data(rank, size)

    # Sort local data using CUDA
    local_data = cuda_merge_sort(local_data)
```

```
    # Communicate data between nodes, and merge lists
    sorted_data = gather_and_merge(local_data, rank, size)

    # Finalize MPI environment
    MPI_Finalize()

    return sorted_data


function void bitonic_sort(arr, numThreads) {

    MPI_Init(...);
    rank = MPI_Comm_rank(...);
    size = MPI_Comm_size(...);

    MPI_Scatter(...);

    for (...) { // major step
        for (...) { // minor step
            sortOnGPU(...);
        }
    }

    MPI_Gather(...);
    MPI_Finalize();
    return sorted_data;
}
```

# 3. Evaluation Plan

We evaluated our sorting algorithm through the following:

Runtimes between parallel sorting algorithms on GPU and CPU (MPI vs CUDA)

Scaling the number of threads or processors

Scaling the problem size (length of array to sort)

Comparing different input types (Random, Sorted, Reverse Sorted, 1 % perturbed)

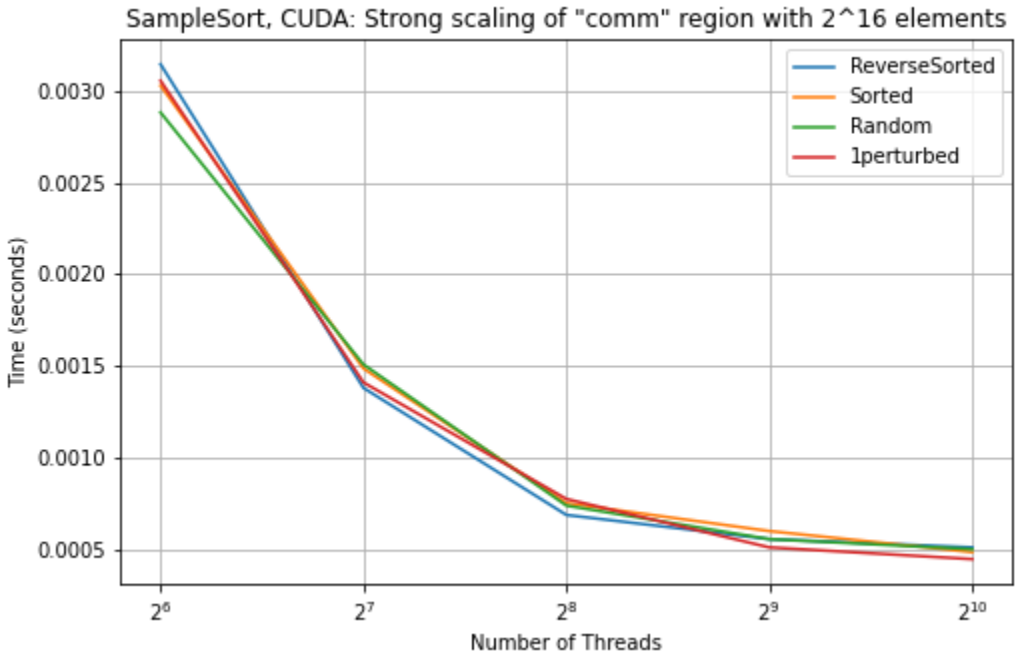
# 3. Project implementation

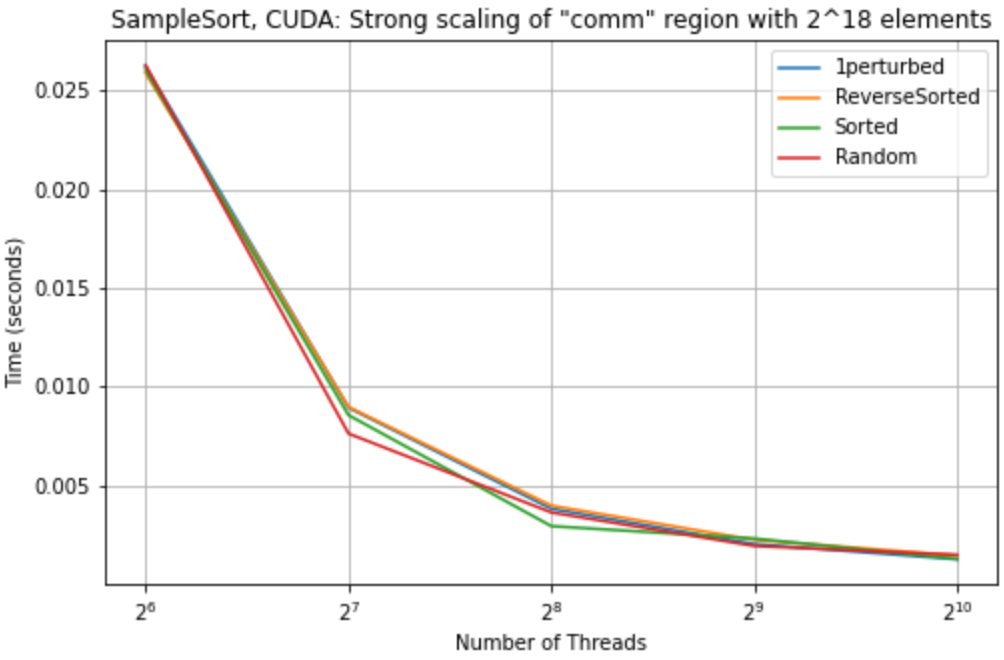
Our implementations can be found in their respective folders (BitonicSort, MergeSort, SampleSort) with each folder having subfolders for MPI and CUDA.

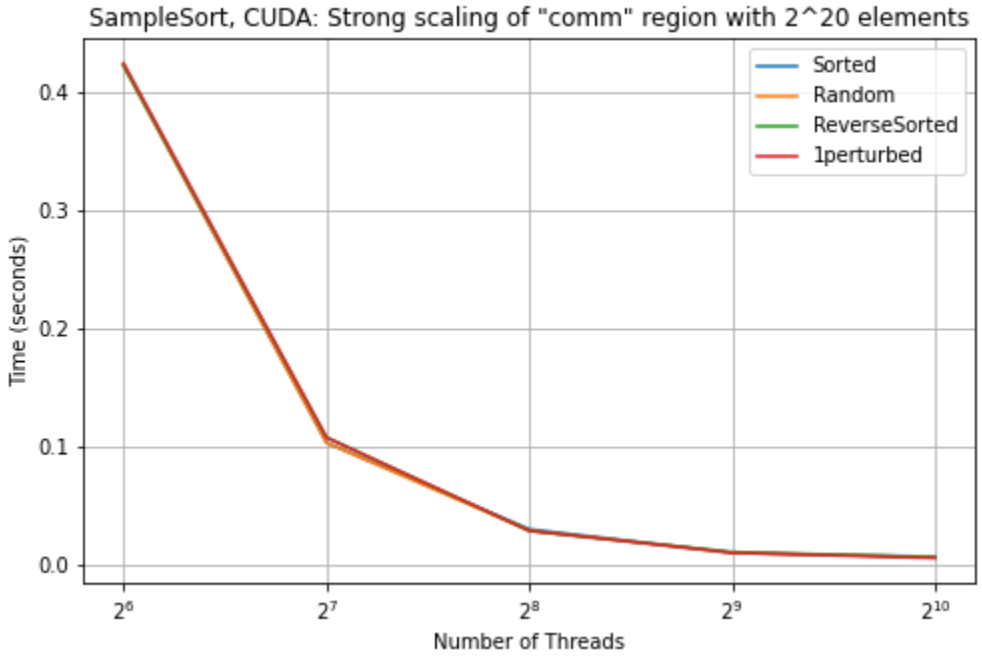
# 4. Performance evaluation

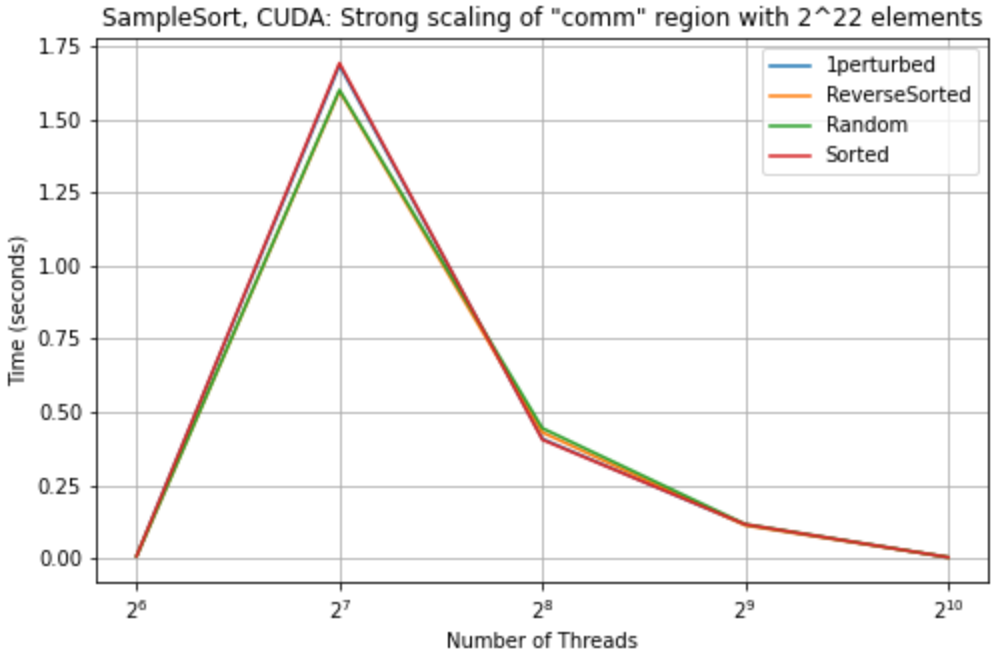
## Sample Sort CUDA

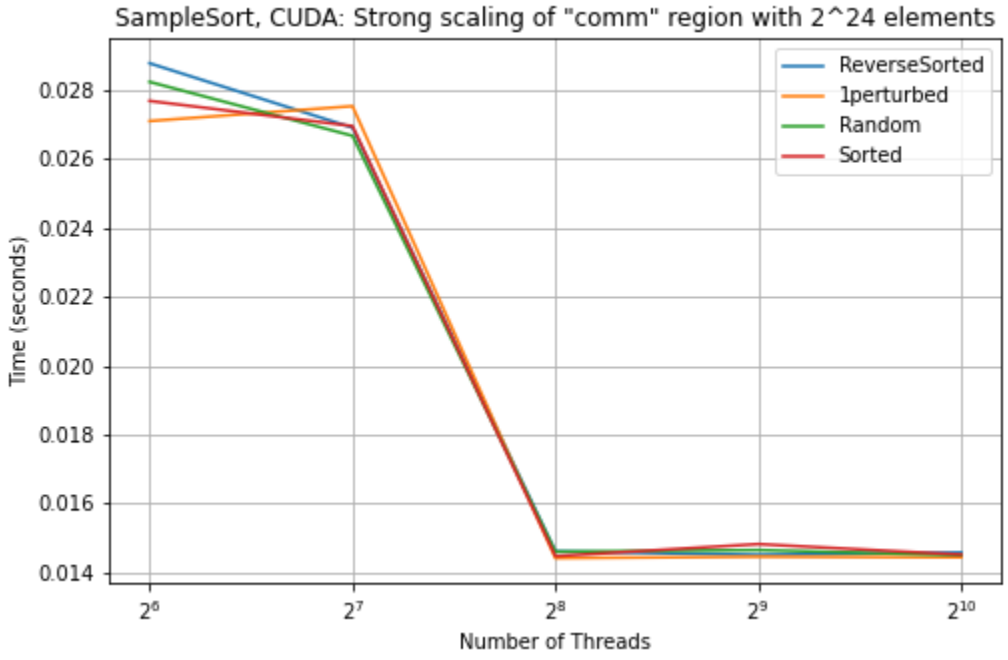
**Graphs**

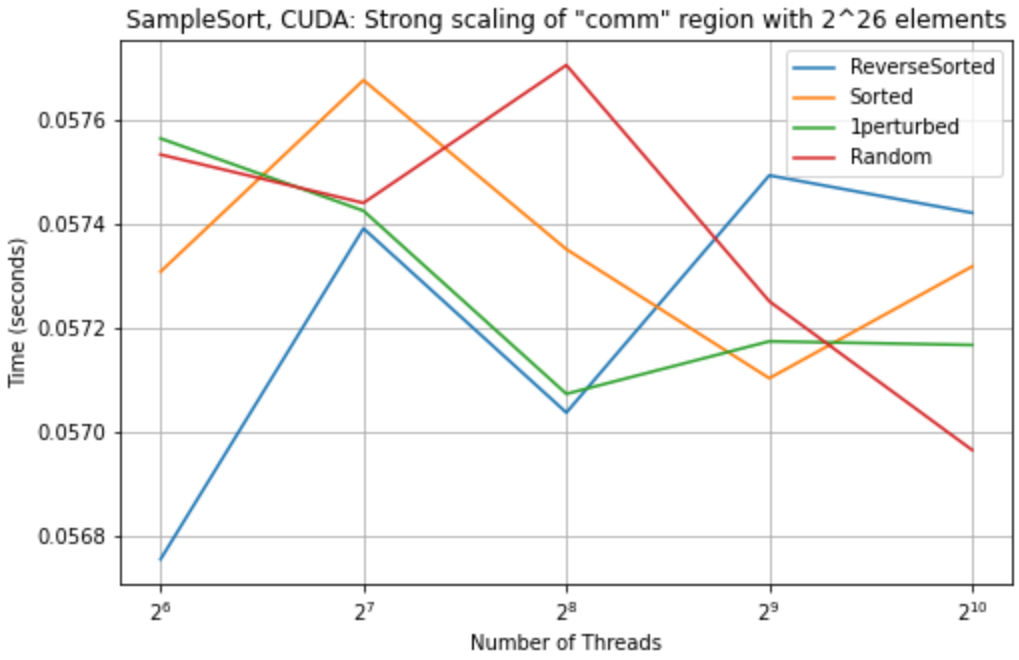


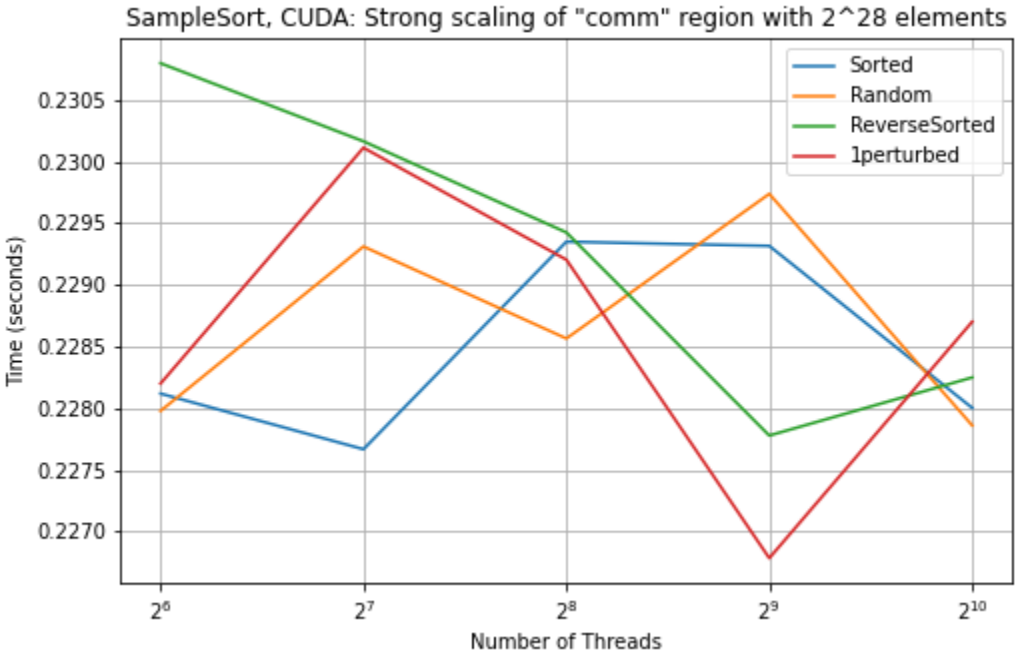

SampleSort, CUDA: Strong scaling of "comm" region with 2^18 elements
1perturbed
ReverseSorted
Sorted
Random
0.025
0.020
0.015
0.010
0.005
Time (seconds)
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

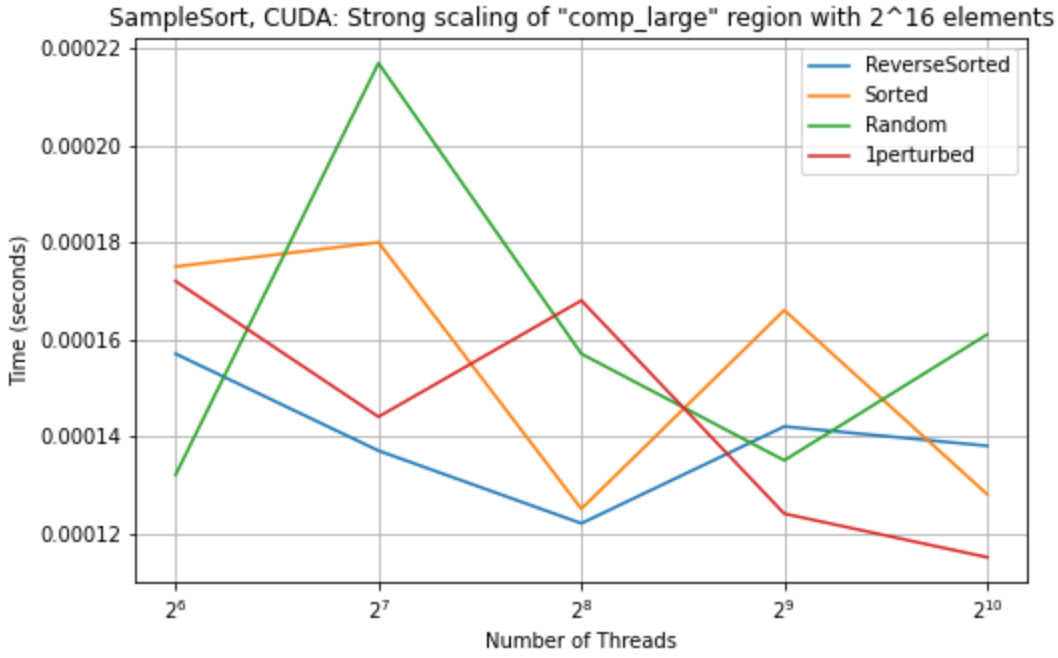

SampleSort, CUDA: Strong scaling of "comm" region with 2^20 elements
Sorted
Random
ReverseSorted
1perturbed
0.4
0.3
0.2
0.1
0.0
Time (seconds)
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

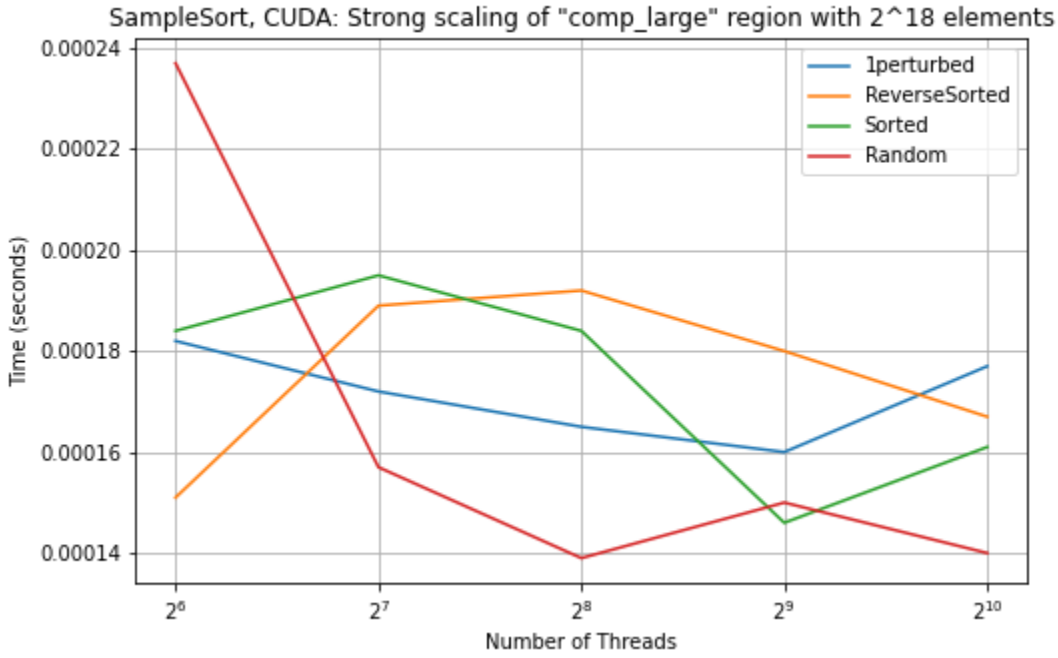

SampleSort, CUDA: Strong scaling of "comm" region with 2^22 elements
Time (seconds)
Number of Threads
1perturbed
ReverseSorted
Random
Sorted
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$

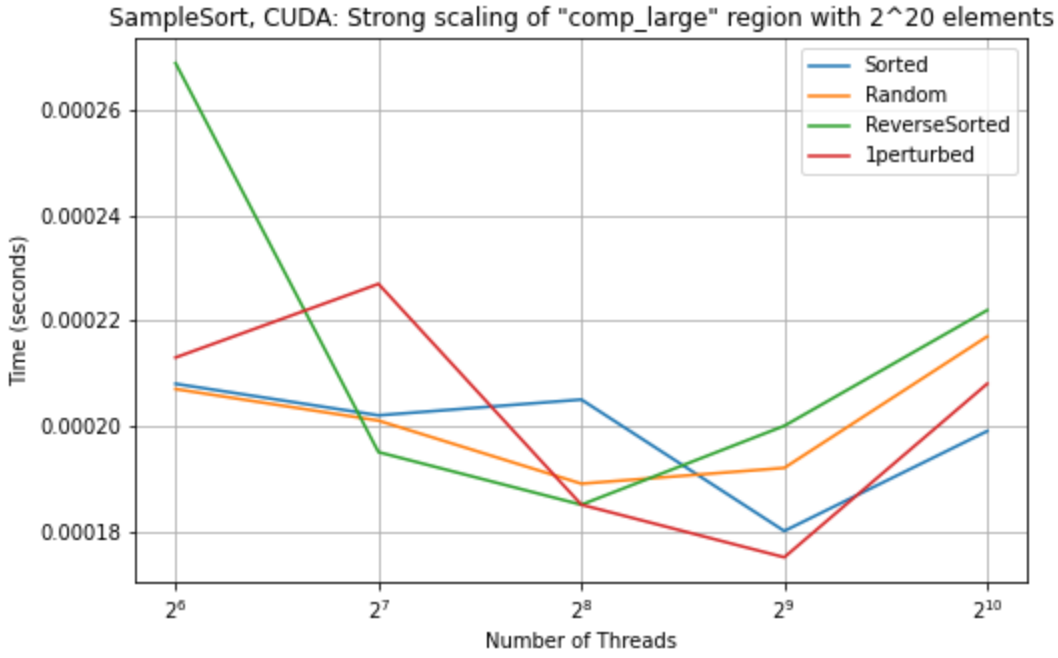

SampleSort, CUDA: Strong scaling of "comm" region with 2^24 elements
Time (seconds)
Number of Threads
ReverseSorted
1perturbed
Random
Sorted
0.028
0.026
0.024
0.022
0.020
0.018
0.016
0.014
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$

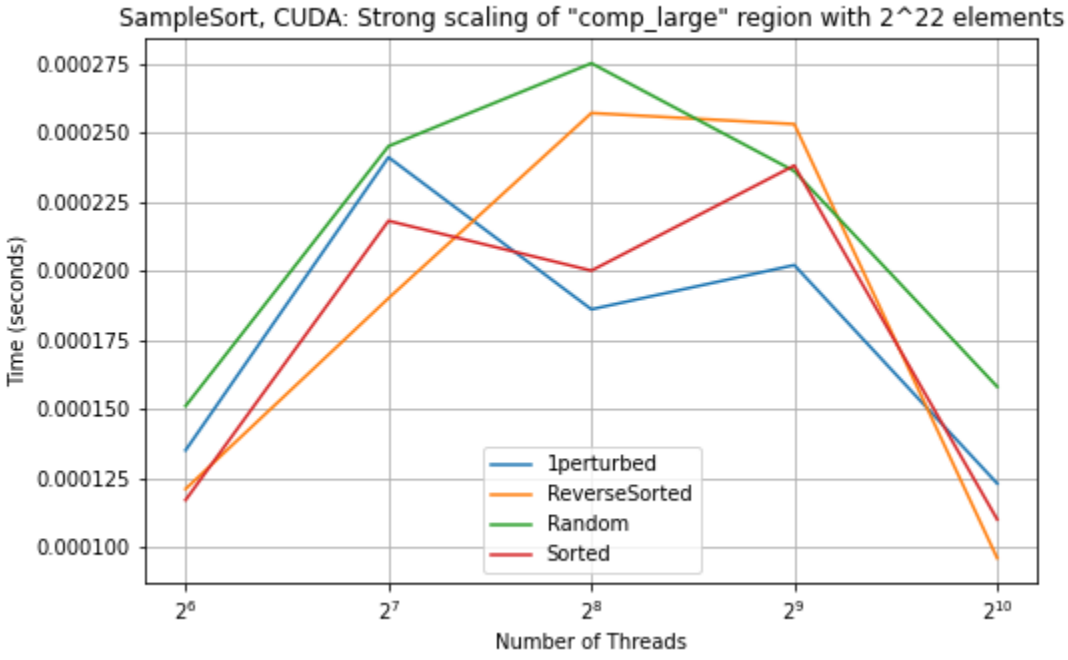

SampleSort, CUDA: Strong scaling of "comm" region with 2^26 elements
ReverseSorted
Sorted
1perturbed
Random
0.0576
0.0574
0.0572
0.0570
0.0568
Time (seconds)
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

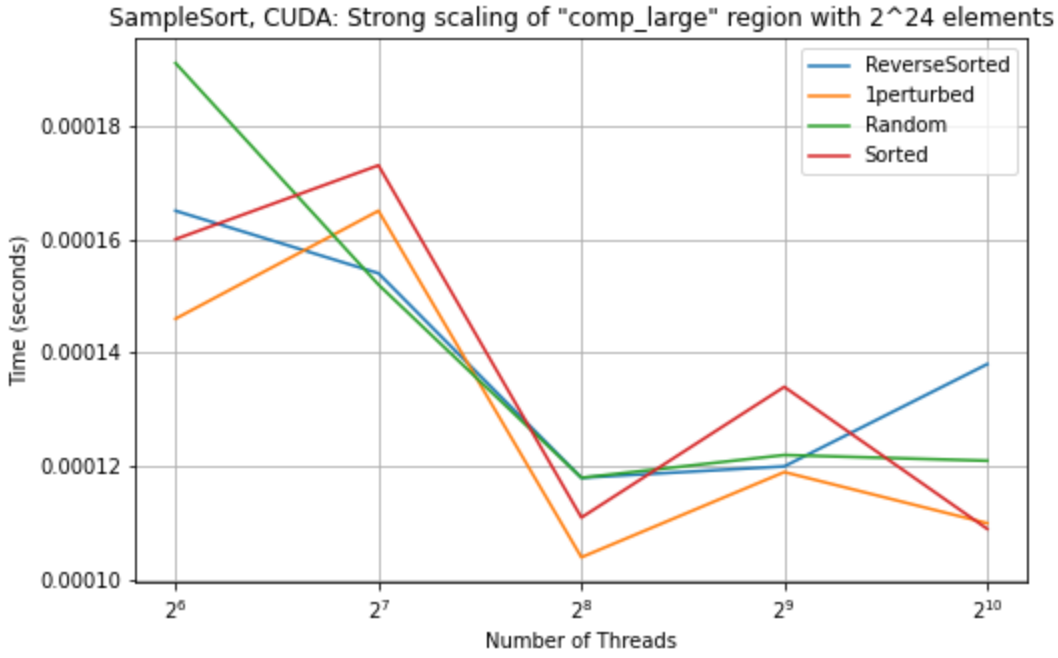

SampleSort, CUDA: Strong scaling of "comm" region with 2^28 elements
Sorted
Random
ReverseSorted
1perturbed
0.2305
0.2300
0.2295
0.2290
0.2285
0.2280
0.2275
0.2270
Time (seconds)
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

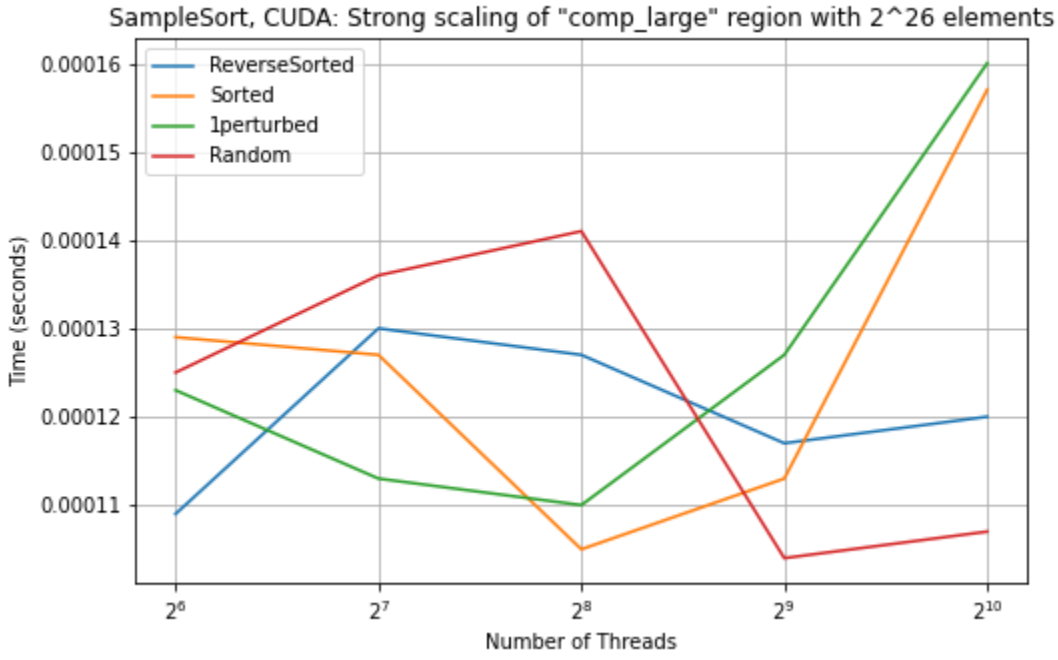

SampleSort, CUDA: Strong scaling of "comp_large" region with 2^16 elements
Time (seconds)
0.00022
0.00020
0.00018
0.00016
0.00014
0.00012
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
ReverseSorted
Sorted
Random
1perturbed

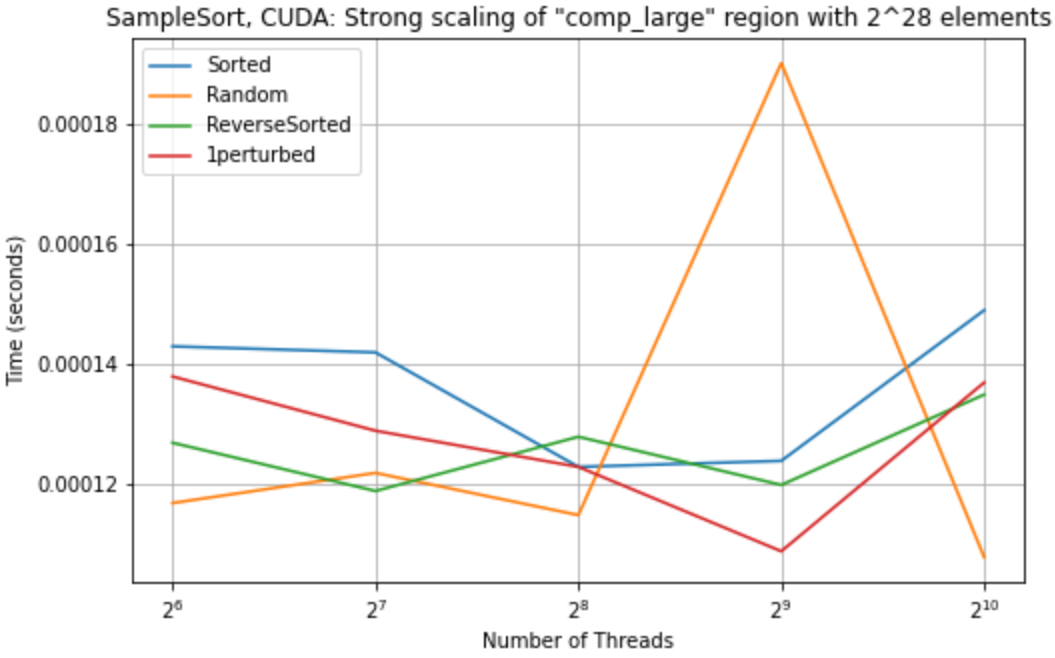

SampleSort, CUDA: Strong scaling of "comp_large" region with 2^18 elements
Time (seconds)
0.00024
0.00022
0.00020
0.00018
0.00016
0.00014
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
1perturbed
ReverseSorted
Sorted
Random

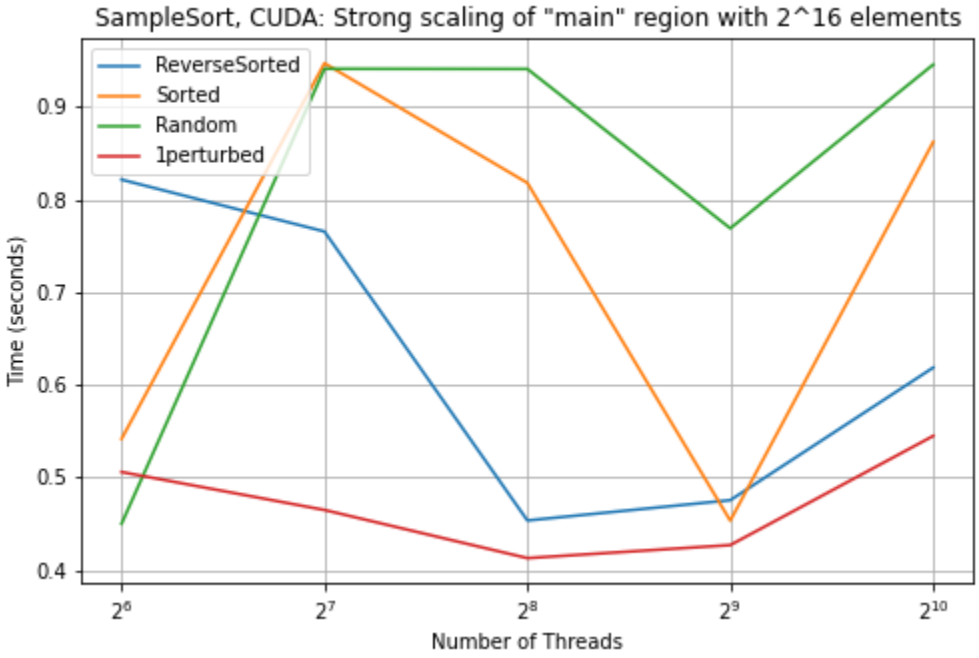

SampleSort, CUDA: Strong scaling of "comp_large" region with 2^20 elements
Time (seconds)
0.00026
0.00024
0.00022
0.00020
0.00018
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
Sorted
Random
ReverseSorted
1perturbed

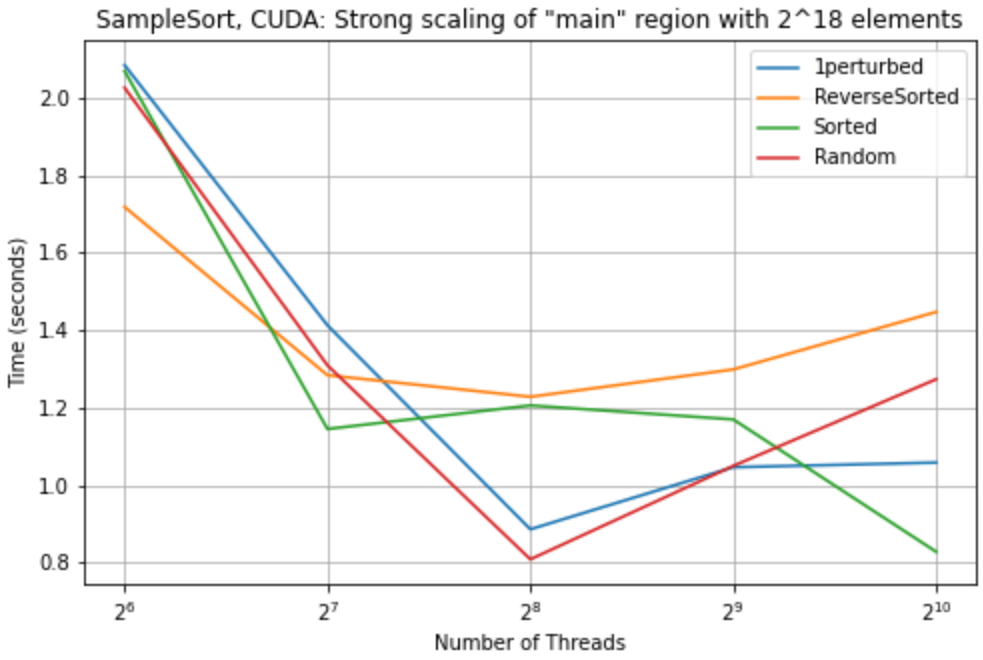

SampleSort, CUDA: Strong scaling of "comp_large" region with 2^22 elements
Time (seconds)
0.000275
0.000250
0.000225
0.000200
0.000175
0.000150
0.000125
0.000100
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
1perturbed
ReverseSorted
Random
Sorted

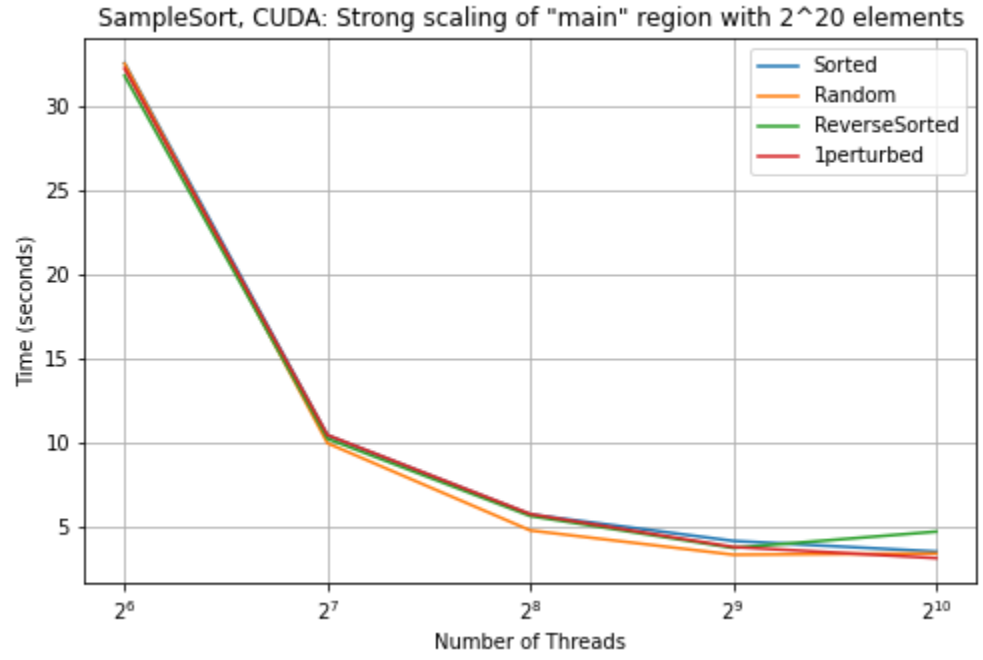

SampleSort, CUDA: Strong scaling of "comp_large" region with 2^24 elements
ReverseSorted
1perturbed
Random
Sorted
0.00018
0.00016
0.00014
0.00012
0.00010
Time (seconds)
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

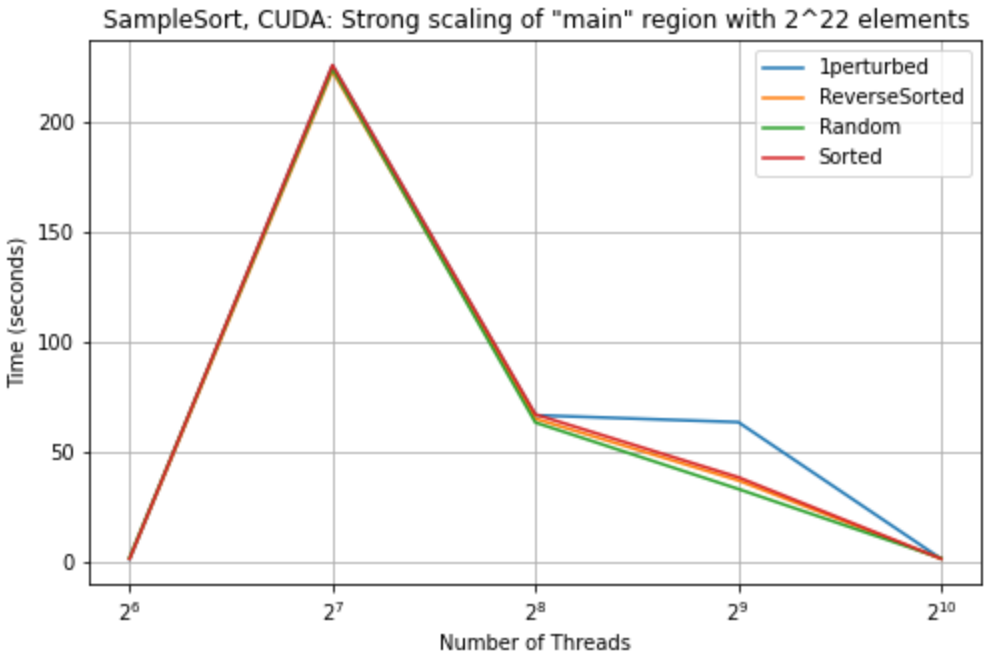

SampleSort, CUDA: Strong scaling of "comp_large" region with 2^26 elements
ReverseSorted
Sorted
1perturbed
Random
0.00016
0.00015
0.00014
0.00013
0.00012
0.00011
Time (seconds)
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

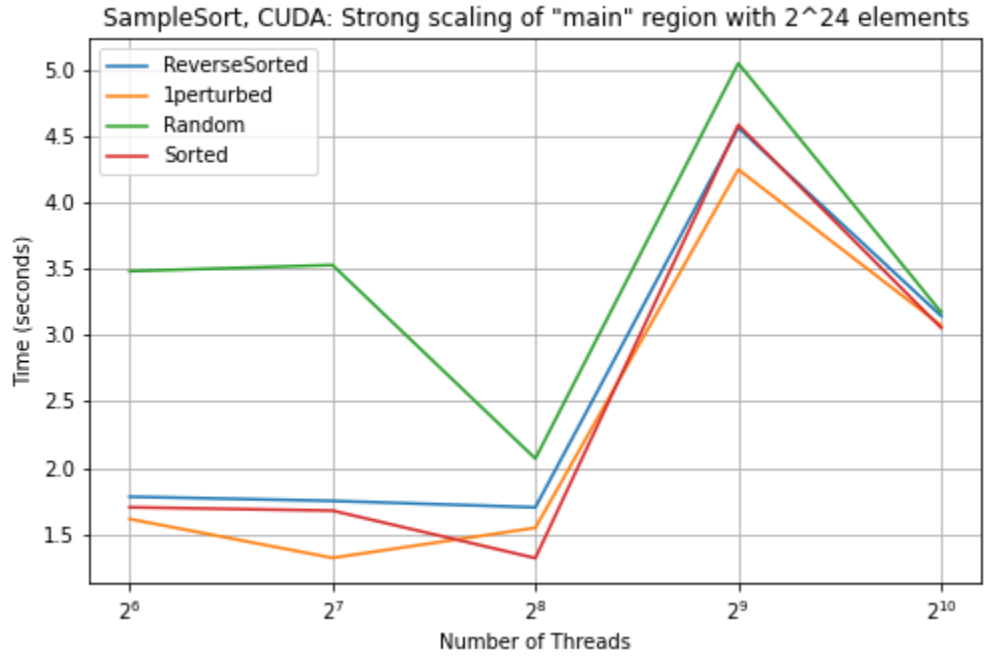

SampleSort, CUDA: Strong scaling of "comp_large" region with 2^28 elements
Sorted
Random
ReverseSorted
1perturbed
0.00018
0.00016
0.00014
0.00012
Time (seconds)
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

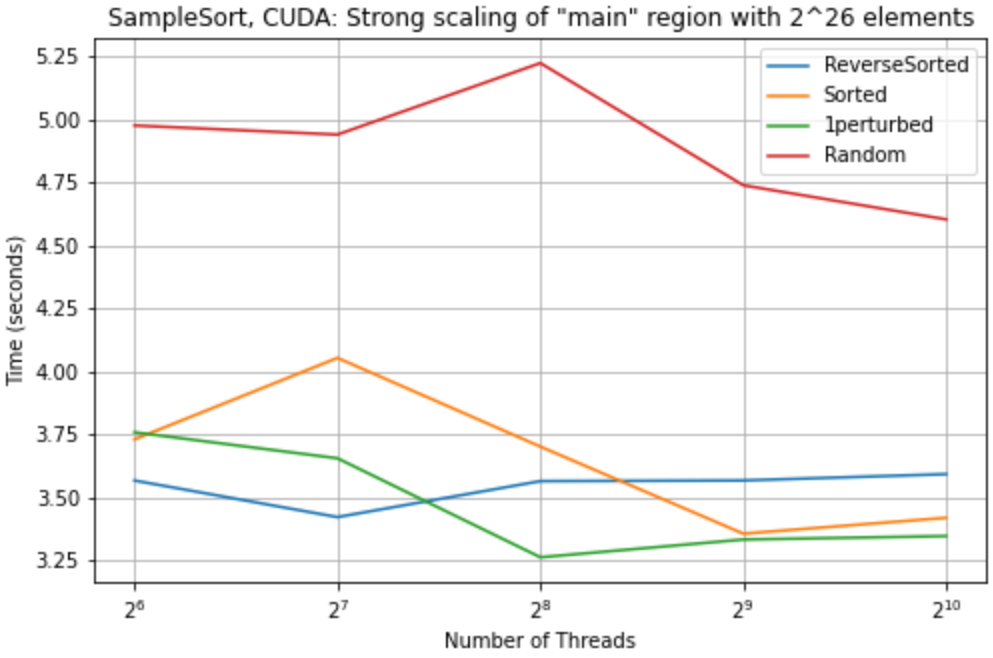

SampleSort, CUDA: Strong scaling of "main" region with 2^16 elements
ReverseSorted
Sorted
Random
1perturbed
0.9
0.8
0.7
0.6
0.5
0.4
Time (seconds)
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

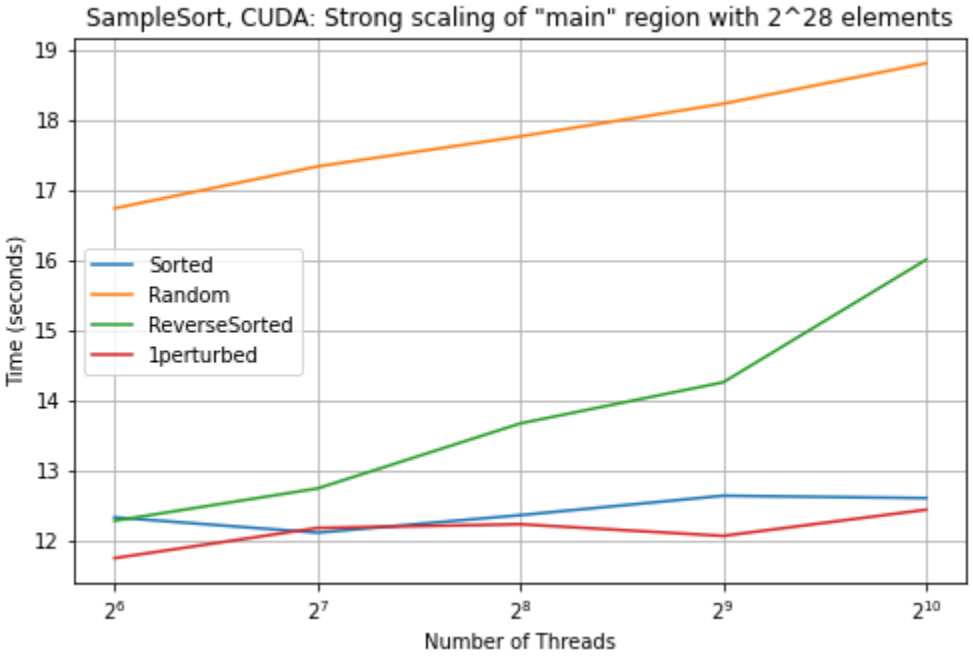

SampleSort, CUDA: Strong scaling of "main" region with 2^18 elements
1perturbed
ReverseSorted
Sorted
Random
Time (seconds)
2.0
1.8
1.6
1.4
1.2
1.0
0.8
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

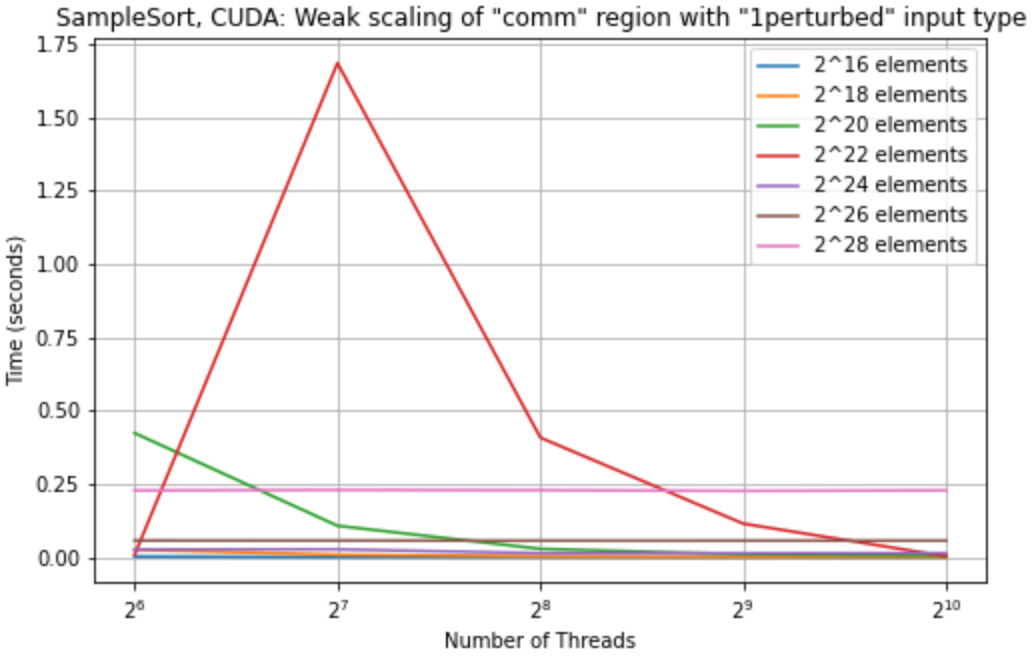

SampleSort, CUDA: Strong scaling of "main" region with 2^20 elements
Sorted
Random
ReverseSorted
1perturbed
Time (seconds)
30
25
20
15
10
5
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

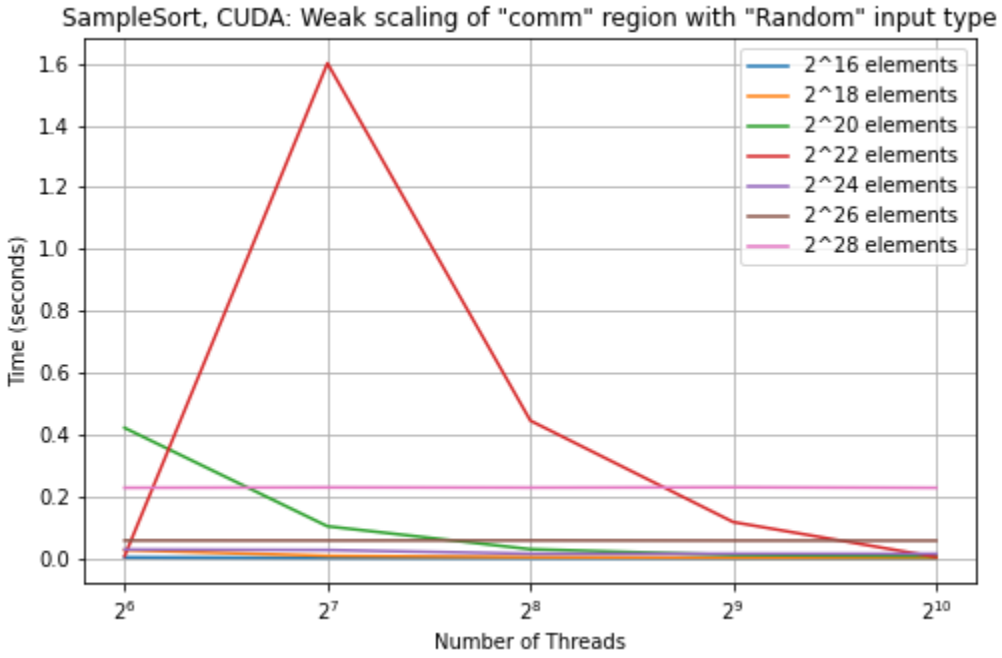

SampleSort, CUDA: Strong scaling of "main" region with 2^22 elements
1perturbed
ReverseSorted
Random
Sorted
Time (seconds)
0
50
100
150
200
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

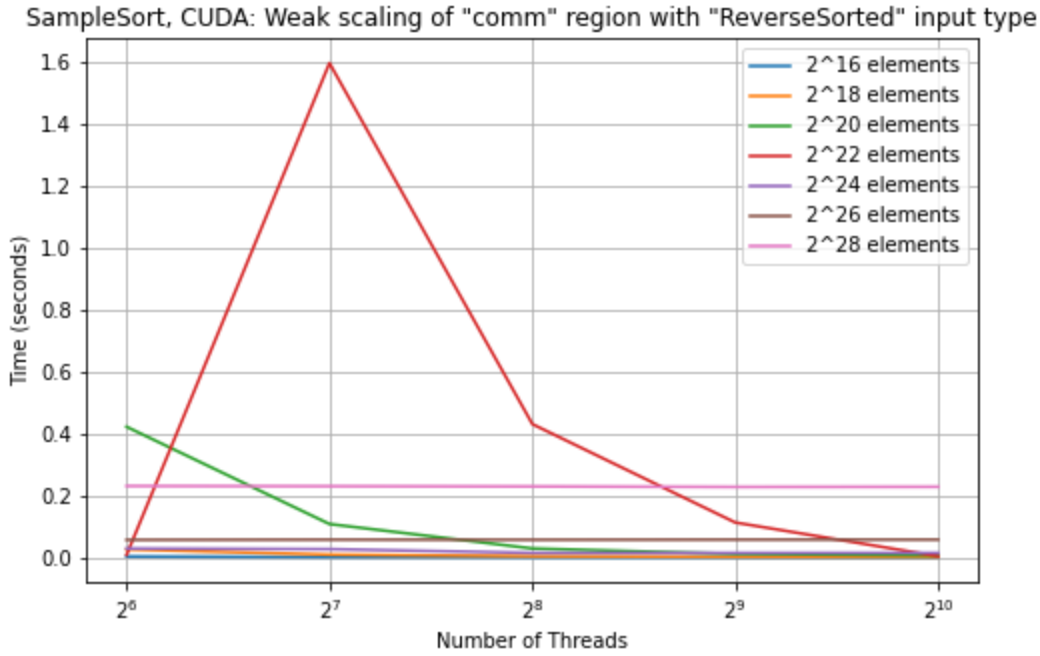

SampleSort, CUDA: Strong scaling of "main" region with 2^24 elements
ReverseSorted
1perturbed
Random
Sorted
Time (seconds)
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

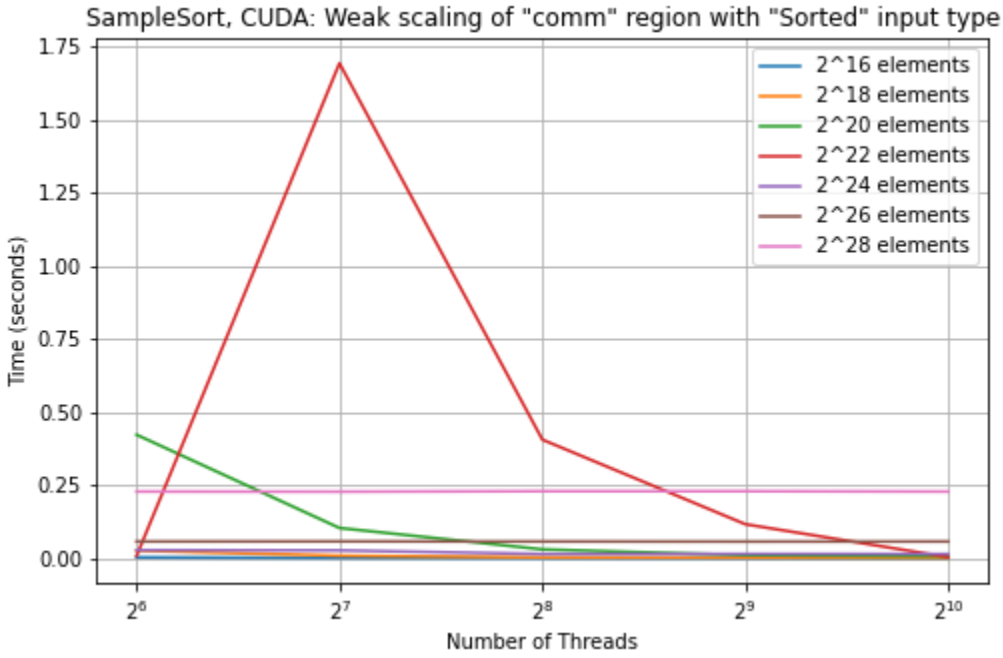

SampleSort, CUDA: Strong scaling of "main" region with 2^26 elements
ReverseSorted
Sorted
1perturbed
Random
Time (seconds)
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

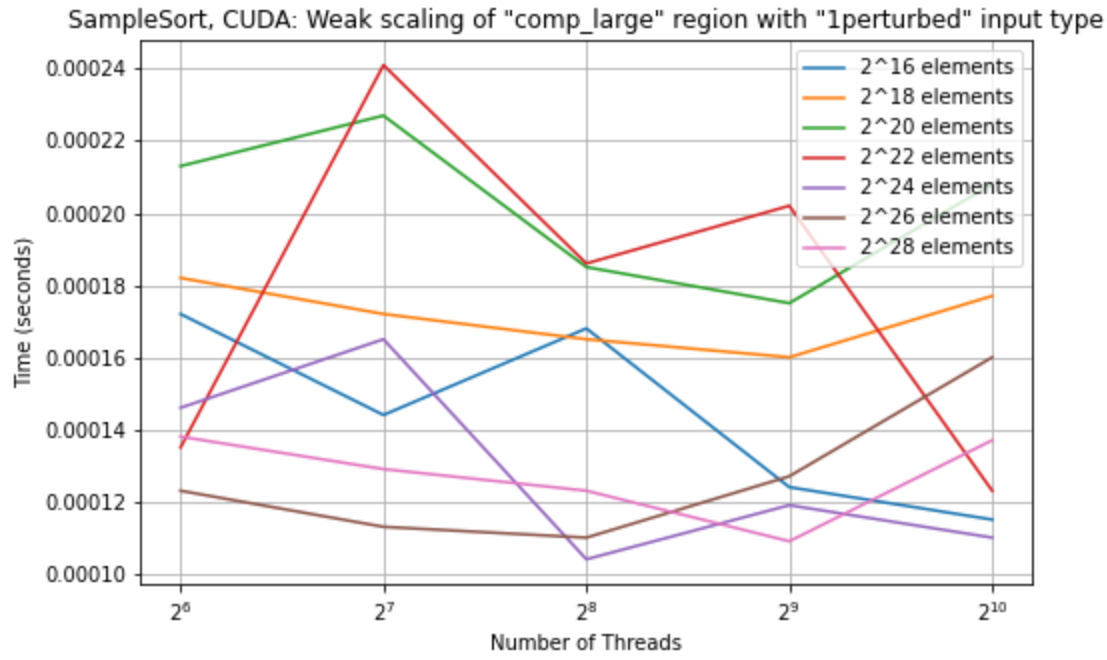

SampleSort, CUDA: Strong scaling of "main" region with 2^28 elements
Sorted
Random
ReverseSorted
1perturbed
Time (seconds)
19
18
17
16
15
14
13
12
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

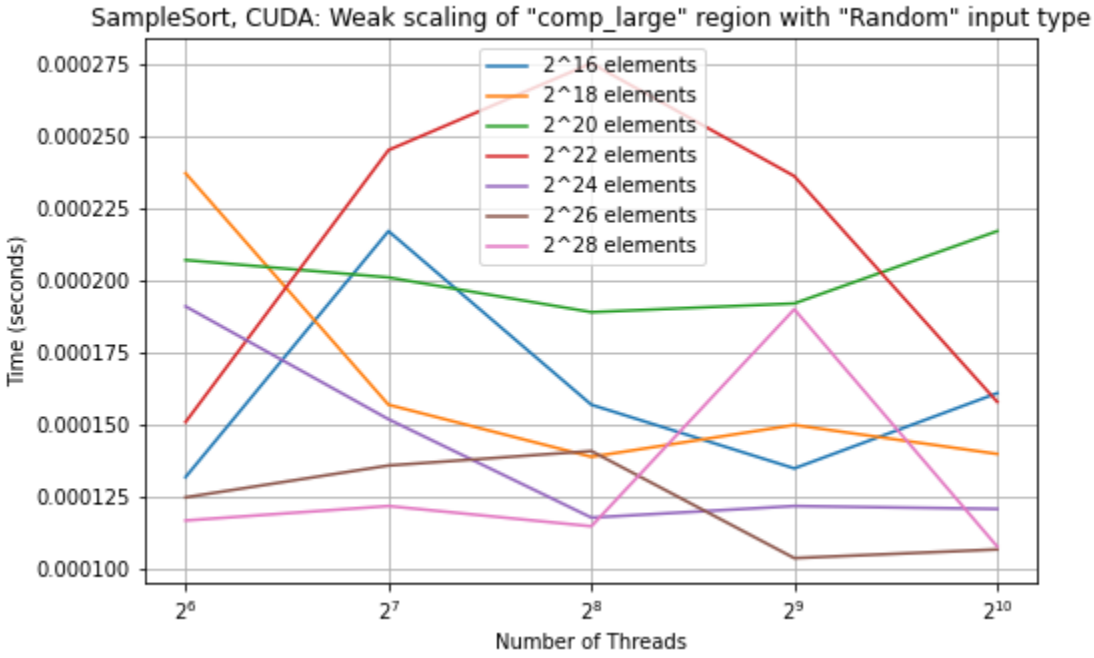

SampleSort, CUDA: Weak scaling of "comm" region with "1perturbed" input type
Time (seconds)
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

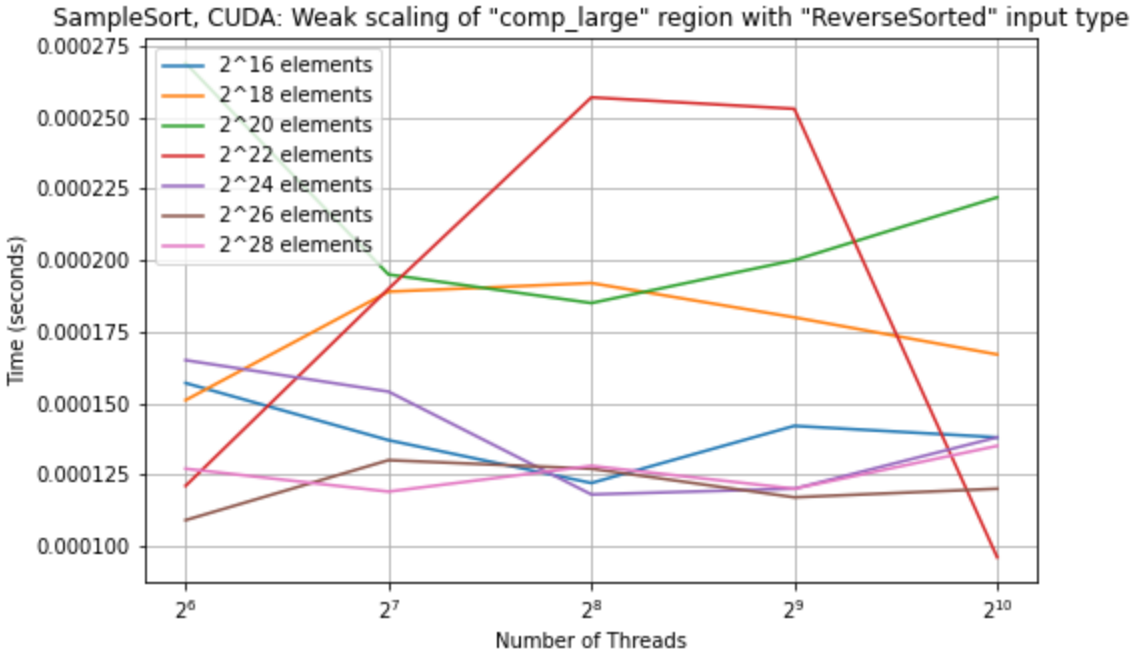

SampleSort, CUDA: Weak scaling of "comm" region with "Random" input type
Time (seconds)
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

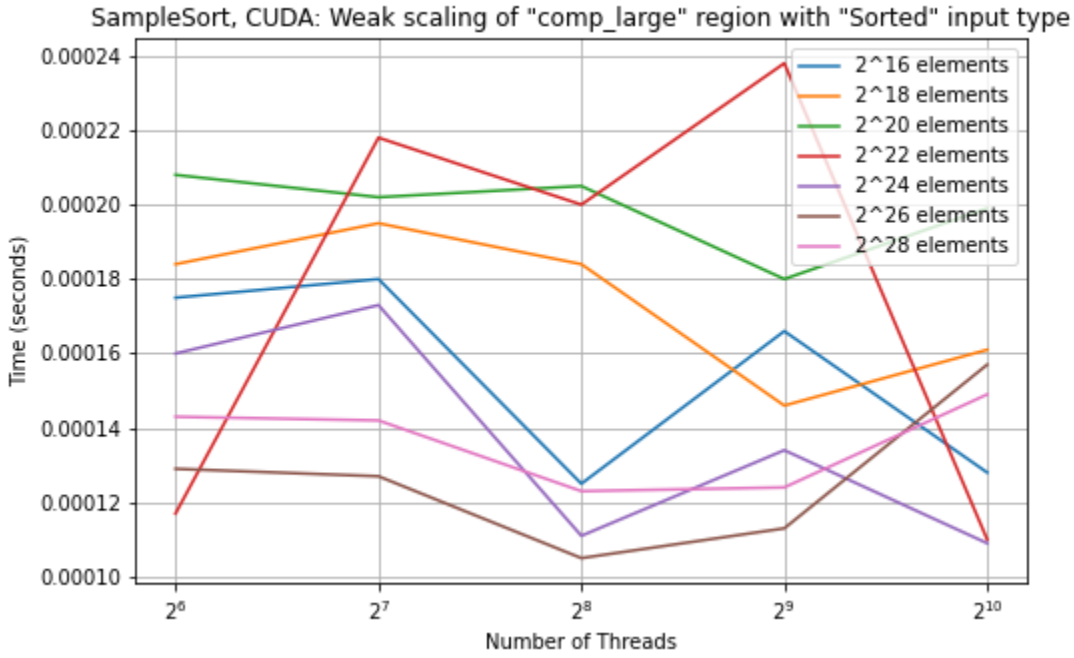

SampleSort, CUDA: Weak scaling of "comm" region with "ReverseSorted" input type
Time (seconds)
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

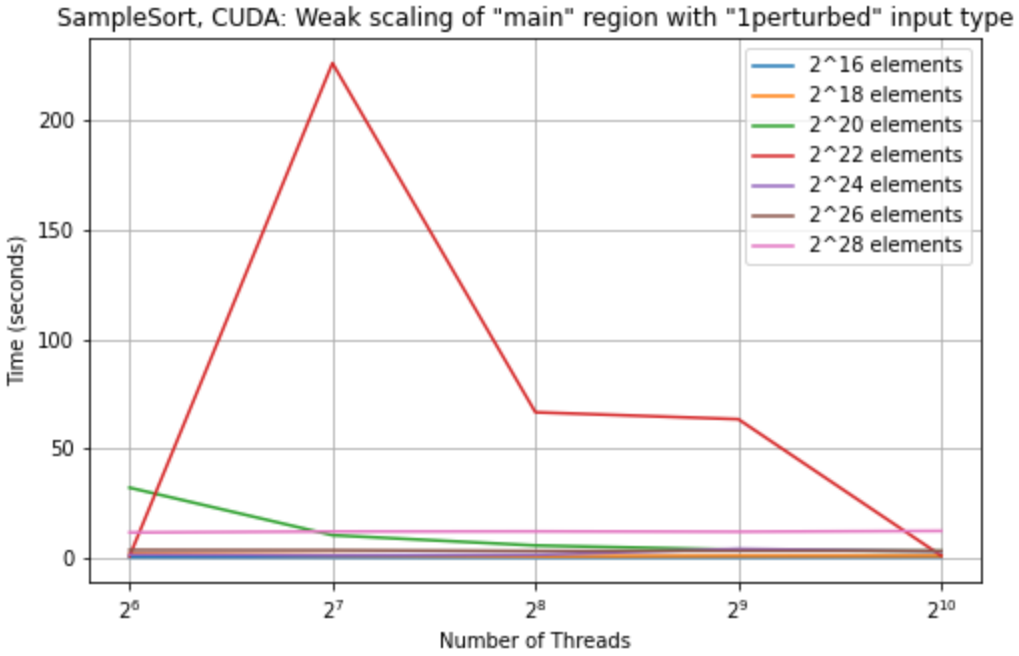

SampleSort, CUDA: Weak scaling of "comm" region with "Sorted" input type
Time (seconds)
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

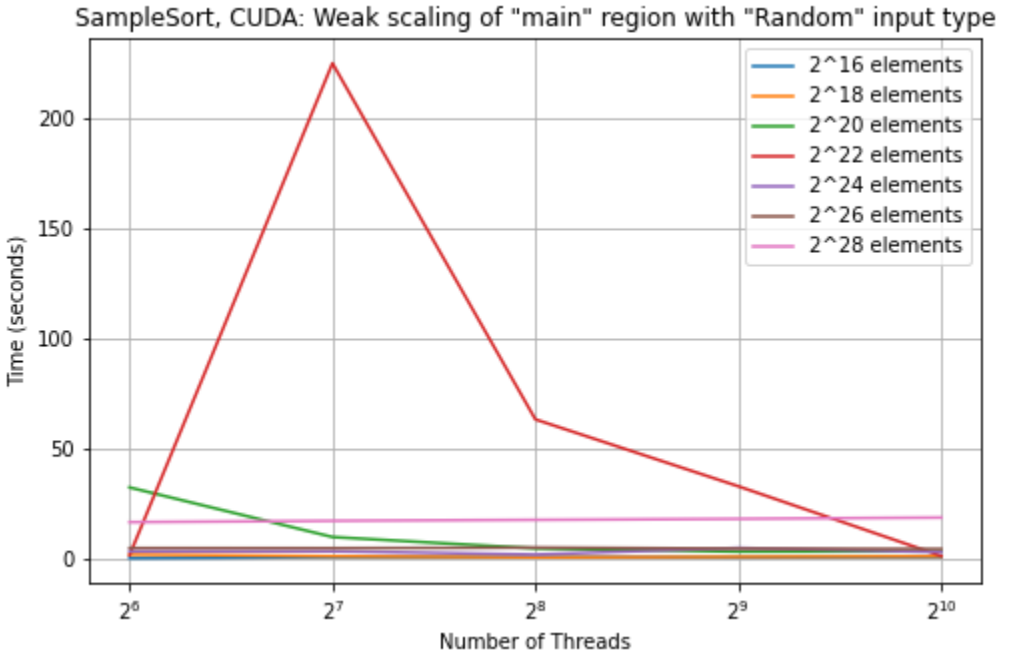

SampleSort, CUDA: Weak scaling of "comp_large" region with "1perturbed" input type
Time (seconds)
0.00024
0.00022
0.00020
0.00018
0.00016
0.00014
0.00012
0.00010
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

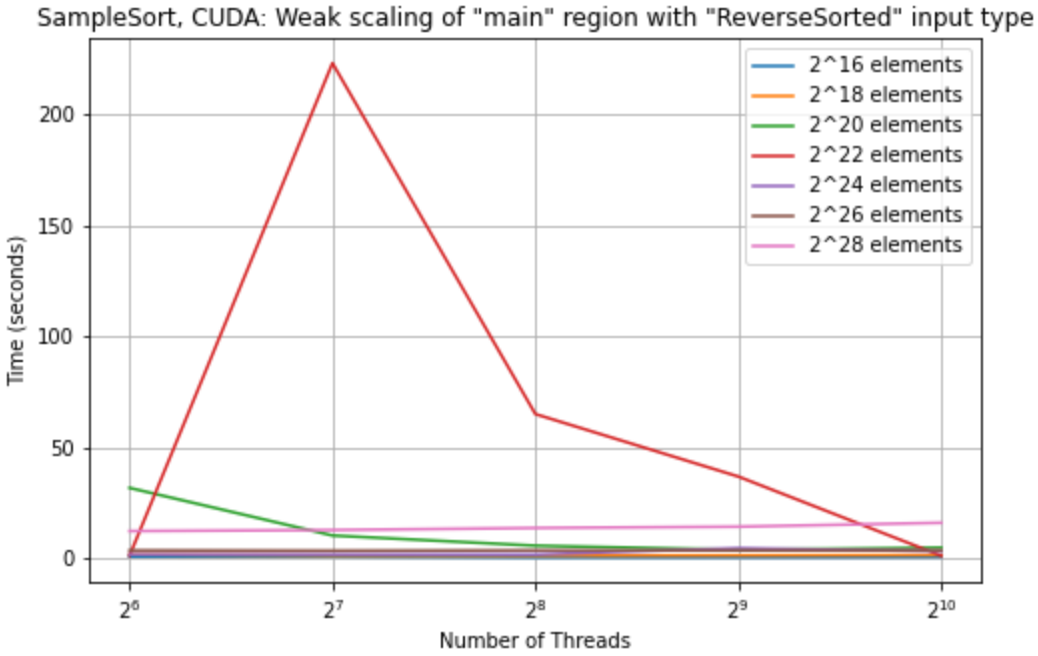

SampleSort, CUDA: Weak scaling of "comp_large" region with "Random" input type
Time (seconds)
0.000275
0.000250
0.000225
0.000200
0.000175
0.000150
0.000125
0.000100
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

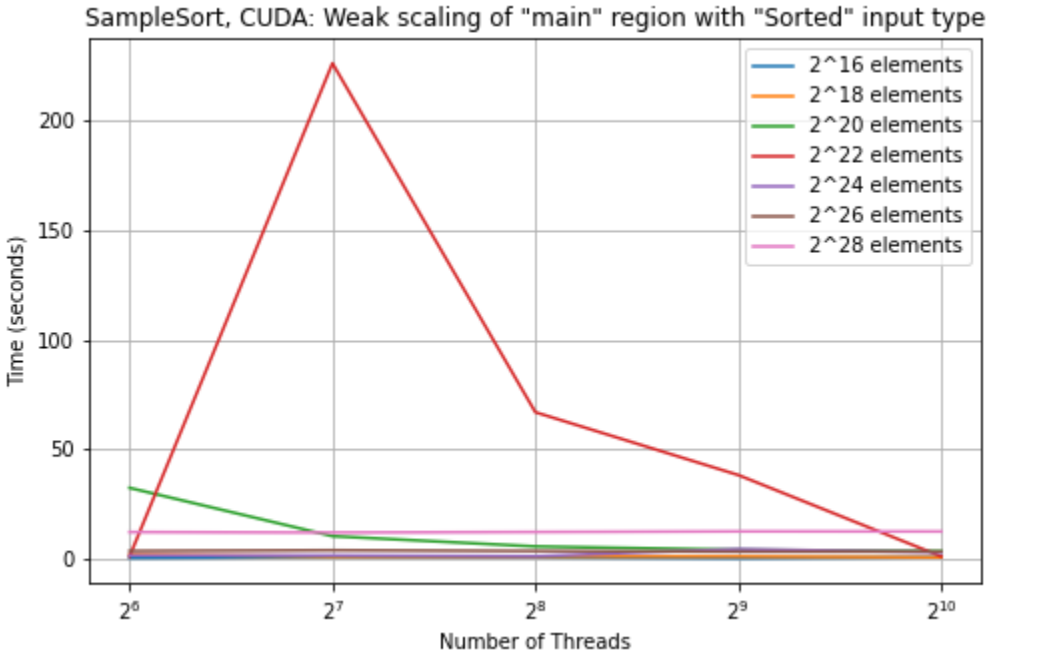

SampleSort, CUDA: Weak scaling of "comp_large" region with "ReverseSorted" input type
Time (seconds)
0.000275
0.000250
0.000225
0.000200
0.000175
0.000150
0.000125
0.000100
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

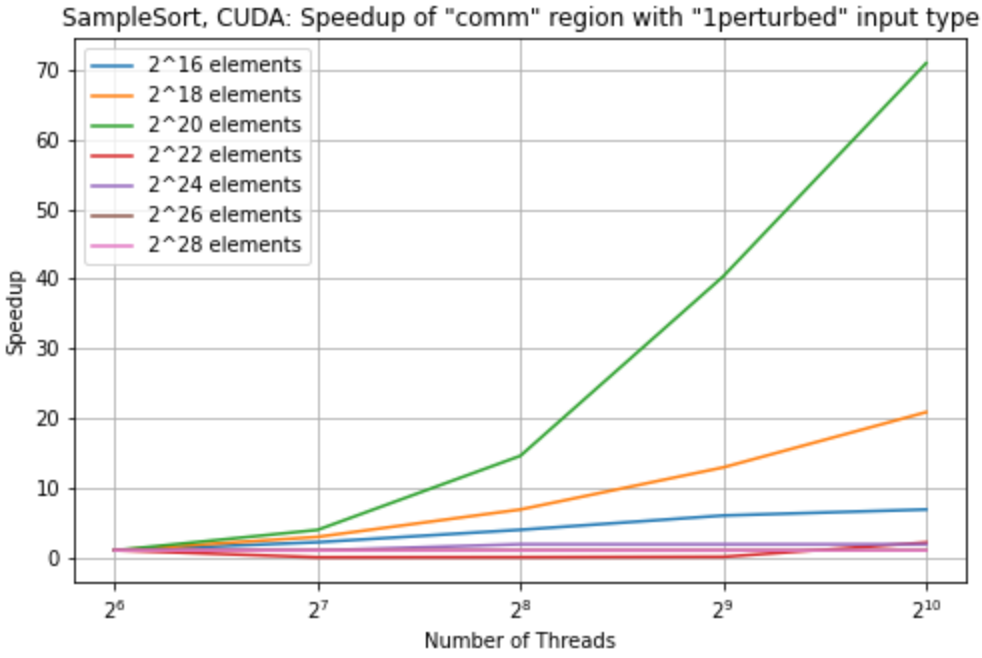

SampleSort, CUDA: Weak scaling of "comp_large" region with "Sorted" input type
Time (seconds)
0.00024
0.00022
0.00020
0.00018
0.00016
0.00014
0.00012
0.00010
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

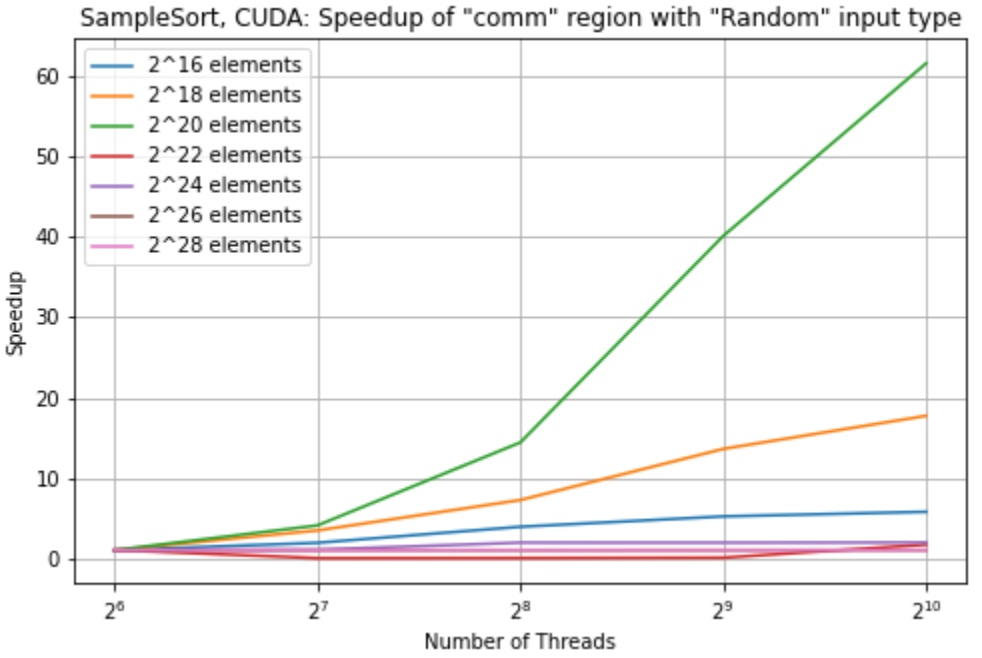

SampleSort, CUDA: Weak scaling of "main" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0
50
100
150
200
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

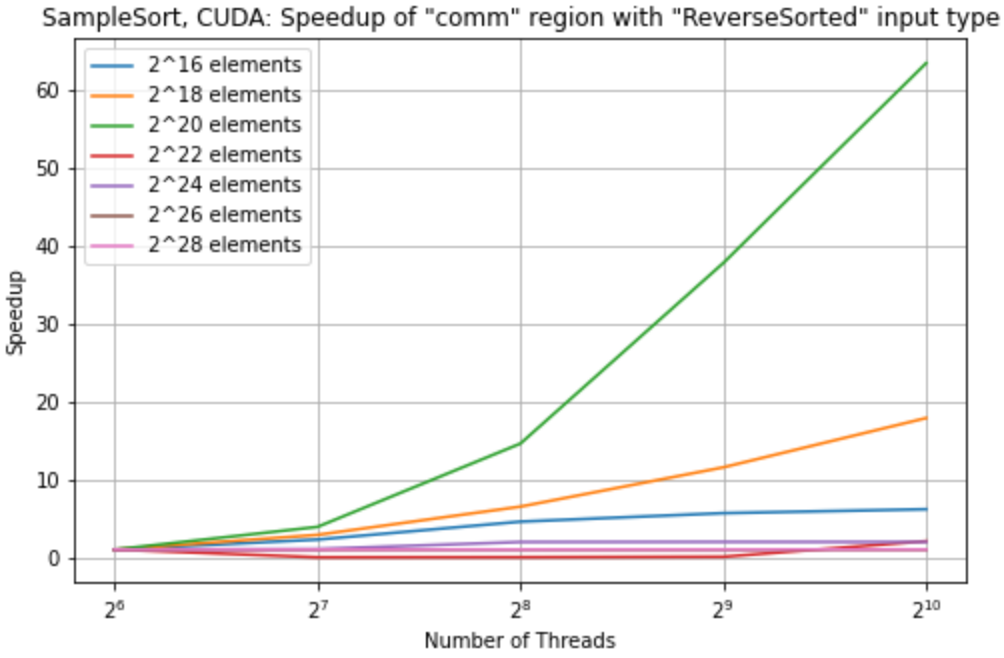

SampleSort, CUDA: Weak scaling of "main" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0
50
100
150
200
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

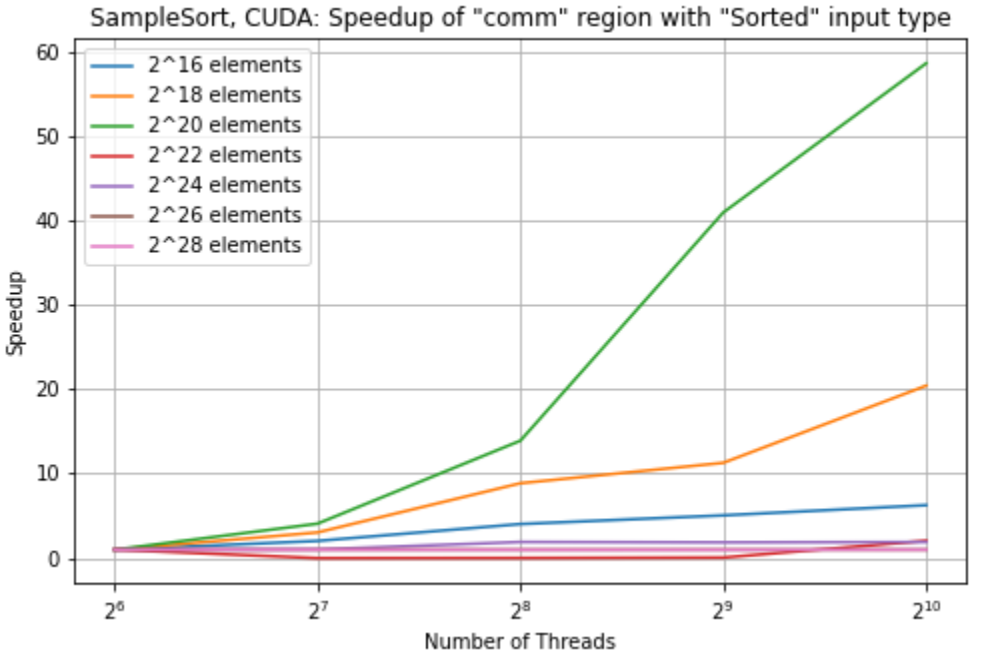

SampleSort, CUDA: Weak scaling of "main" region with "ReverseSorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0
50
100
150
200
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

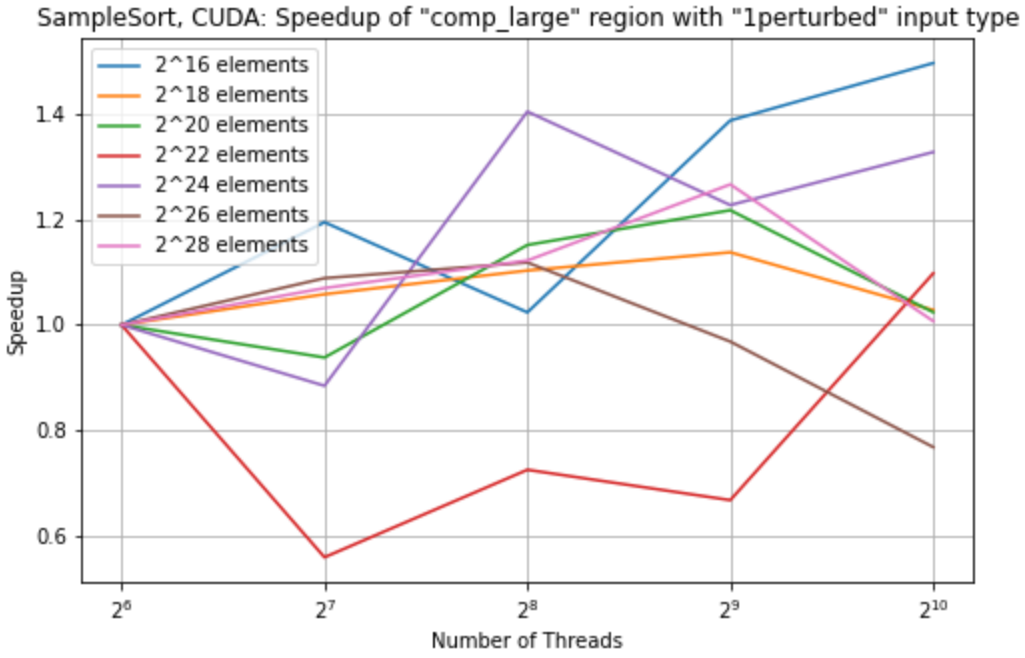

SampleSort, CUDA: Weak scaling of "main" region with "Sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0
50
100
150
200
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

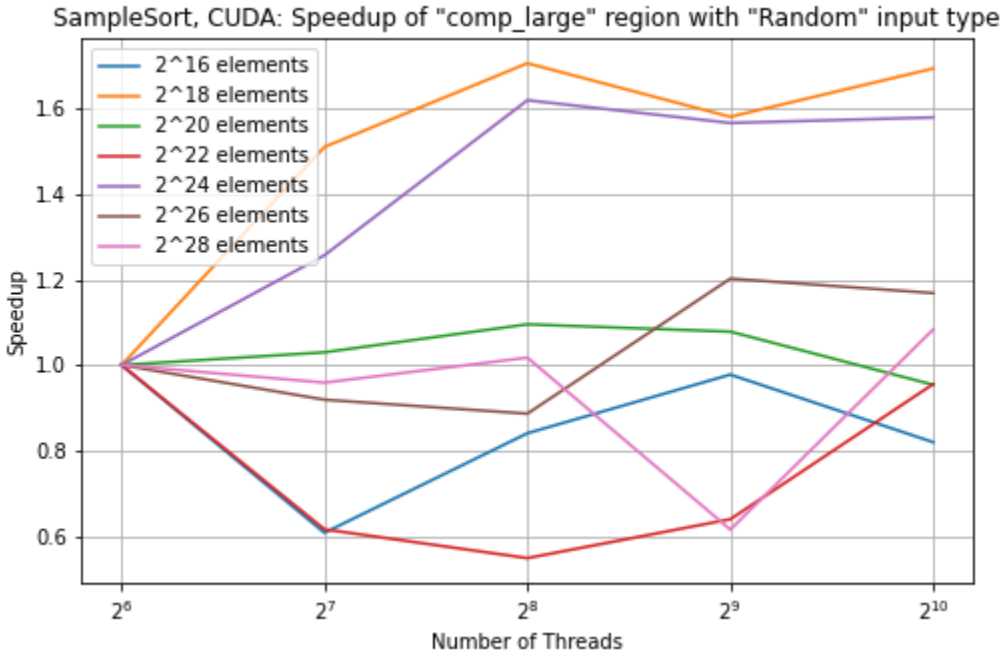

SampleSort, CUDA: Speedup of "comm" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0
10
20
30
40
50
60
70
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

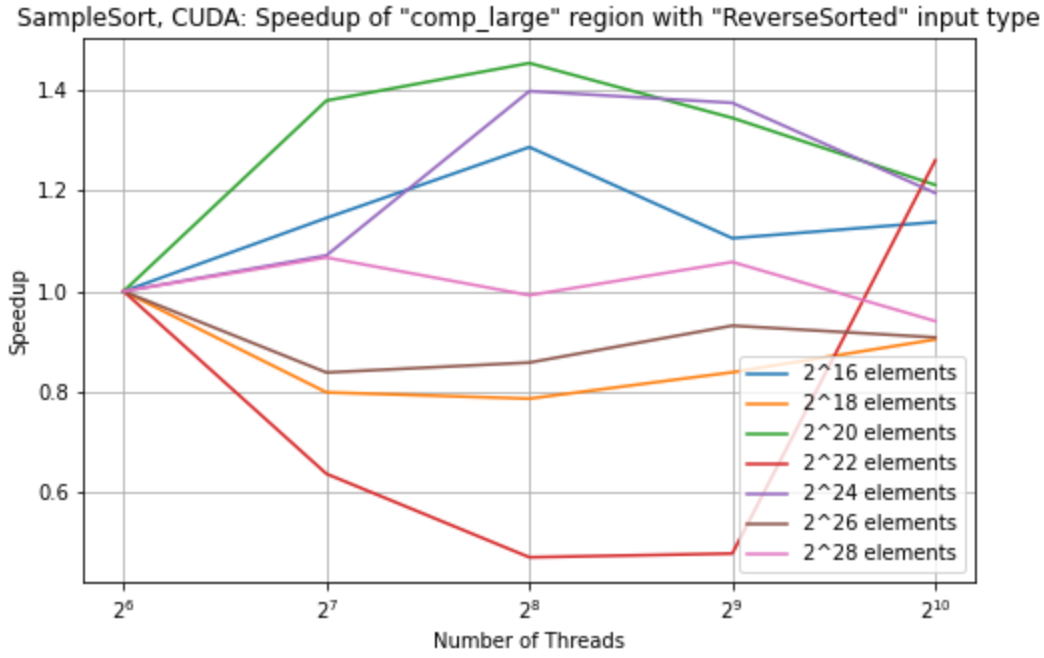

SampleSort, CUDA: Speedup of "comm" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0
10
20
30
40
50
60
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

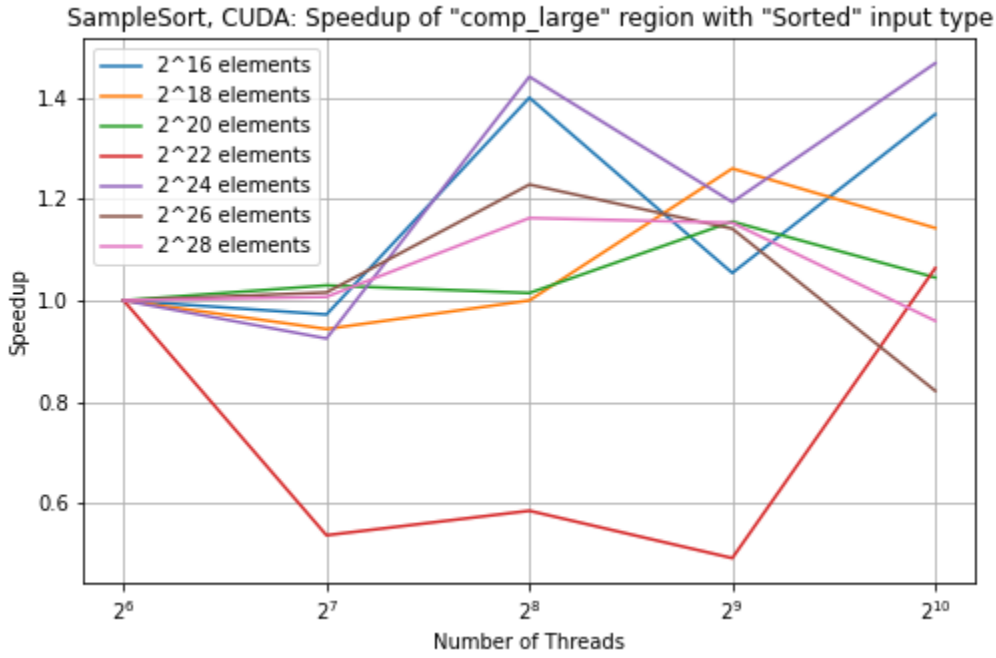

SampleSort, CUDA: Speedup of "comm" region with "ReverseSorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0
10
20
30
40
50
60
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

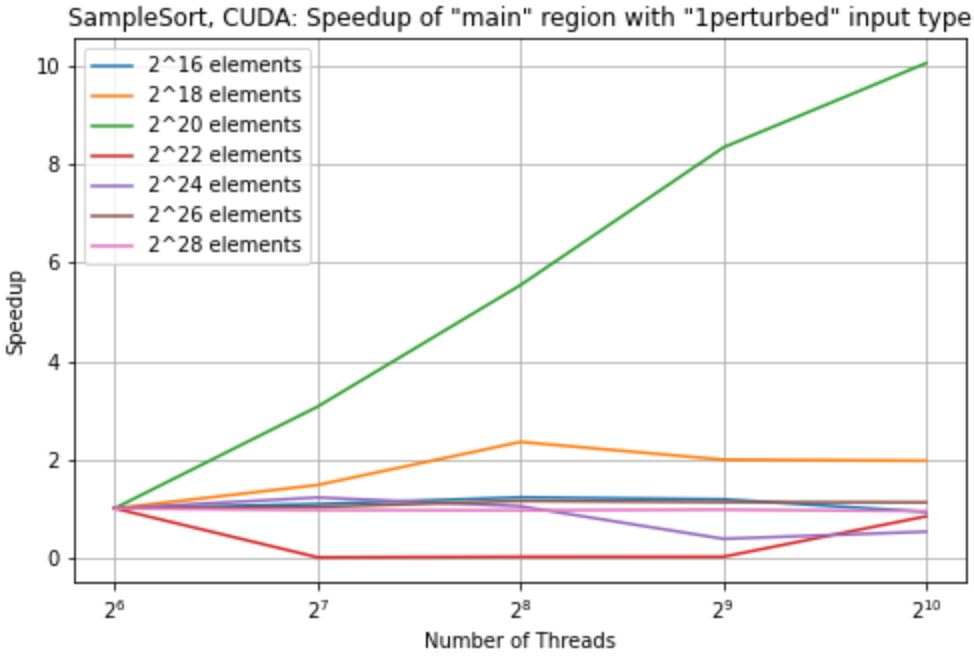

SampleSort, CUDA: Speedup of "comm" region with "Sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0
10
20
30
40
50
60
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

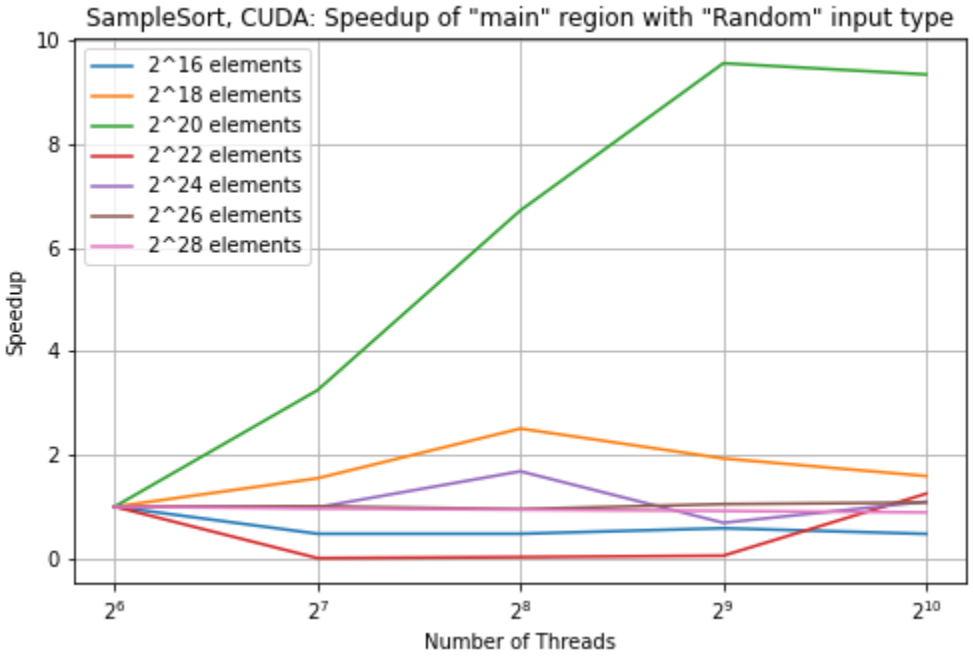

SampleSort, CUDA: Speedup of "comp_large" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
1.4
1.2
1.0
0.8
0.6
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

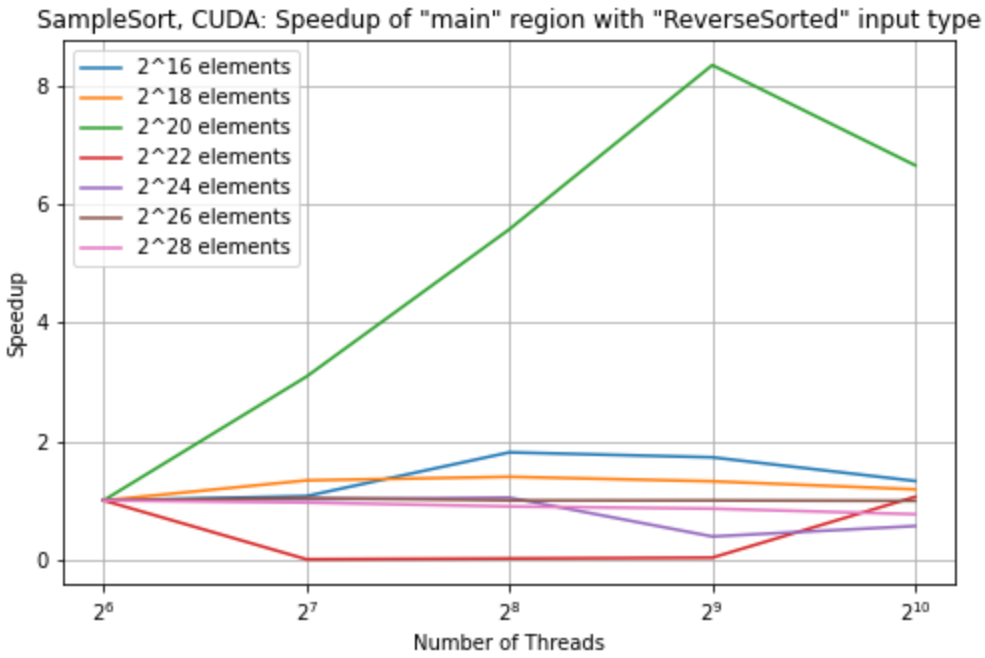

SampleSort, CUDA: Speedup of "comp_large" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
1.6
1.4
1.2
1.0
0.8
0.6
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

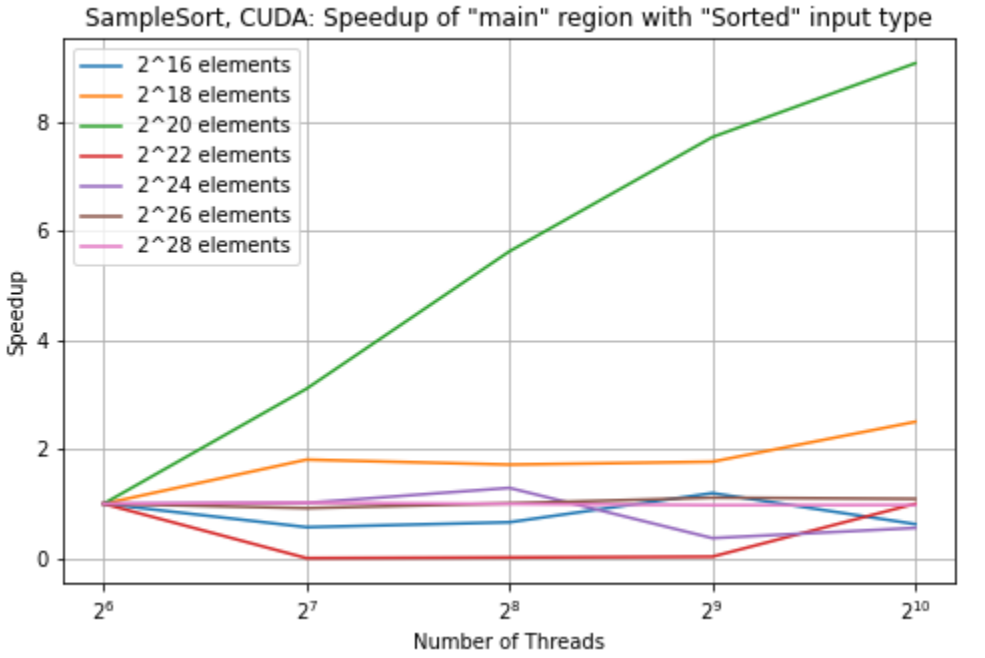

SampleSort, CUDA: Speedup of "comp_large" region with "ReverseSorted" input type
Speedup
1.4
1.2
1.0
0.8
0.6
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads


SampleSort, CUDA: Speedup of "comp_large" region with "Sorted" input type
Speedup
1.4
1.2
1.0
0.8
0.6
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads


SampleSort, CUDA: Speedup of "main" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Threads


SampleSort, CUDA: Speedup of "main" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Threads




**Analysis**

The strong scaling comm graphs show an overall trend of computation time going down as the number of threads increases. This is reflected in the speedup graph of comm as well with a speedup of nearly 60x for 2^20 elements.

On the other hand, computation seems to be completely independent of the number of threads and stays relatively constant. This is shown by all the comp graphs – strong, weak, and speedup – showing no overall trend, and speedup graphs showing only a slight variance. This still shows good scalability in that although computation times don't decrease, they don't increase as well when the number of threads increases.

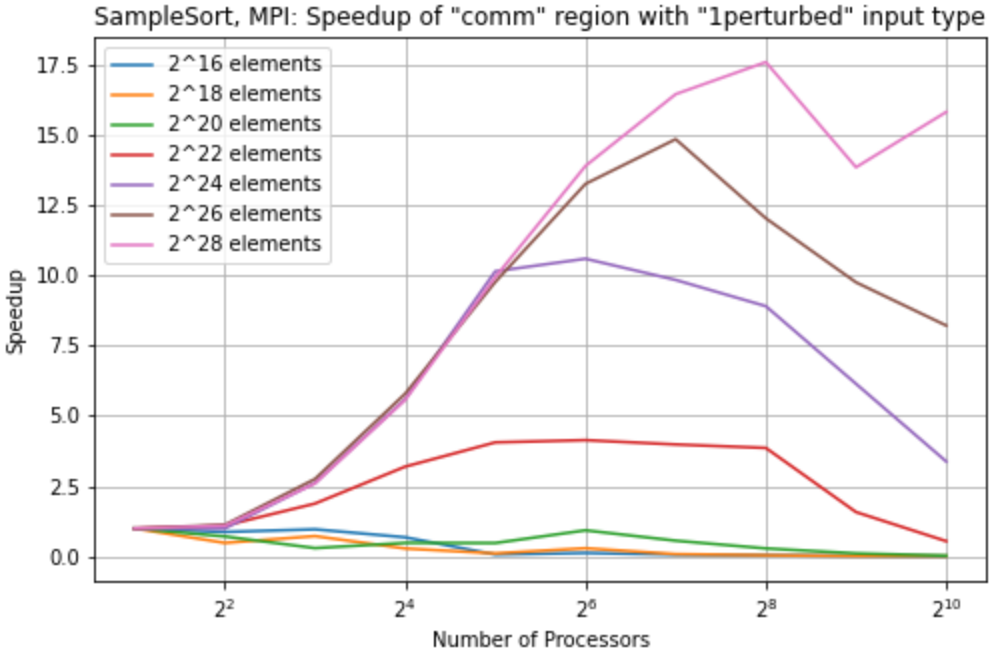
There doesn't appear to be any difference in times between different input types. This is because the sample sort algorithm I implemented will still have to go through all the stages of sample sort from communication to computation no matter how sorted the original input data already is.

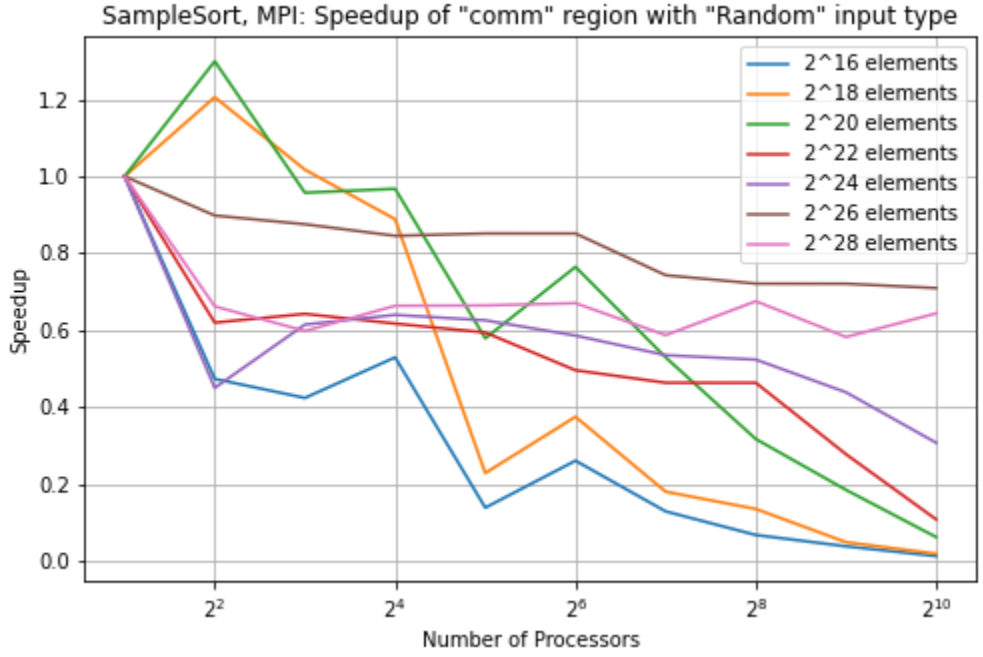
Overall, there is a significant speedup in the algorithm as the number of threads increases as shown by the main speedup graphs.

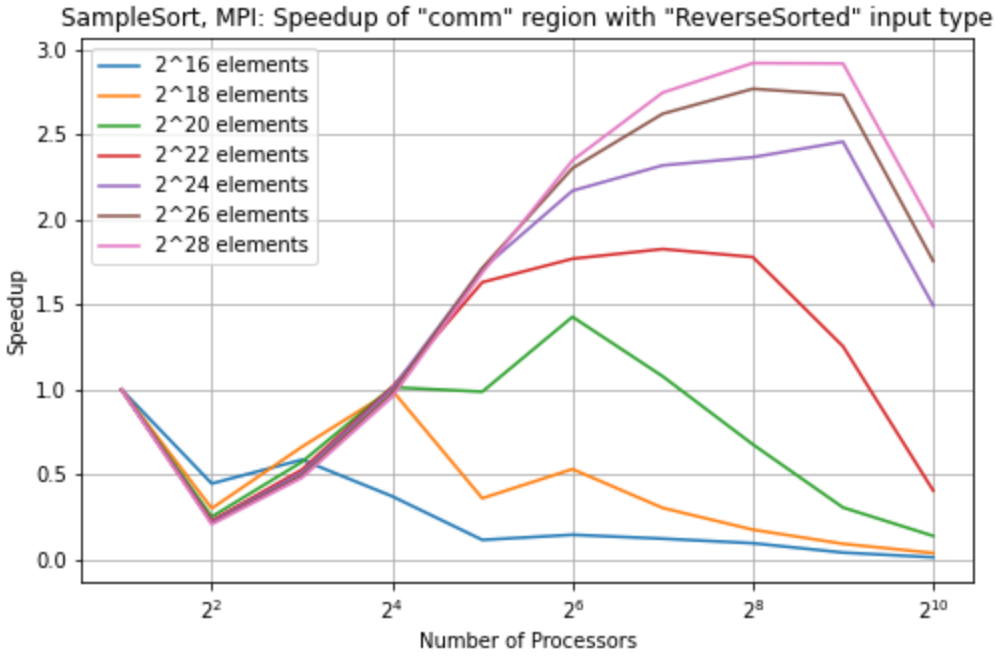
# Sample Sort MPI

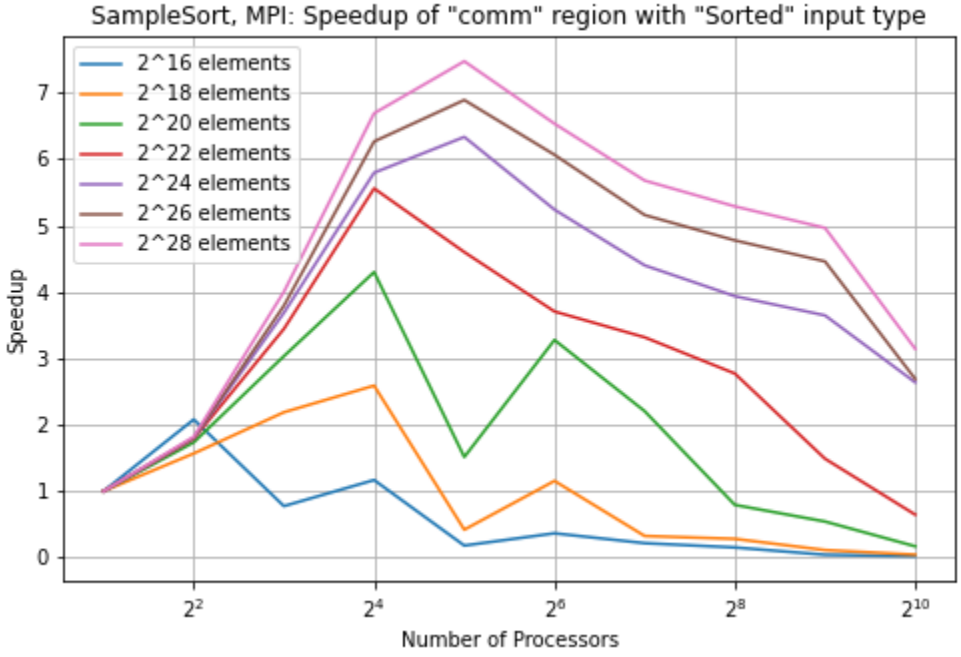
**Graphs:**

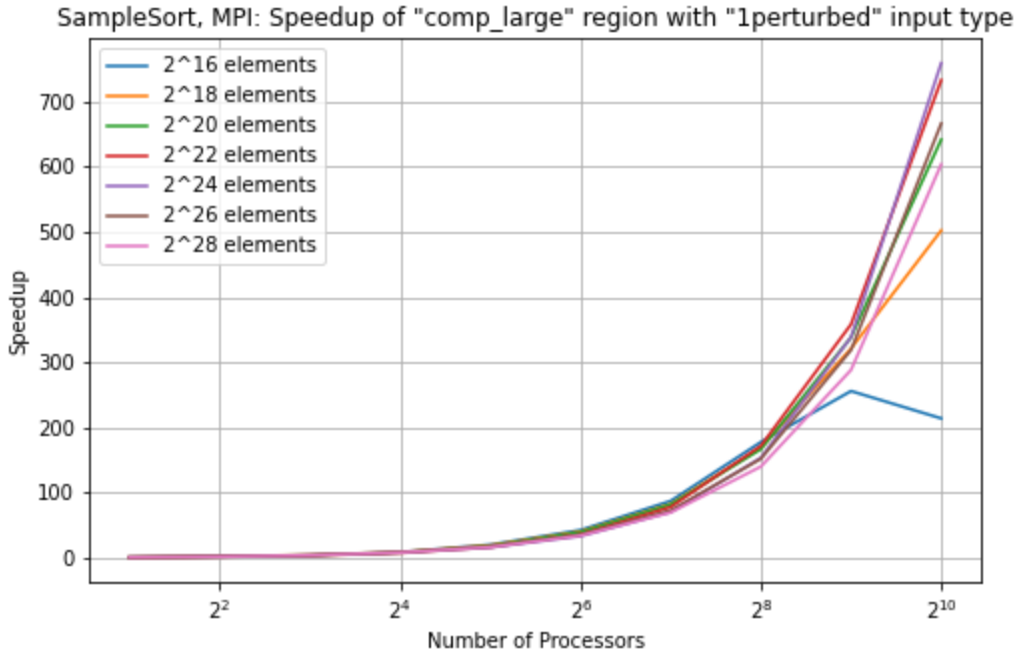
- **Speed Up:**

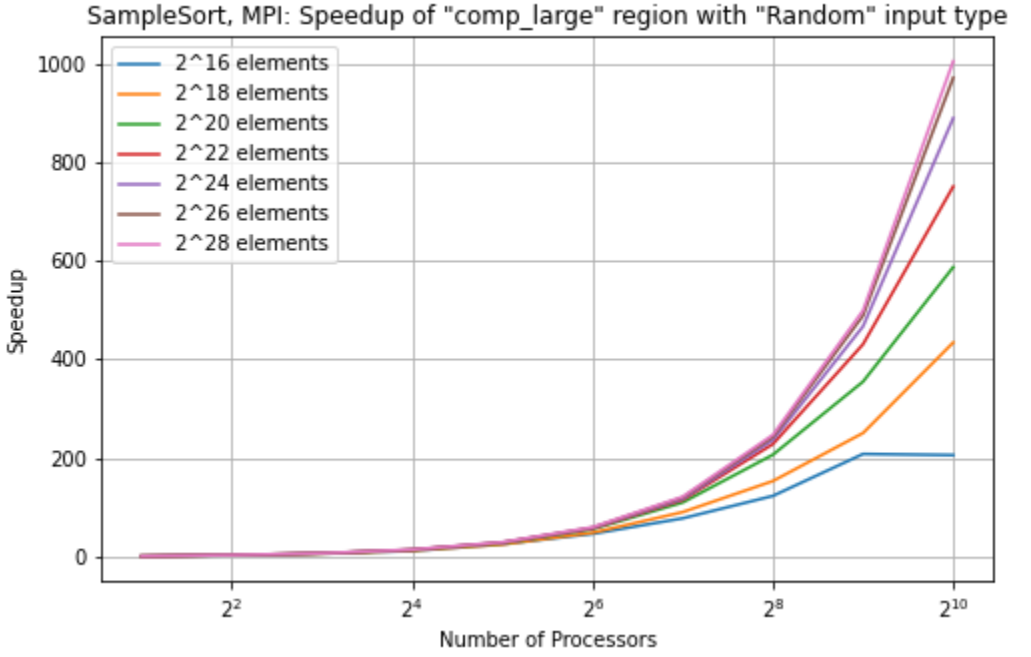

SampleSort, MPI: Speedup of "comm" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors

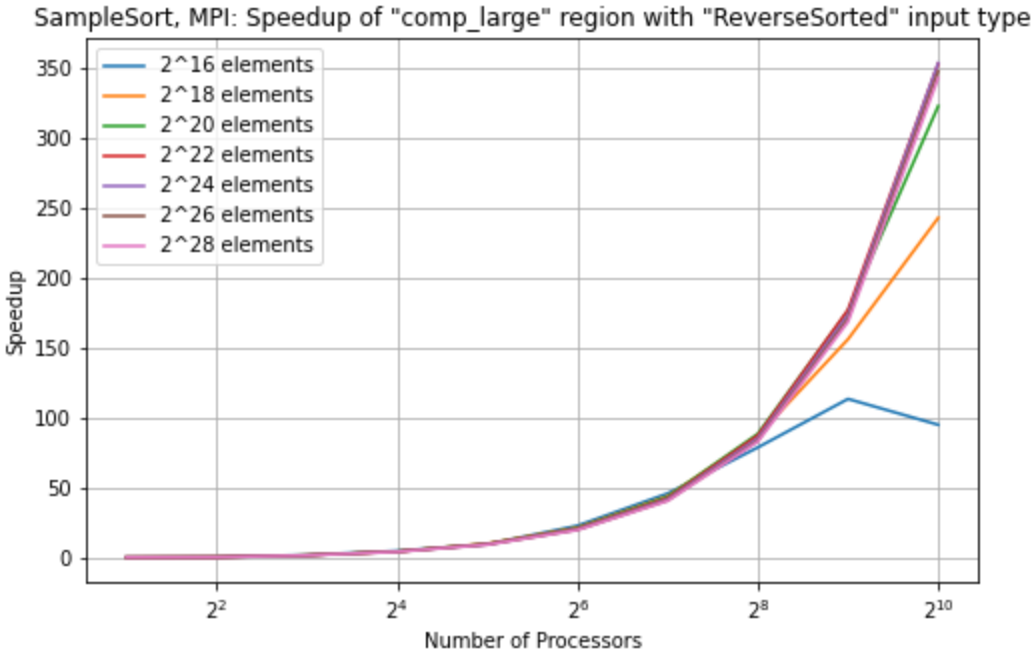

SampleSort, MPI: Speedup of "comm" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors

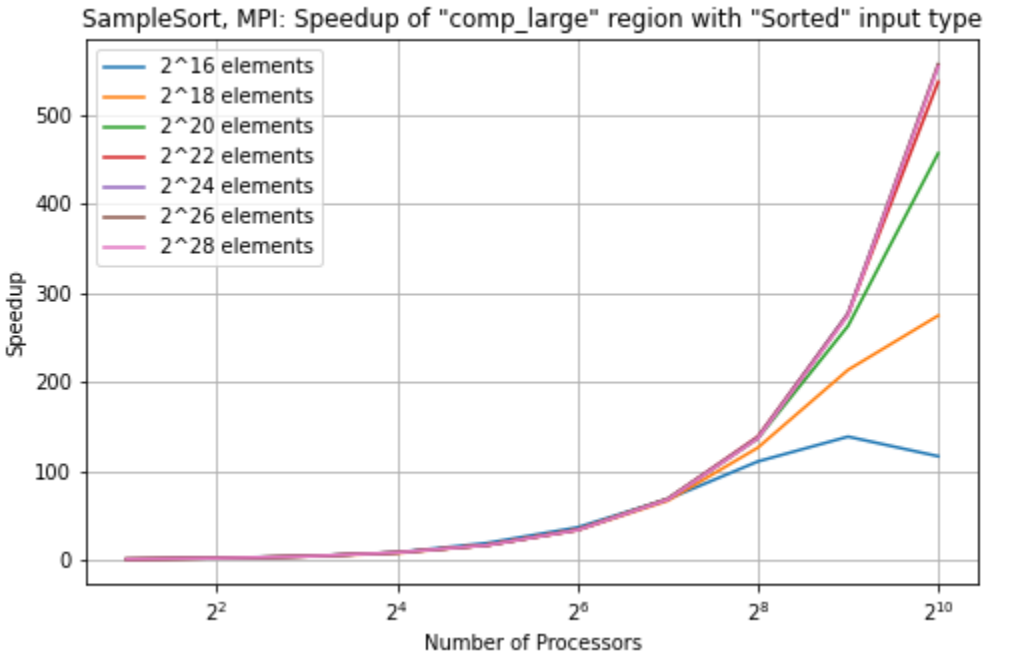

SampleSort, MPI: Speedup of "comm" region with "ReverseSorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors

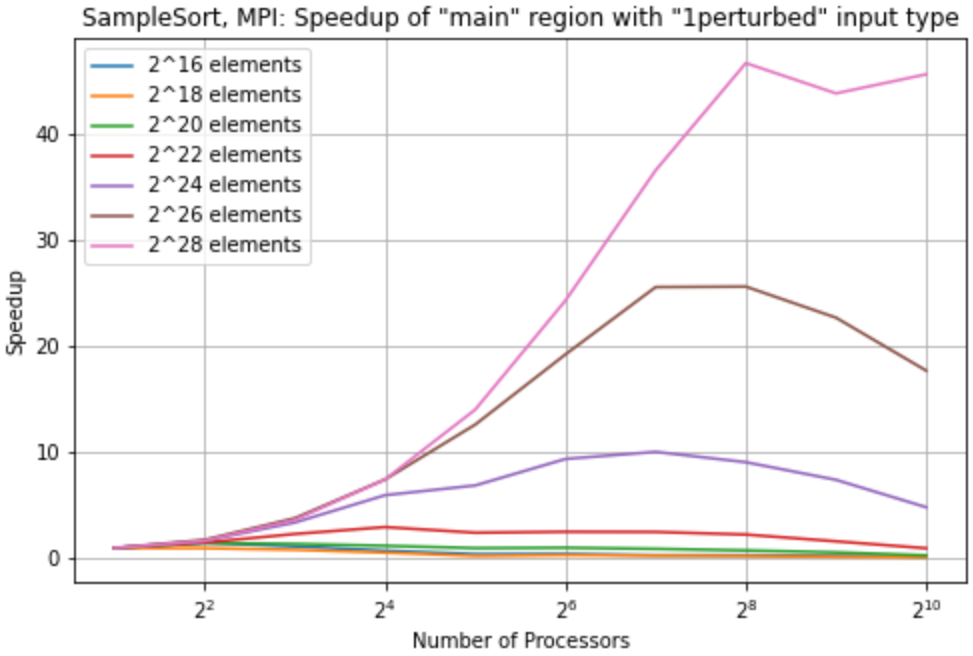

SampleSort, MPI: Speedup of "comm" region with "Sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors

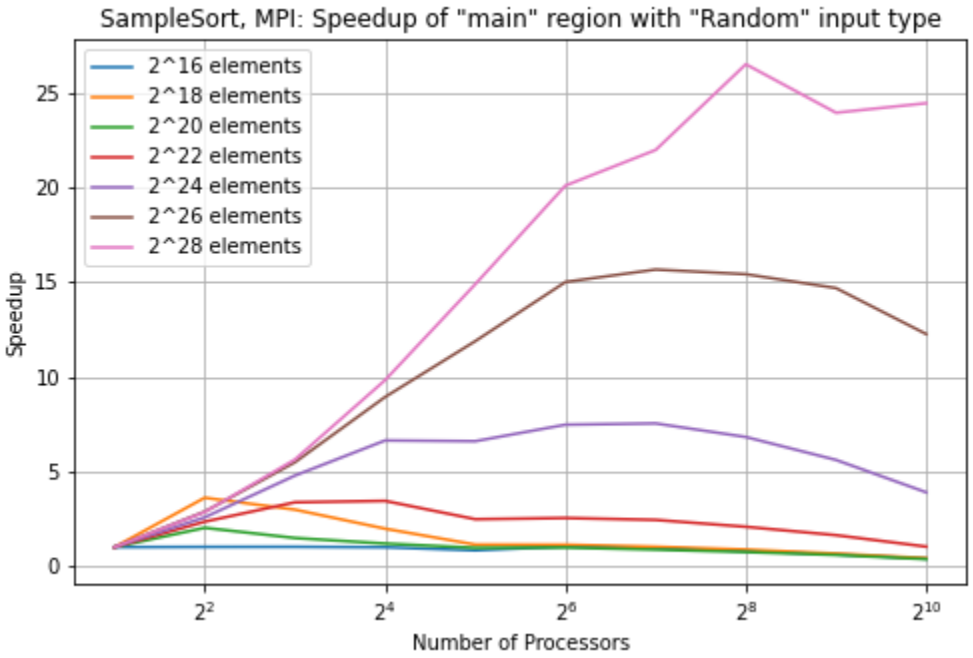

SampleSort, MPI: Speedup of "comp_large" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0
100
200
300
400
500
600
700
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

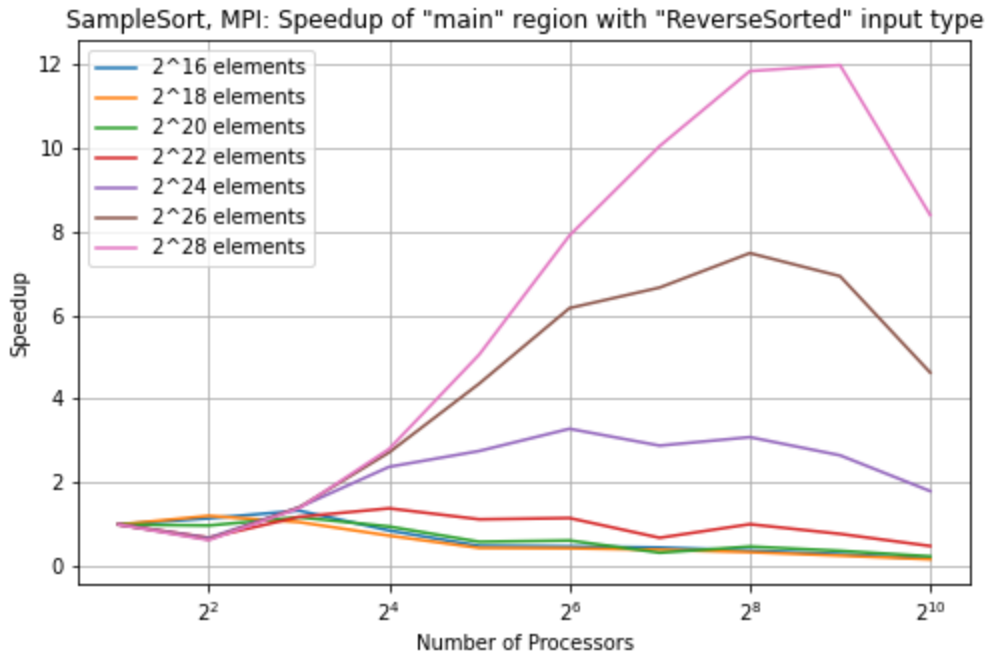

SampleSort, MPI: Speedup of "comp_large" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0
200
400
600
800
1000
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

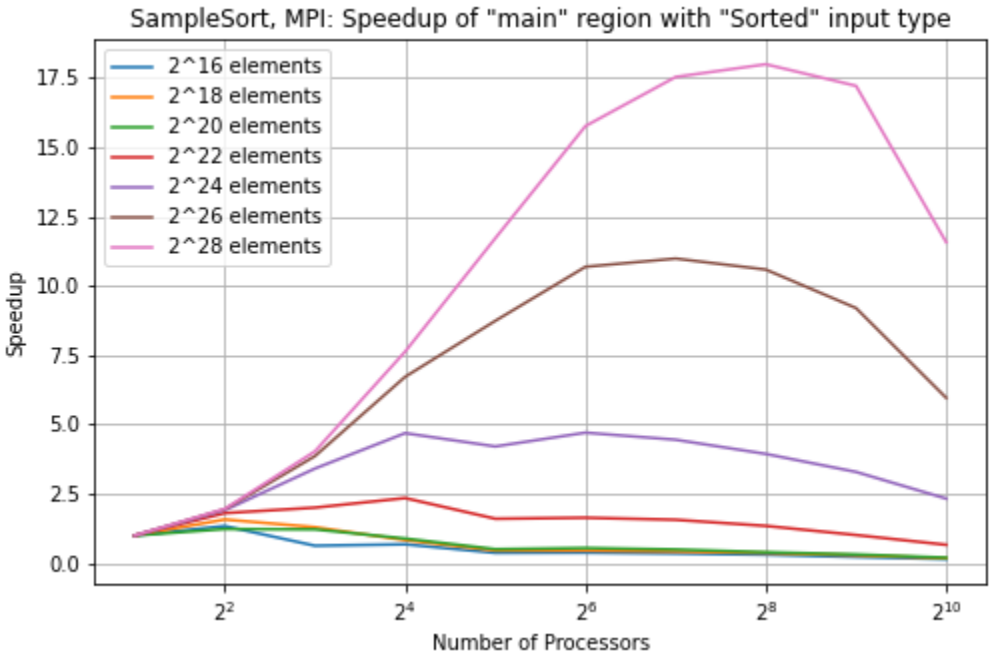

SampleSort, MPI: Speedup of "comp_large" region with "ReverseSorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0
50
100
150
200
250
300
350
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors


SampleSort, MPI: Speedup of "comp_large" region with "Sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0
100
200
300
400
500
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

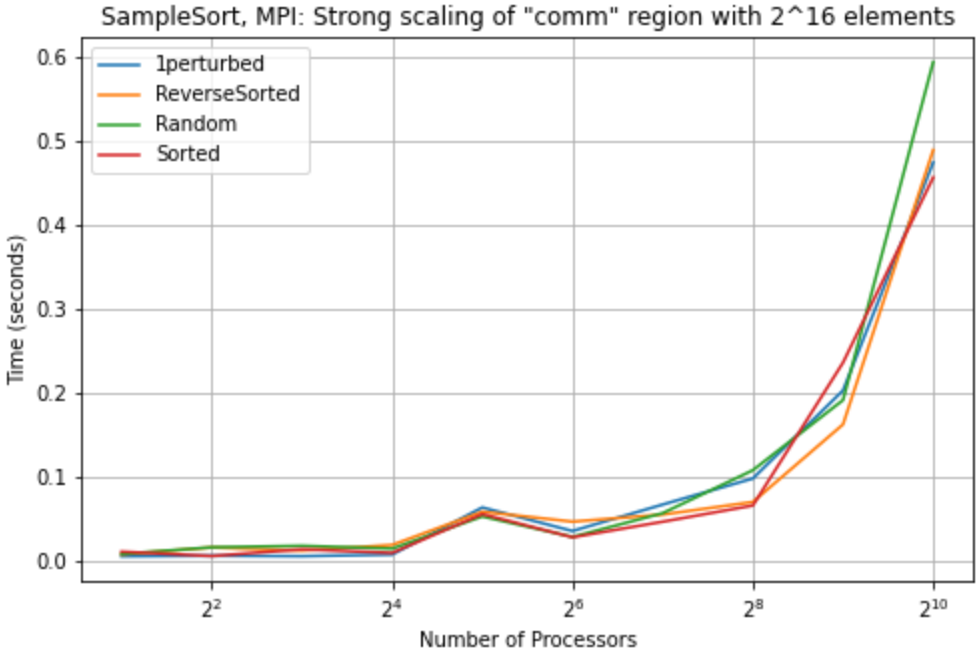

SampleSort, MPI: Speedup of "main" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0
10
20
30
40
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

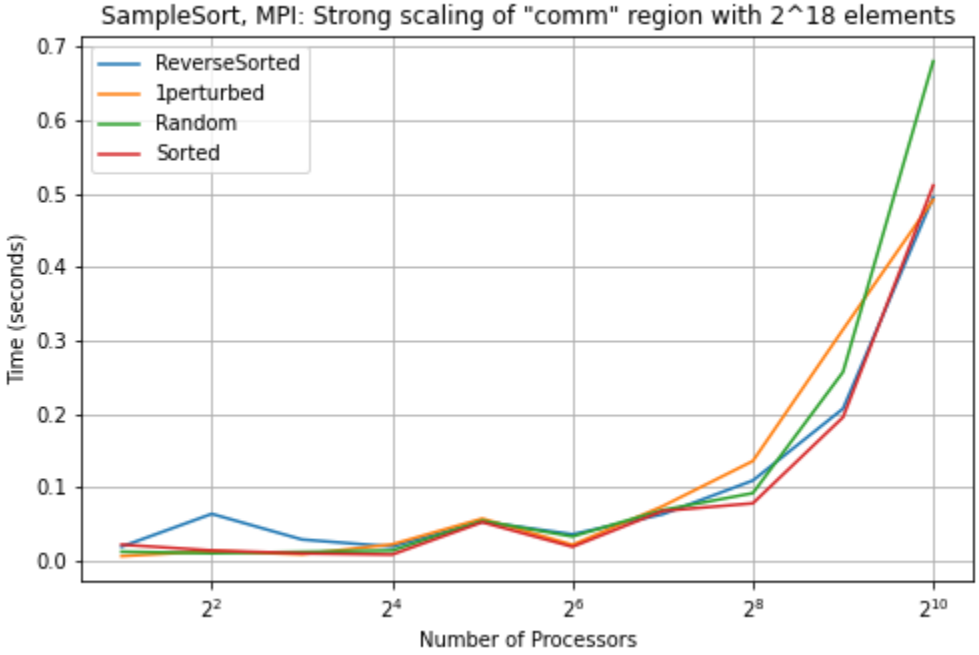

SampleSort, MPI: Speedup of "main" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0
5
10
15
20
25
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

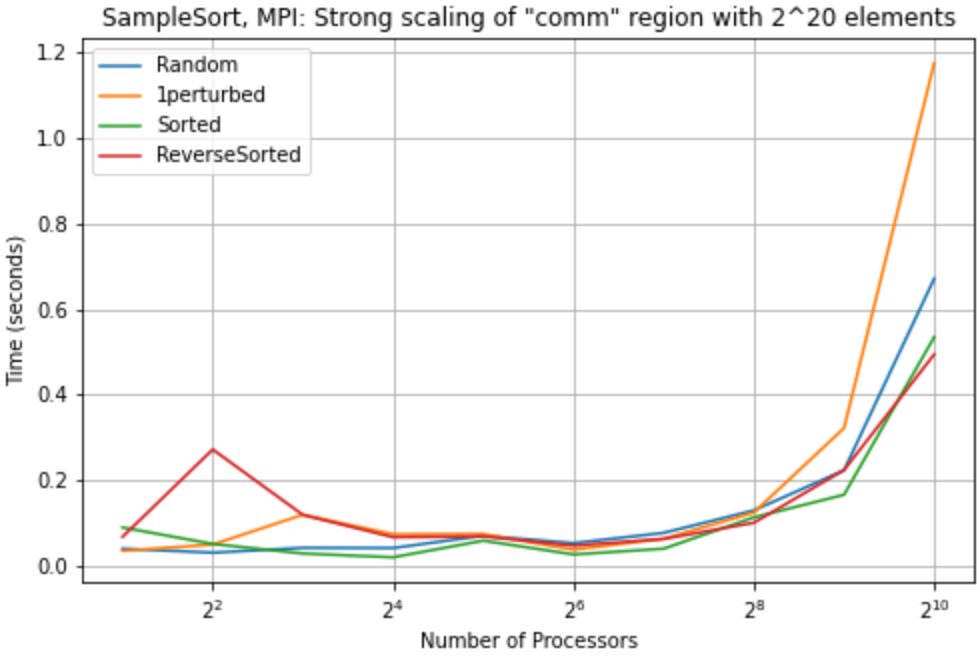


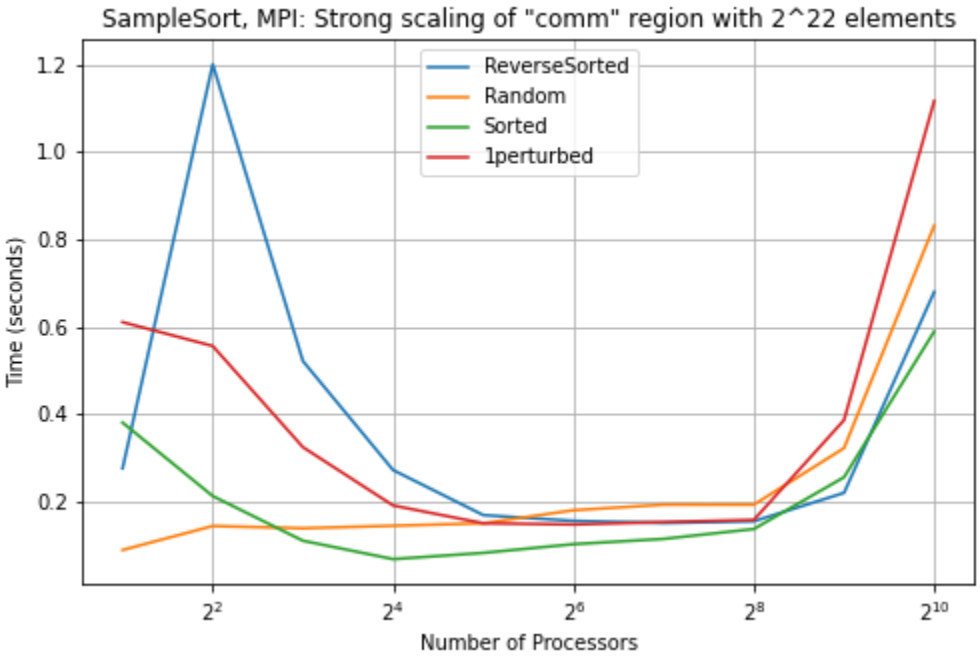



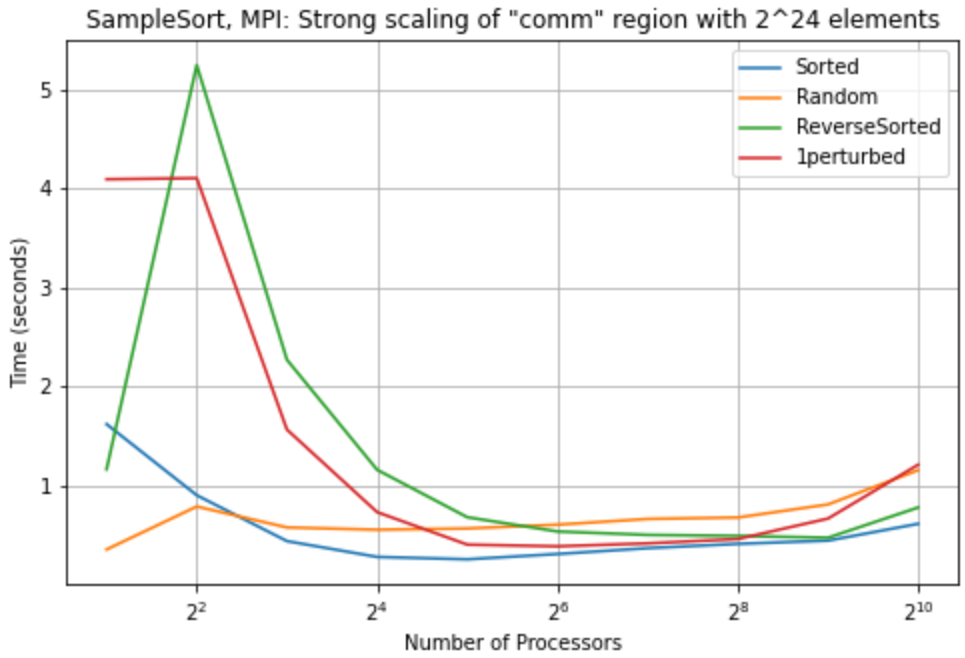

- **<u>Strong Scaling:</u>**

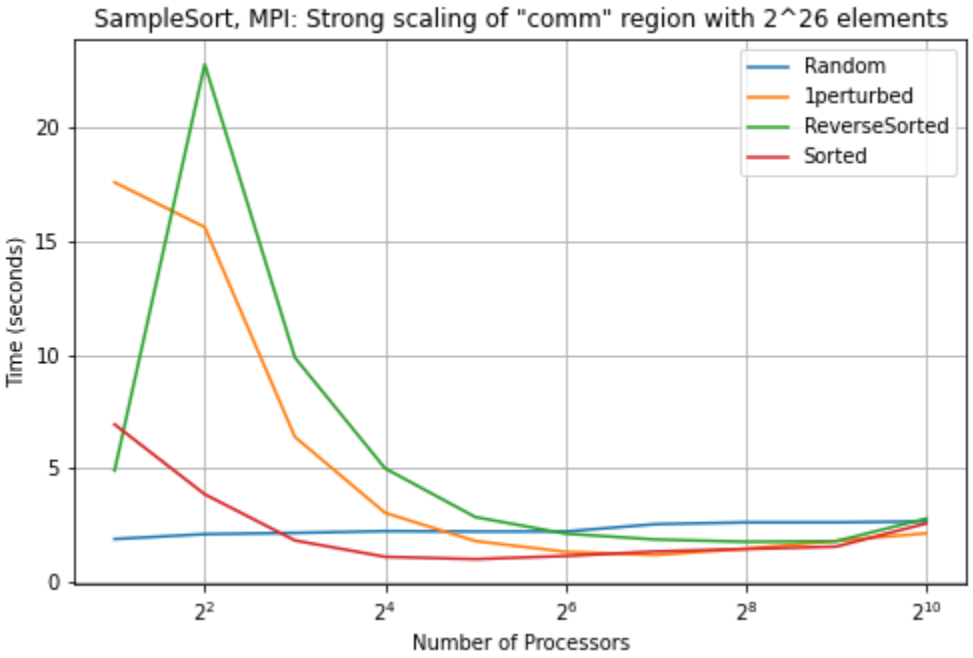

SampleSort, MPI: Strong scaling of "comm" region with 2^16 elements
1perturbed
ReverseSorted
Random
Sorted
Time (seconds)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

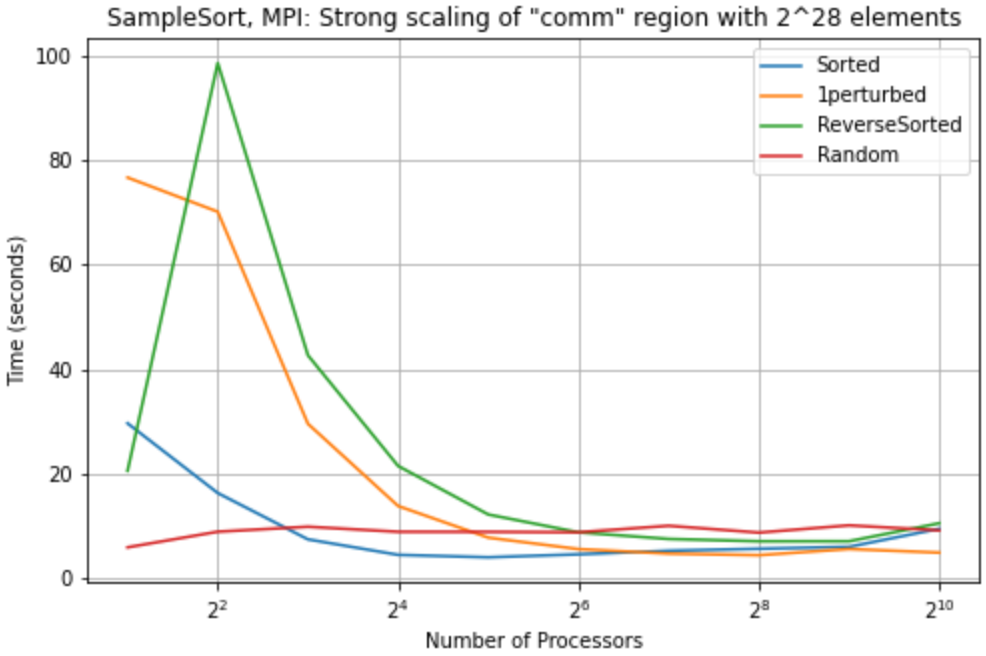

SampleSort, MPI: Strong scaling of "comm" region with 2^18 elements
ReverseSorted
1perturbed
Random
Sorted
Time (seconds)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

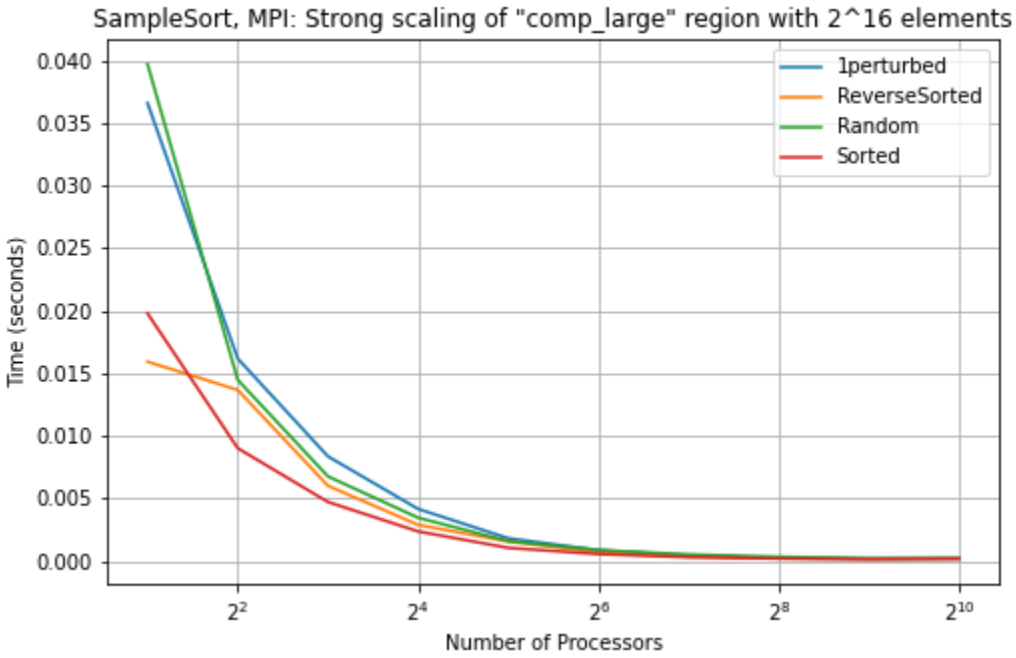

SampleSort, MPI: Strong scaling of "comm" region with 2^20 elements
Random
1perturbed
Sorted
ReverseSorted
Time (seconds)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

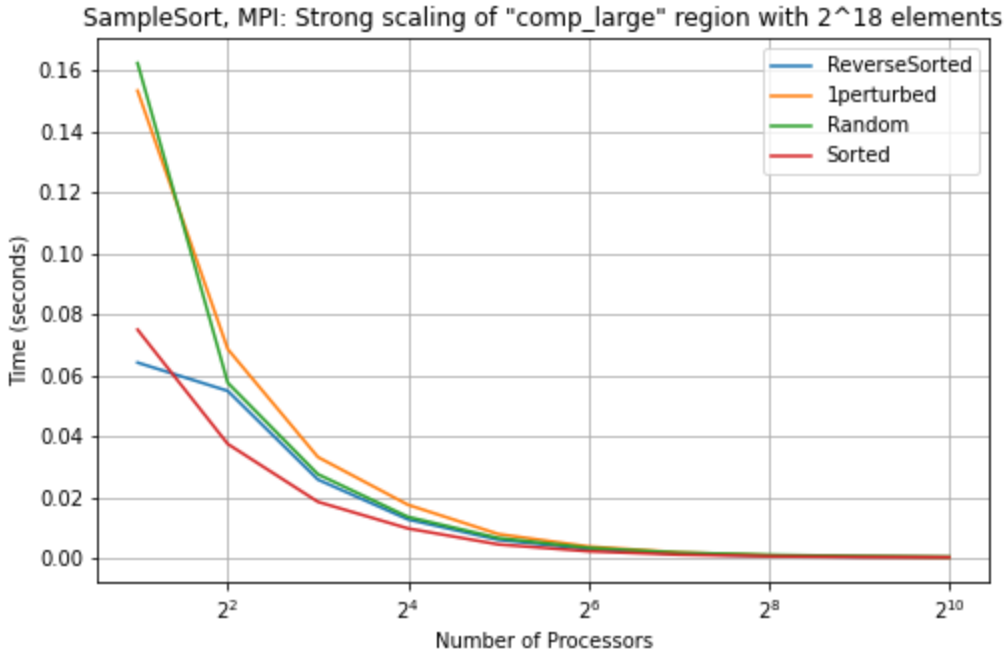

SampleSort, MPI: Strong scaling of "comm" region with 2^22 elements
ReverseSorted
Random
Sorted
1perturbed
Time (seconds)
0.2
0.4
0.6
0.8
1.0
1.2
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

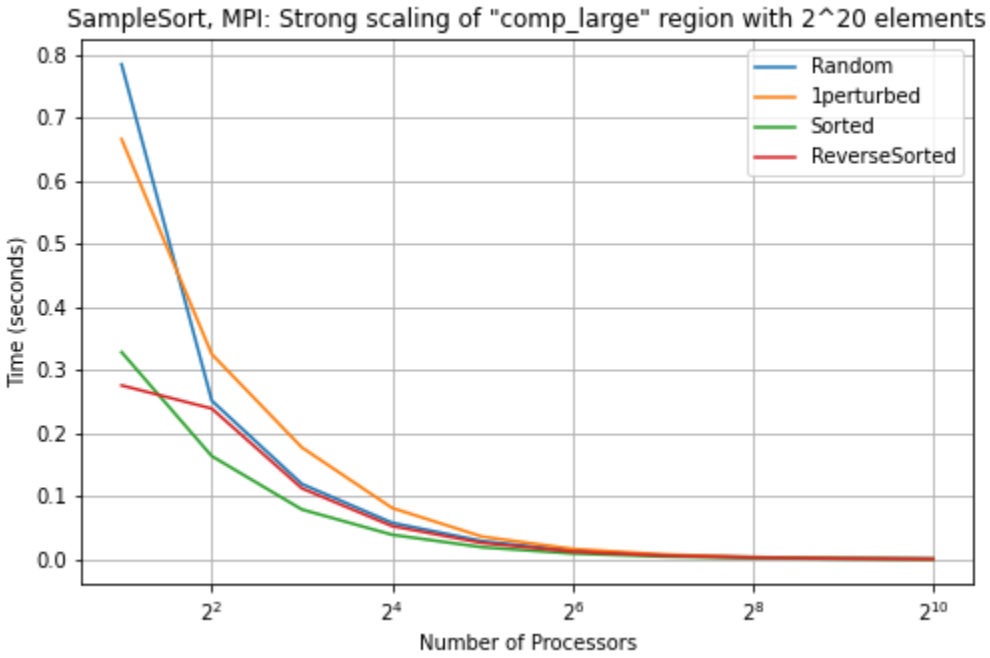

SampleSort, MPI: Strong scaling of "comm" region with 2^24 elements
Sorted
Random
ReverseSorted
1perturbed
Time (seconds)
Number of Processors

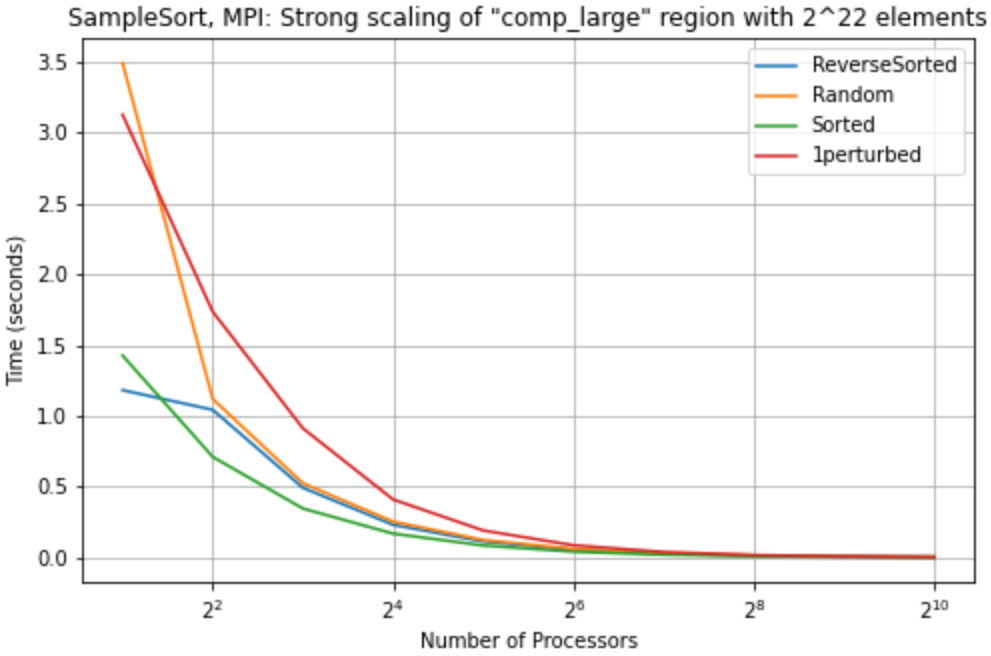

SampleSort, MPI: Strong scaling of "comm" region with 2^26 elements
Random
1perturbed
ReverseSorted
Sorted
Time (seconds)
Number of Processors

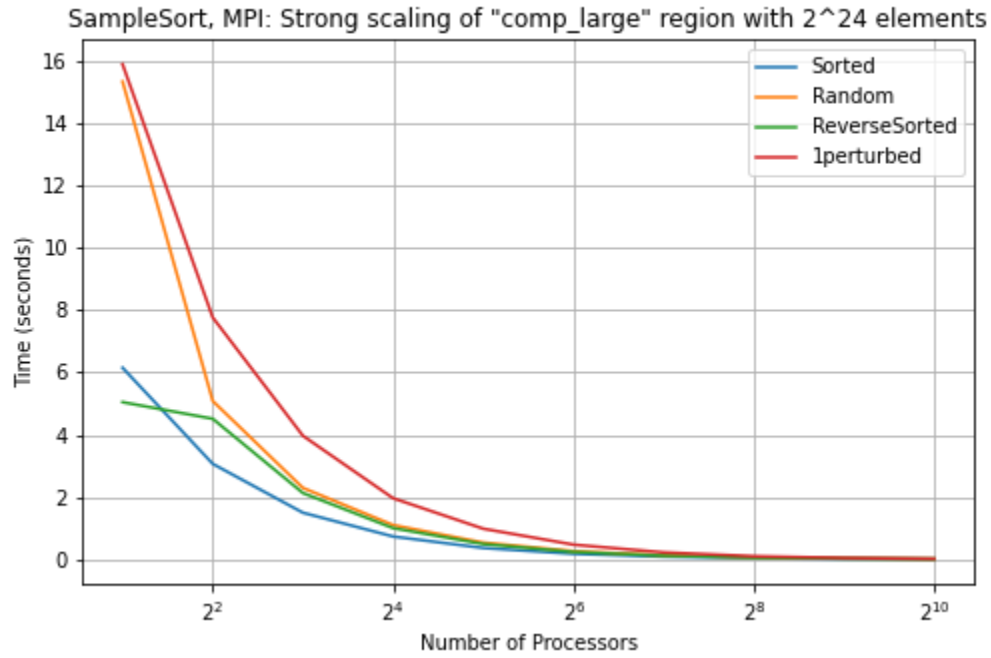

SampleSort, MPI: Strong scaling of "comm" region with 2^28 elements
Time (seconds)
Number of Processors
Sorted
1perturbed
ReverseSorted
Random
0
20
40
60
80
100
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$

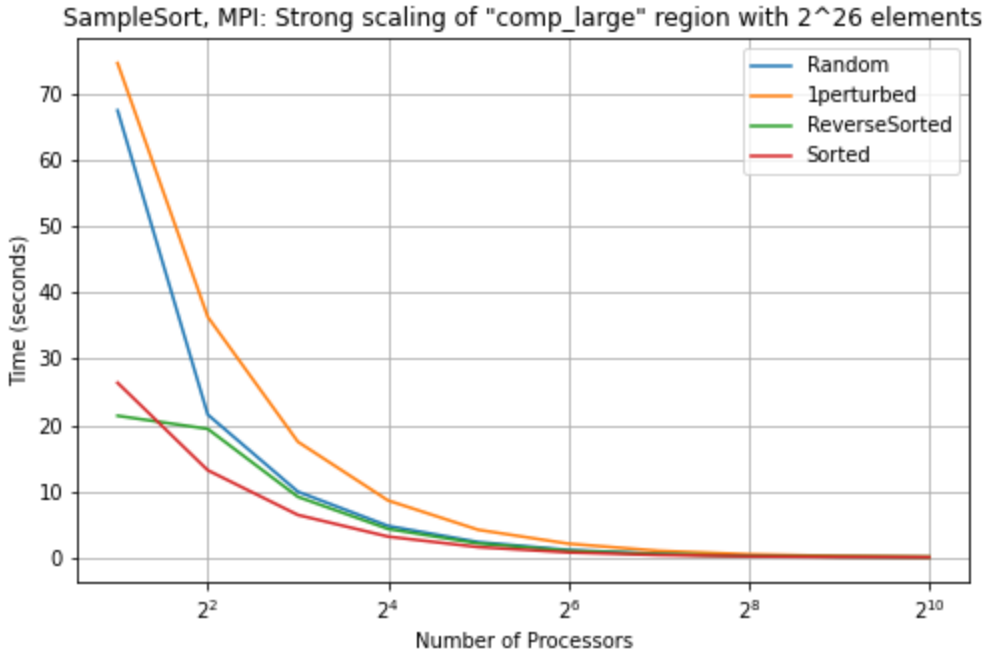

SampleSort, MPI: Strong scaling of "comp_large" region with 2^16 elements
Time (seconds)
Number of Processors
1perturbed
ReverseSorted
Random
Sorted
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
0.040
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$

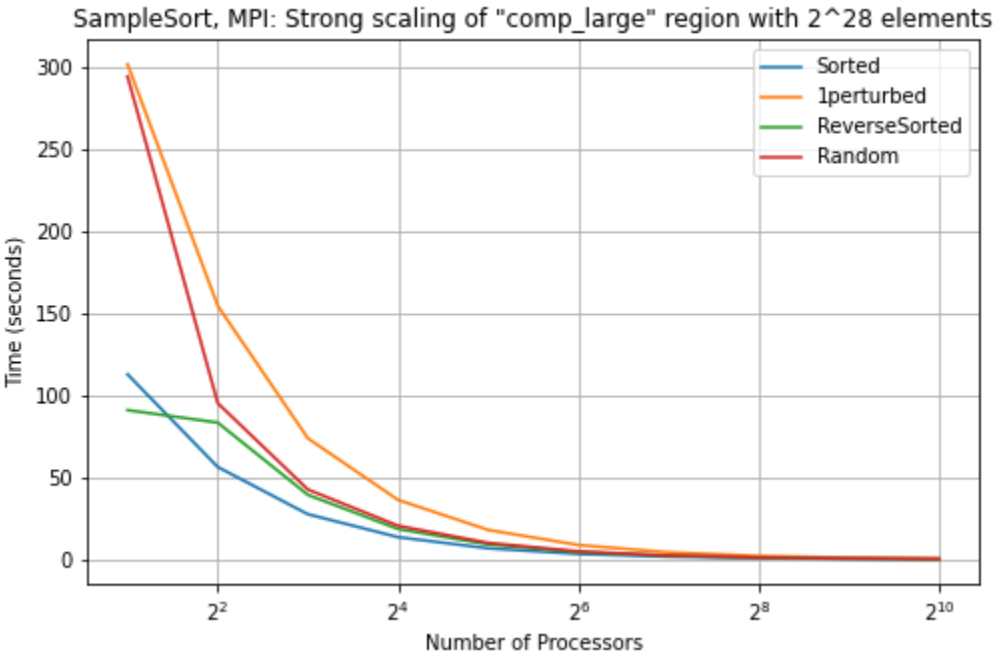

SampleSort, MPI: Strong scaling of "comp_large" region with 2^18 elements
ReverseSorted
1perturbed
Random
Sorted
Time (seconds)
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

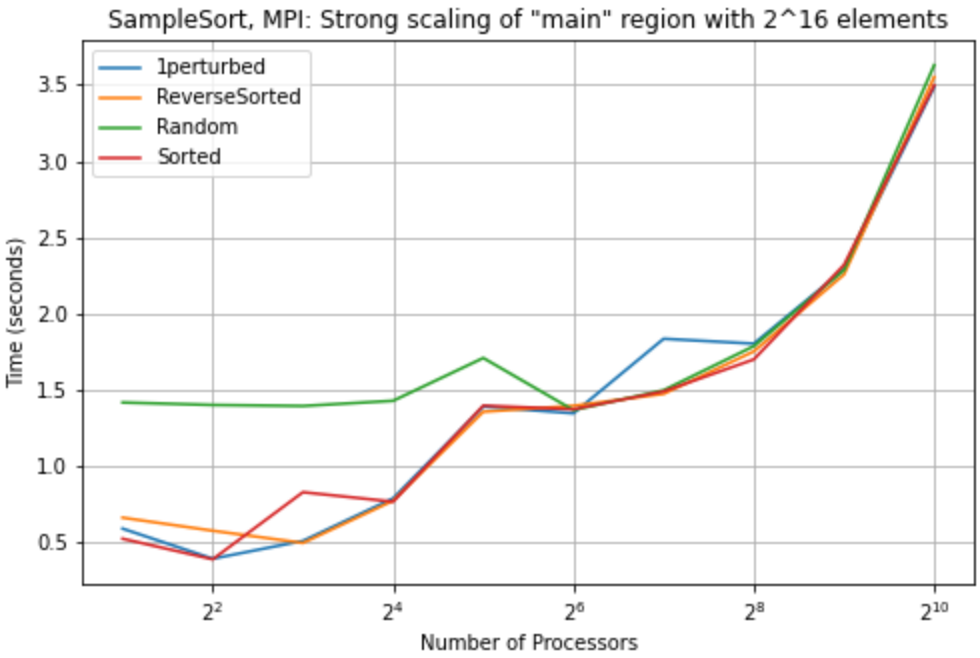

SampleSort, MPI: Strong scaling of "comp_large" region with 2^20 elements
Random
1perturbed
Sorted
ReverseSorted
Time (seconds)
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

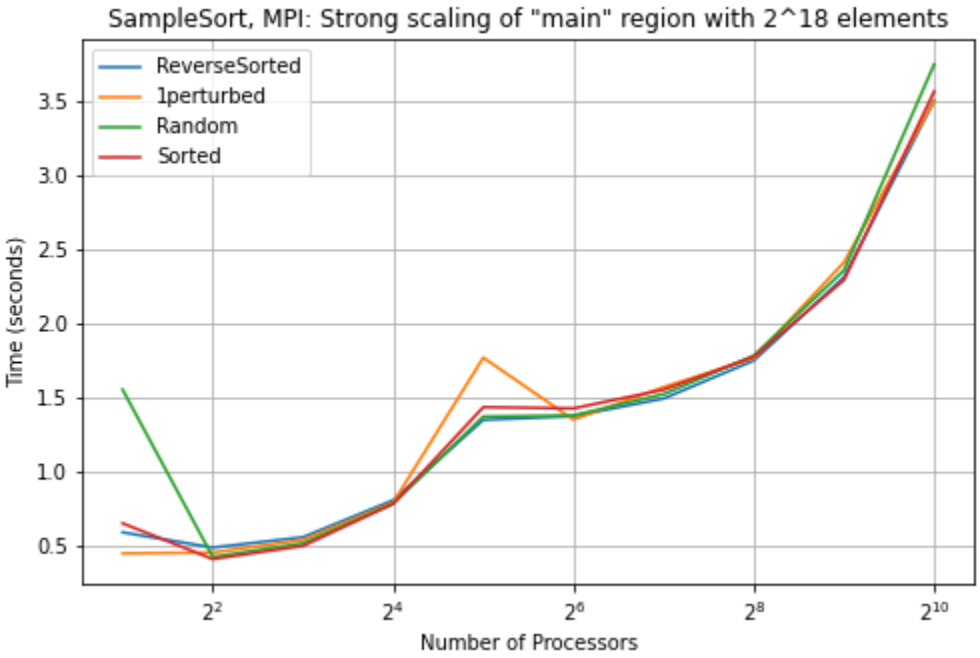

SampleSort, MPI: Strong scaling of "comp_large" region with 2^22 elements
ReverseSorted
Random
Sorted
1perturbed
Time (seconds)
Number of Processors

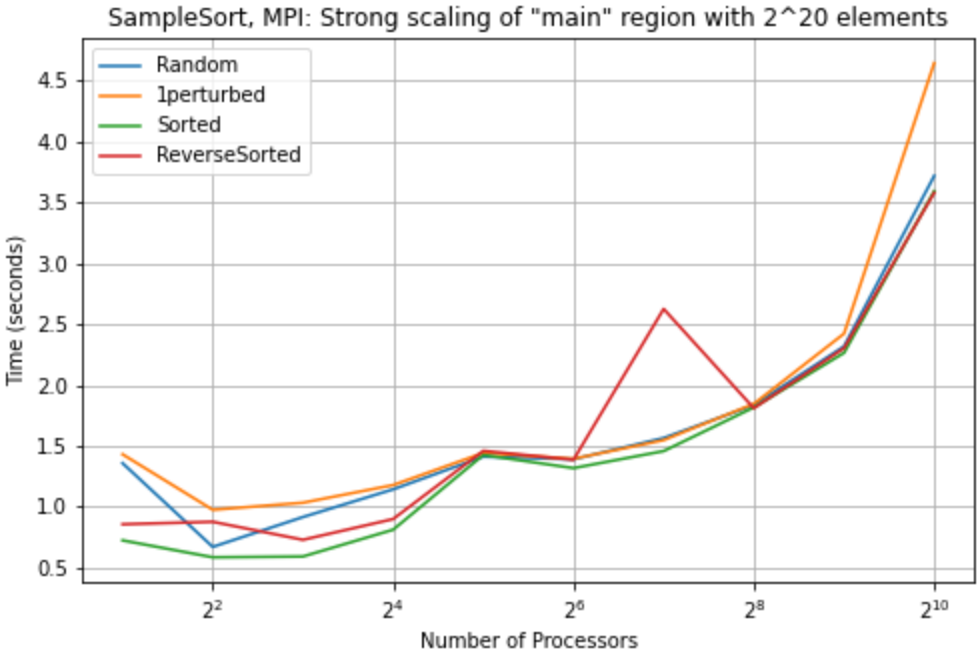

SampleSort, MPI: Strong scaling of "comp_large" region with 2^24 elements
Sorted
Random
ReverseSorted
1perturbed
Time (seconds)
Number of Processors

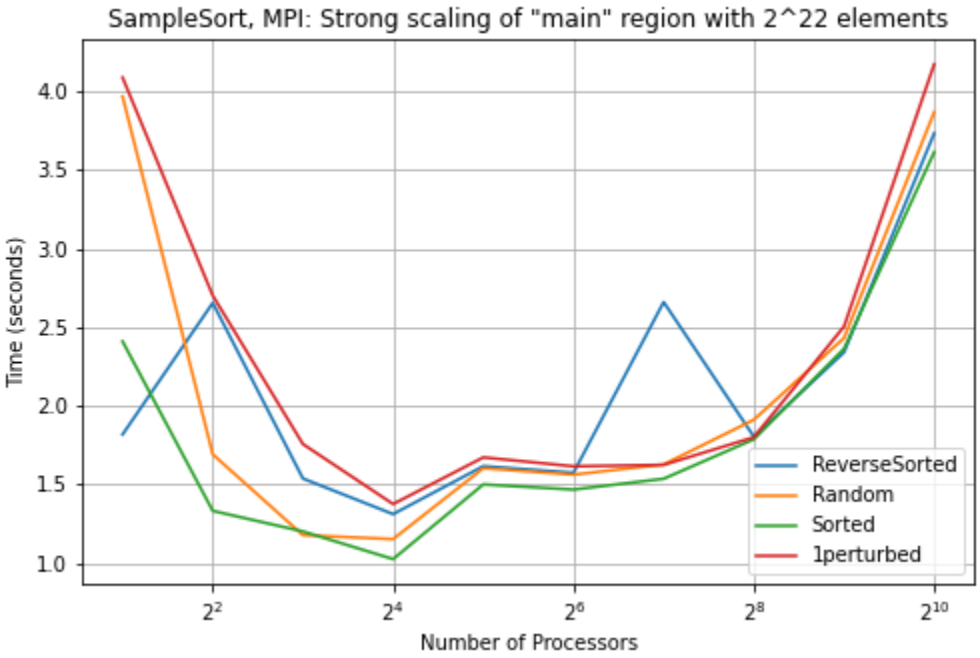

SampleSort, MPI: Strong scaling of "comp_large" region with 2^26 elements
Random
1perturbed
ReverseSorted
Sorted
Time (seconds)
0
10
20
30
40
50
60
70
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

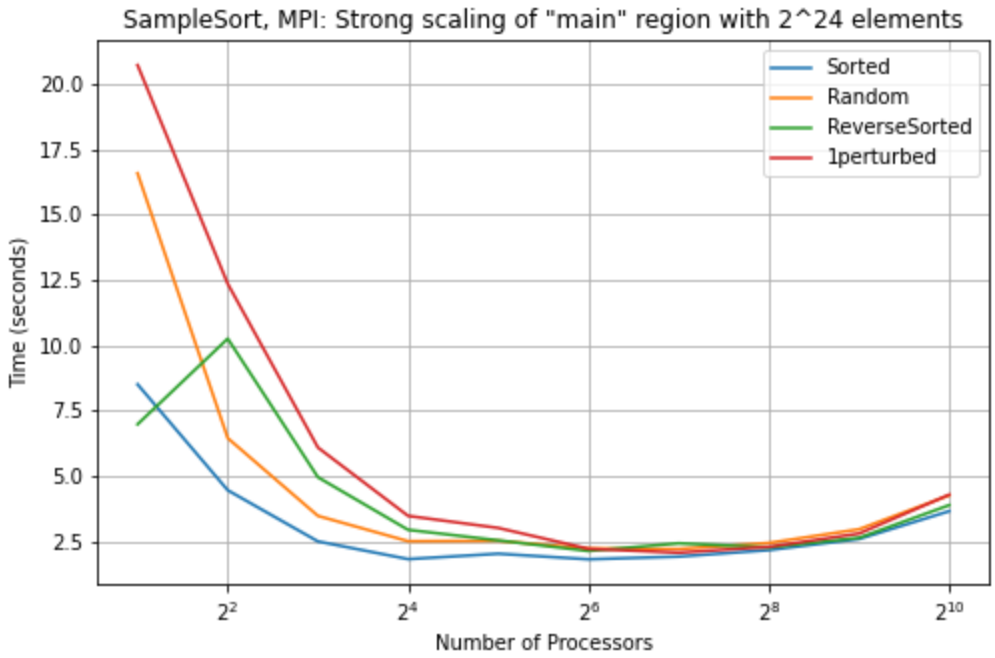

SampleSort, MPI: Strong scaling of "comp_large" region with 2^28 elements
Sorted
1perturbed
ReverseSorted
Random
Time (seconds)
0
50
100
150
200
250
300
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

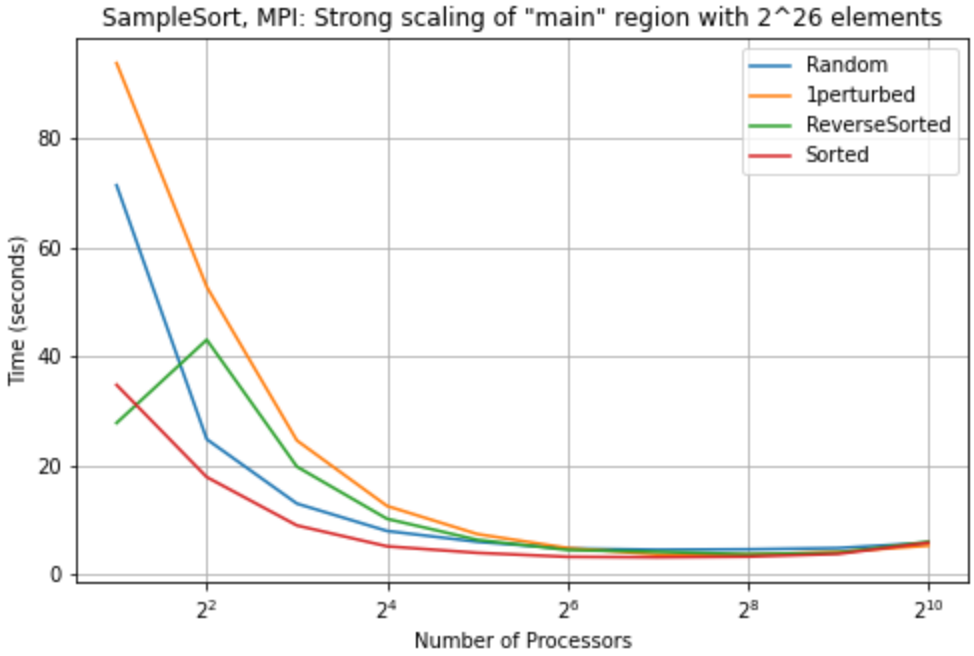

SampleSort, MPI: Strong scaling of "main" region with 2^16 elements
1perturbed
ReverseSorted
Random
Sorted
Time (seconds)
0.5
1.0
1.5
2.0
2.5
3.0
3.5
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

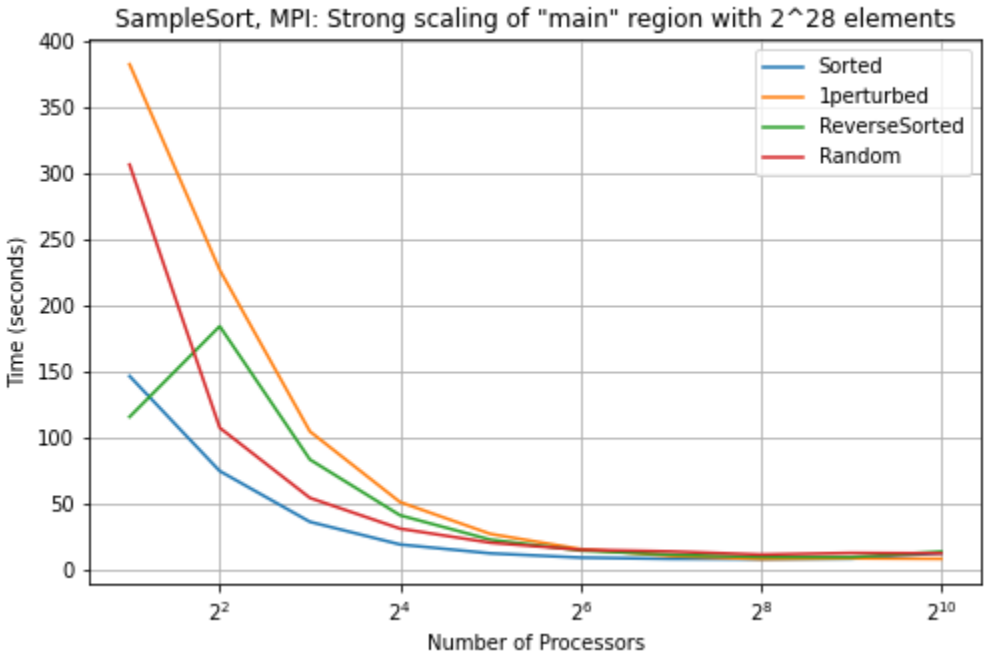

SampleSort, MPI: Strong scaling of "main" region with 2^18 elements
ReverseSorted
1perturbed
Random
Sorted
Time (seconds)
0.5
1.0
1.5
2.0
2.5
3.0
3.5
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors


SampleSort, MPI: Strong scaling of "main" region with 2^20 elements
Random
1perturbed
Sorted
ReverseSorted
Time (seconds)
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

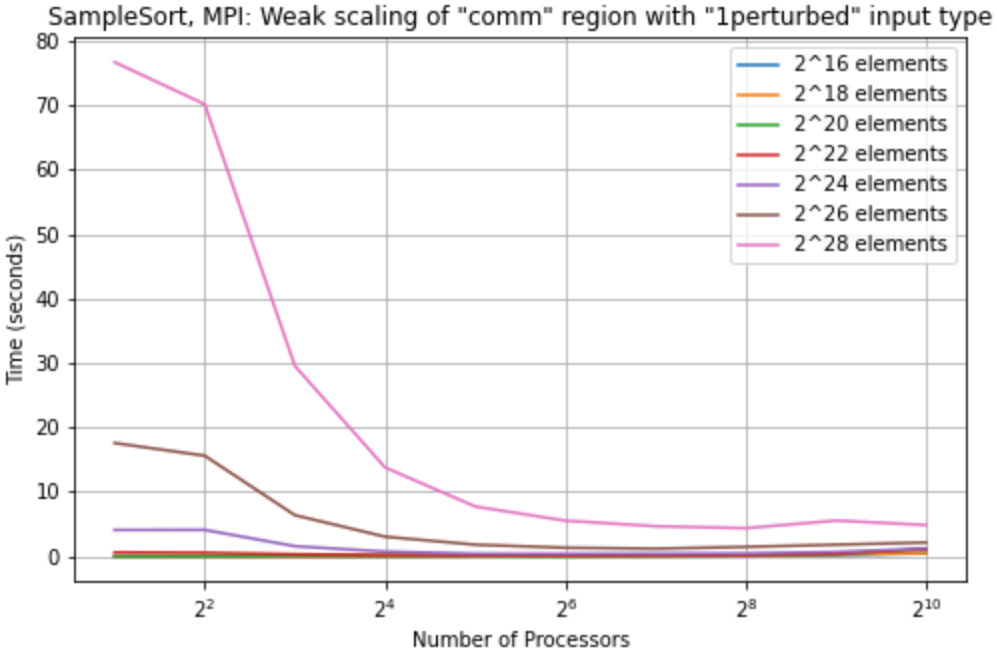

SampleSort, MPI: Strong scaling of "main" region with 2^22 elements
Time (seconds)
1.0
1.5
2.0
2.5
3.0
3.5
4.0
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors
ReverseSorted
Random
Sorted
1perturbed

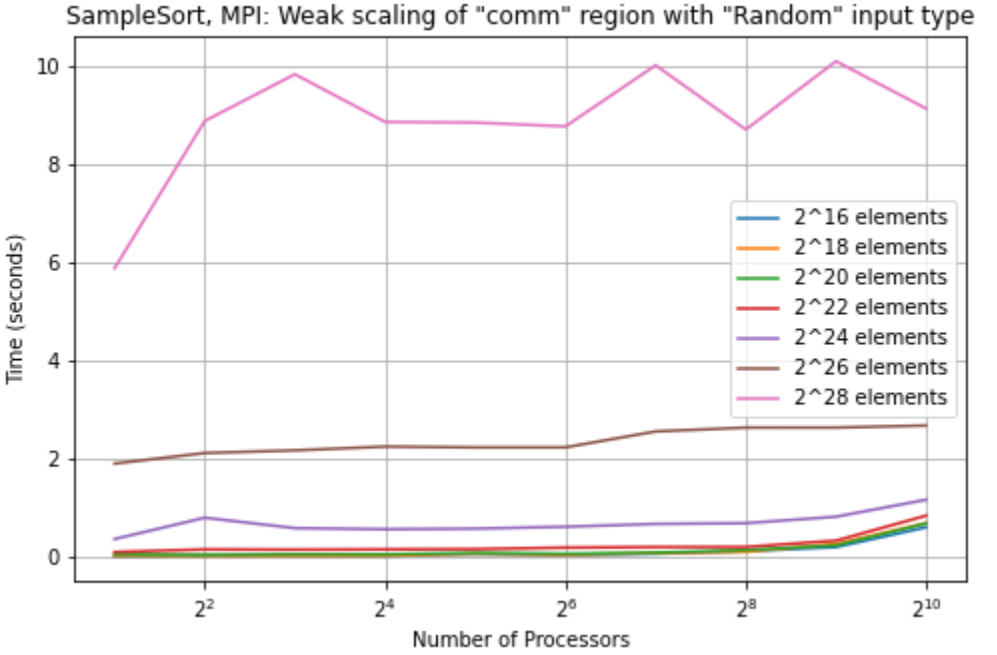

SampleSort, MPI: Strong scaling of "main" region with 2^24 elements
Sorted
Random
ReverseSorted
1perturbed
Time (seconds)
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

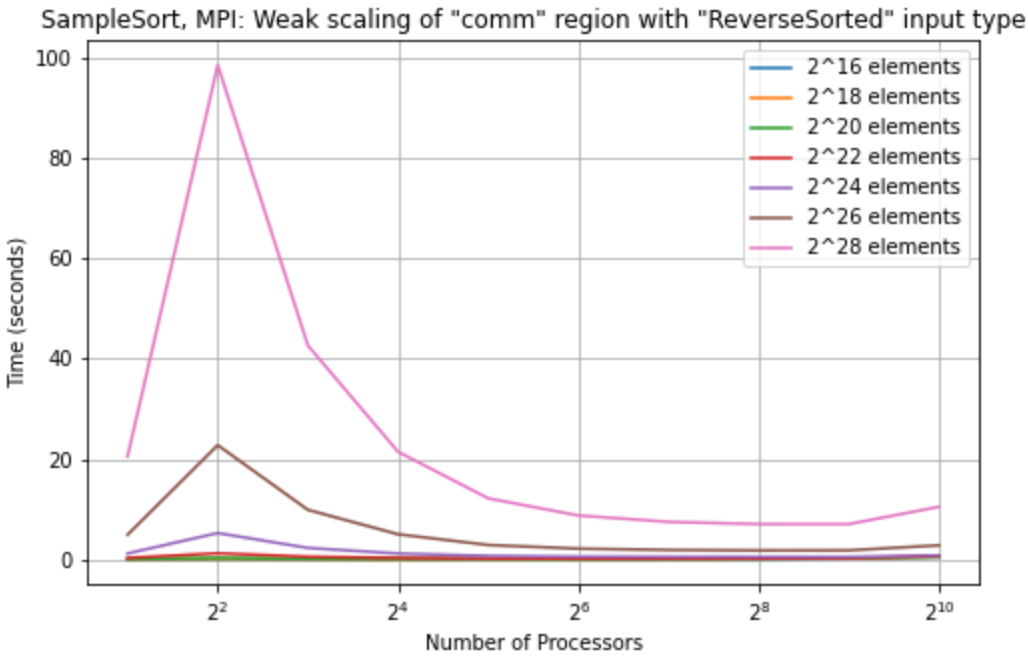


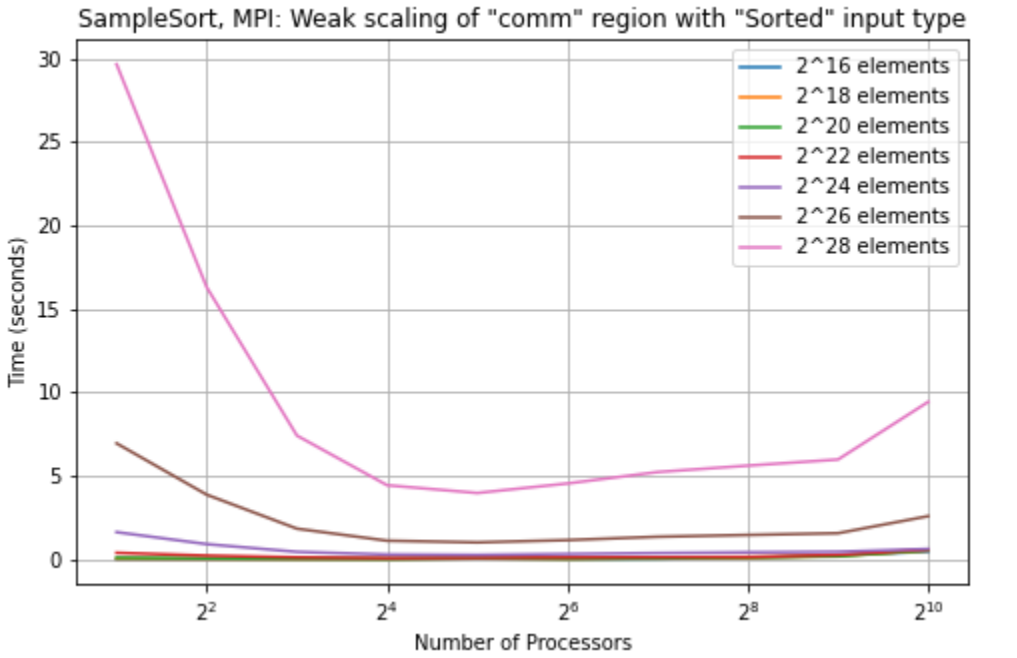



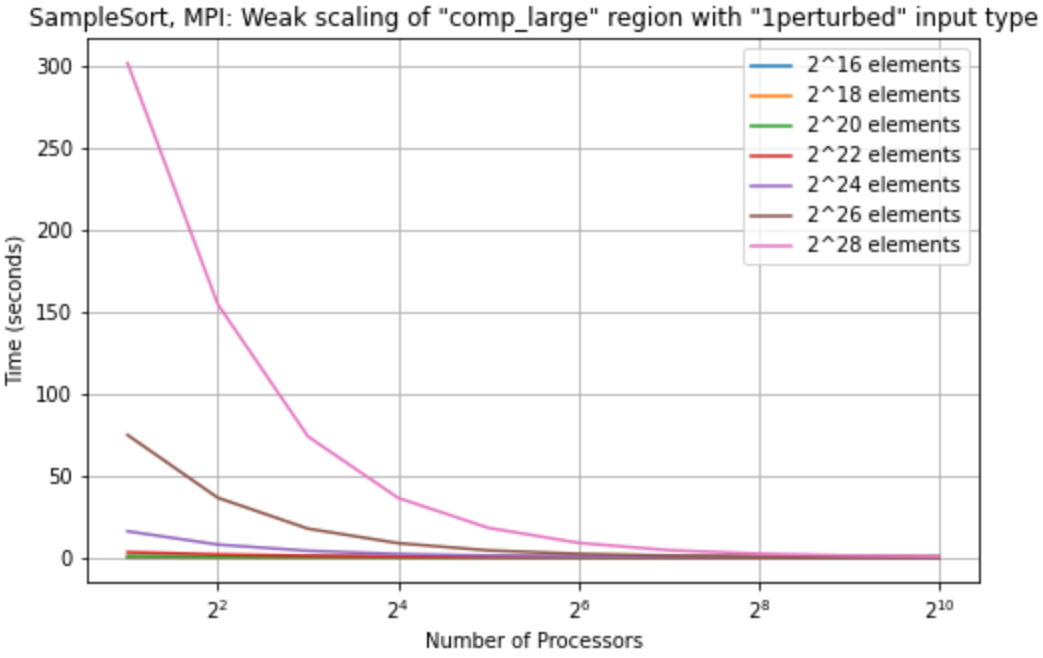

- **<u>Weak Scaling:</u>**

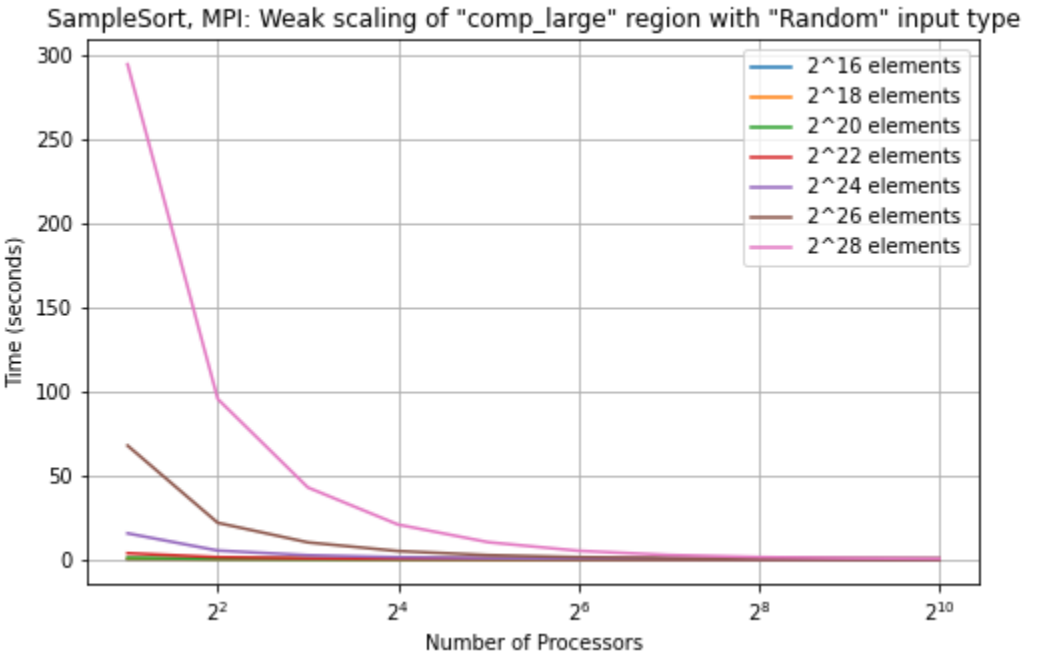

SampleSort, MPI: Weak scaling of "comm" region with "1perturbed" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

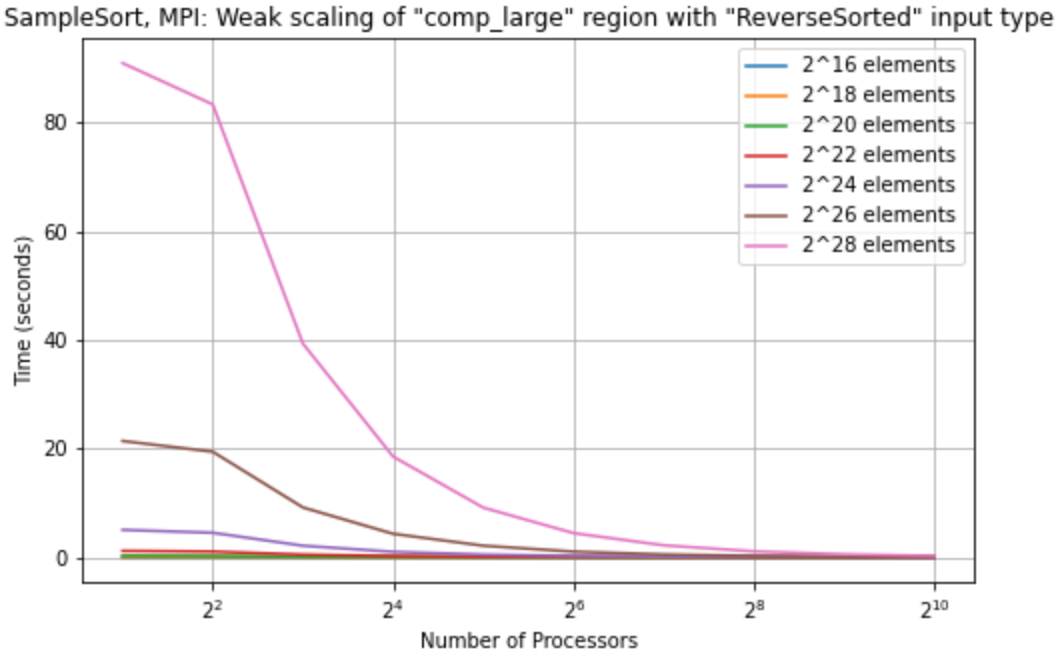

SampleSort, MPI: Weak scaling of "comm" region with "Random" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

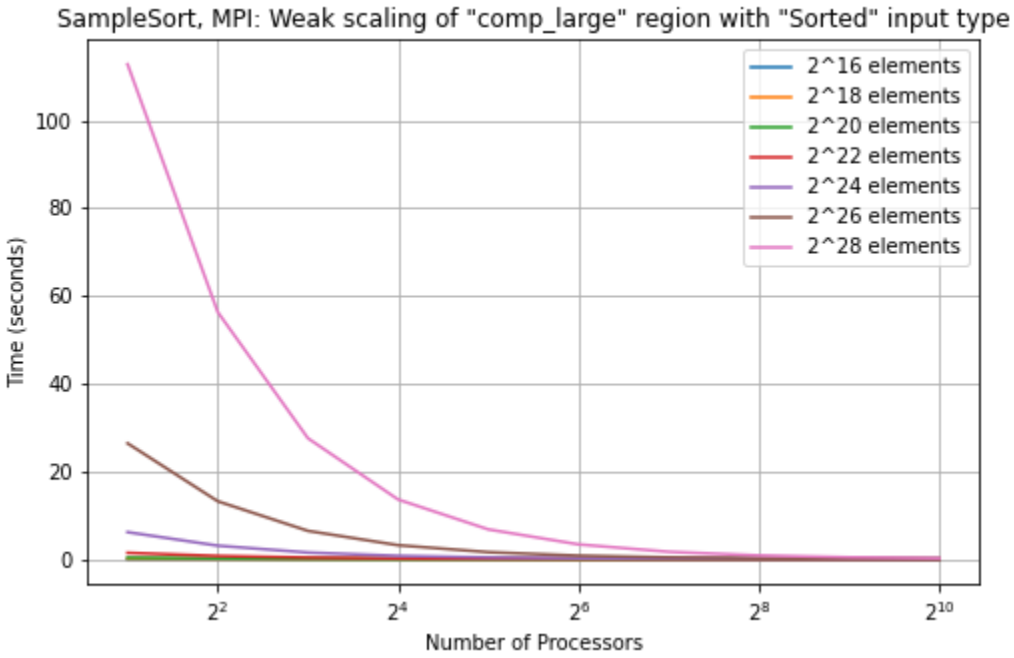

SampleSort, MPI: Weak scaling of "comm" region with "ReverseSorted" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

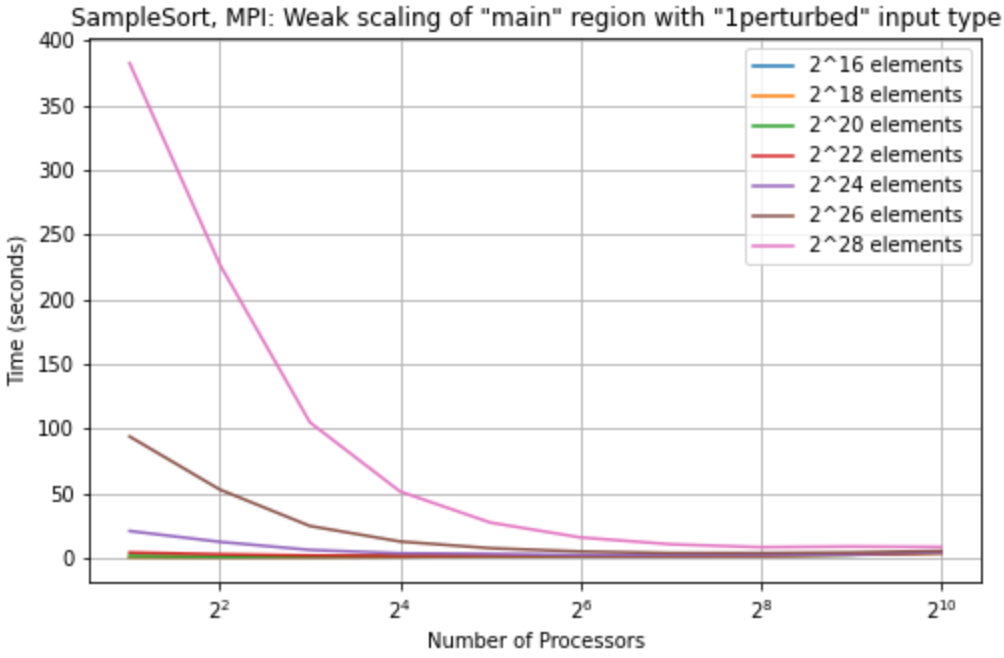

SampleSort, MPI: Weak scaling of "comm" region with "Sorted" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

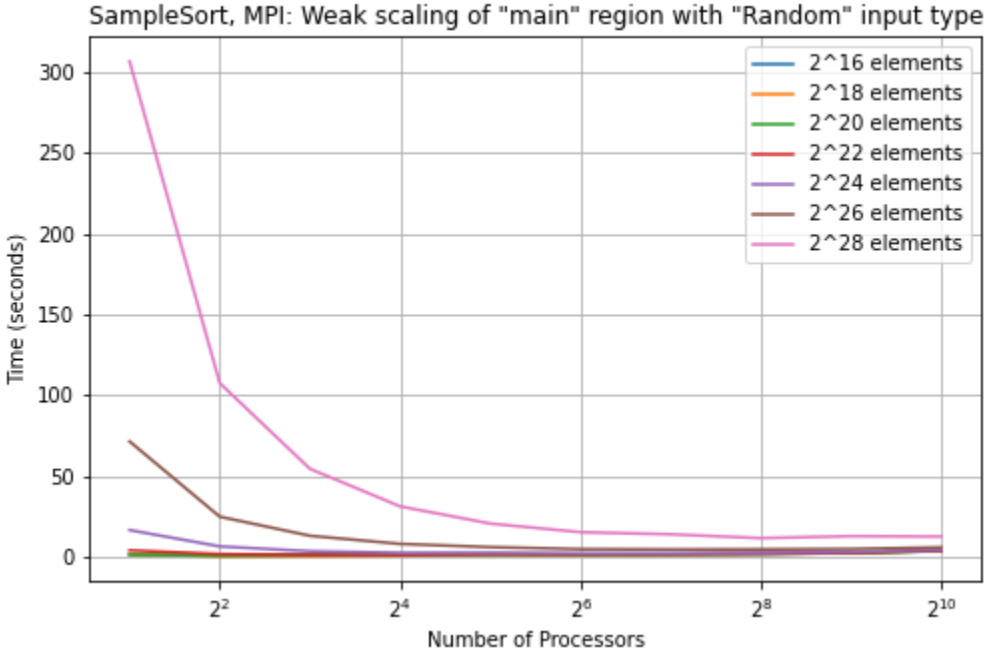

SampleSort, MPI: Weak scaling of "comp_large" region with "1perturbed" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
0
50
100
150
200
250
300
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$

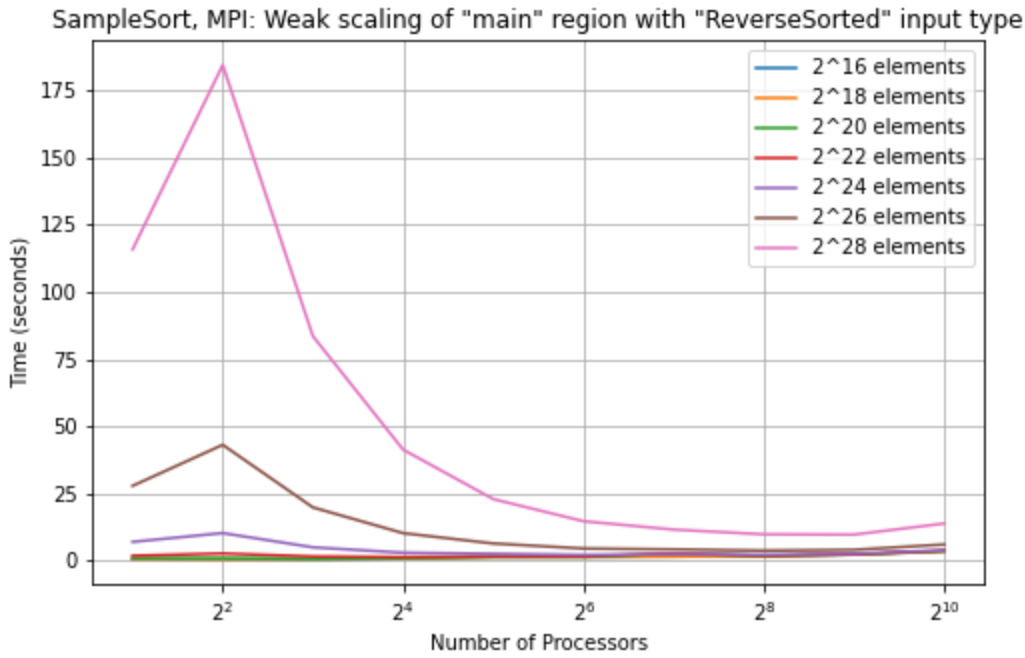

SampleSort, MPI: Weak scaling of "comp_large" region with "Random" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
0
50
100
150
200
250
300
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$

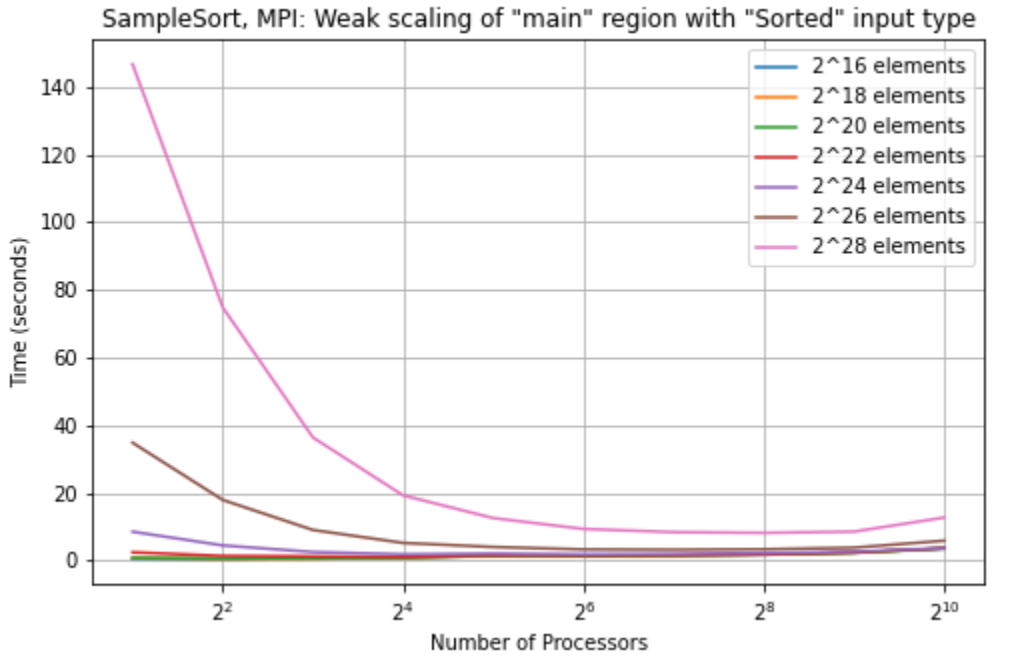

SampleSort, MPI: Weak scaling of "comp_large" region with "ReverseSorted" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
0
20
40
60
80
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$


SampleSort, MPI: Weak scaling of "comp_large" region with "Sorted" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
0
20
40
60
80
100
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$


SampleSort, MPI: Weak scaling of "main" region with "1perturbed" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements


SampleSort, MPI: Weak scaling of "main" region with "Random" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements




**Analysis:**

One can see from the strong scaling graphs, that with an increasing number of processors, the execution time of the large computation decreased for all input types and all input sizes. The decrease in time was more pronounced with greater input sizes.

At the extreme, it took 300 seconds to finish large computation with two processors for Random and 1% Perturbed and it took less than one second to do the same with 1024 processors.

One can see from the strong scaling graphs, that with an increasing number of processors, the execution time of the whole program increased for input sizes of 2^16, 2^18, and 2^20. There was an initial decrease and then increase (the saddle point being 2^6 processors) for input size of 2^22. There was a decrease for 2^24, 2^26, and 2^28 elements. This makes sense as the communication overhead for higher processors is expensive and outweighs the benefits of parallel computation for smaller input sizes.

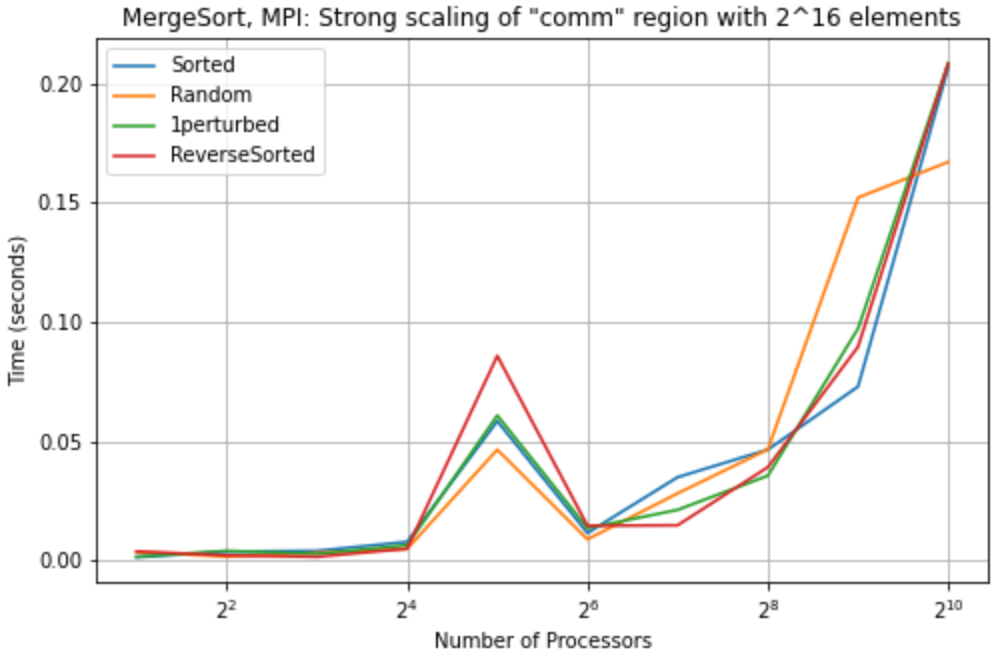
One can see from the strong scaling graphs, that with an increasing number of processors, the execution time of the communication increased for input sizes of 2^16, 2^18, and 2^20. There was an initial decrease and then increase (the saddle point being 2^6 processors) for input size of 2^22. There was a decrease for 2^24, 2^26, and 2^28 elements. This makes sense as the communication overhead for higher processors is expensive and outweighs the benefits of parallel computation for smaller input sizes.

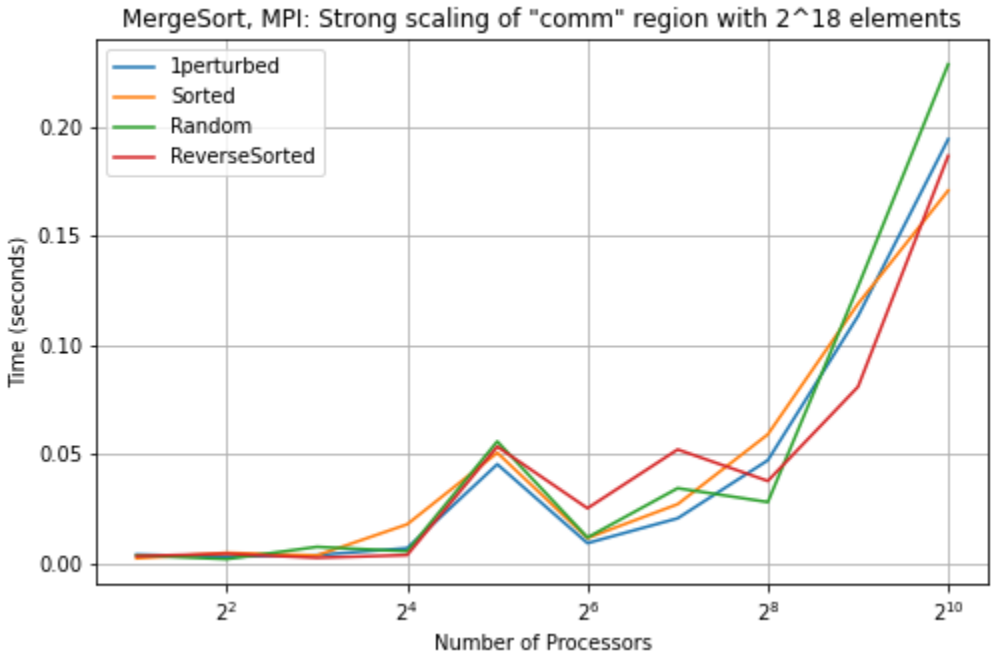
One can see from the weak scaling graphs, that with an increasing number of processors, the execution time of both the large computation and whole program decreased for all input types and all input sizes. The decrease was more pronounced with more greater input sizes.

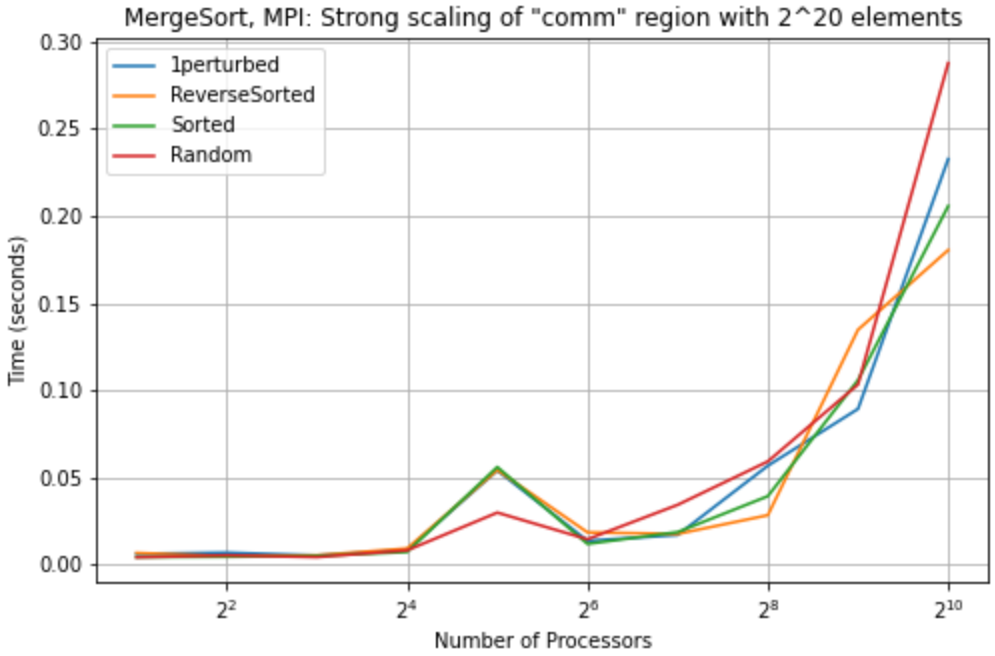
One can see from the speed up graphs, that with an increasing number of processors, the execution time of both the large computation and whole program increased for all input types but only large input sizes like 2^28 and 2^26. We can conclude that the speed up benefits only occurred for input sizes where the benefits of parallel computation outweighed the communication overhead in a higher number of processors. For the whole program speed-up, depending on the type of input type, I was achieving up to 40x speed up. Considering the ideal speed up is 512, I am content with this speed up. For large computational speed up, while I did achieve speed-up times of up to 1000, the explanation for this occurrence is because our baseline time was 300 seconds (with two processors) and the time with 1024 processors was less than a second. I expect this to be because Sample Sort works extremely ineffectively with a small sample size and since my sample size was always one less than the number of processors, my baseline time used one sample. Having an extremely large time for my baseline would create unusually high speed up times.

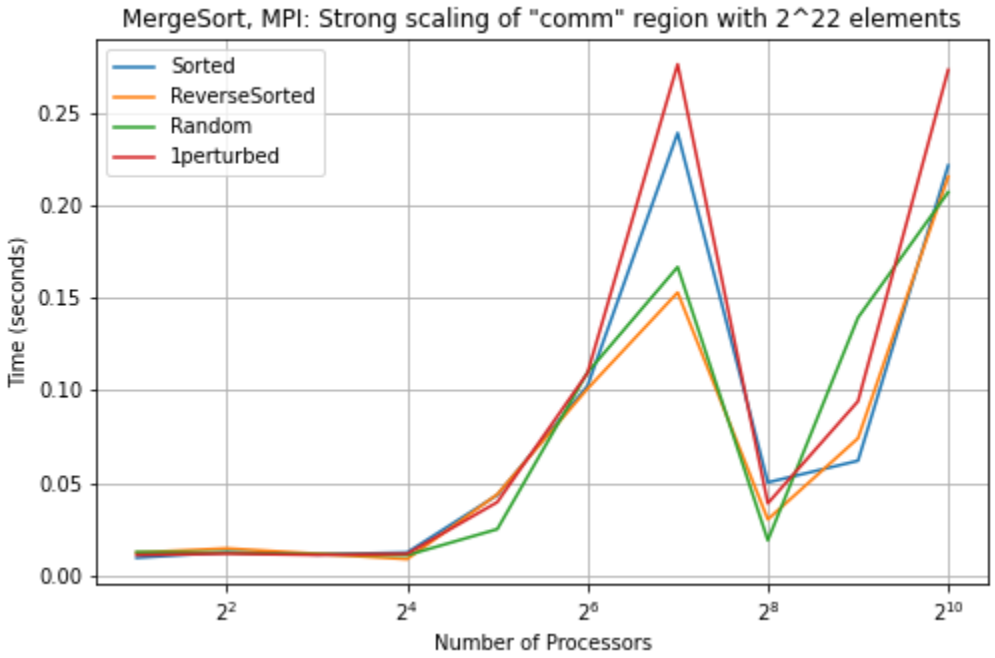
# Merge Sort MPI

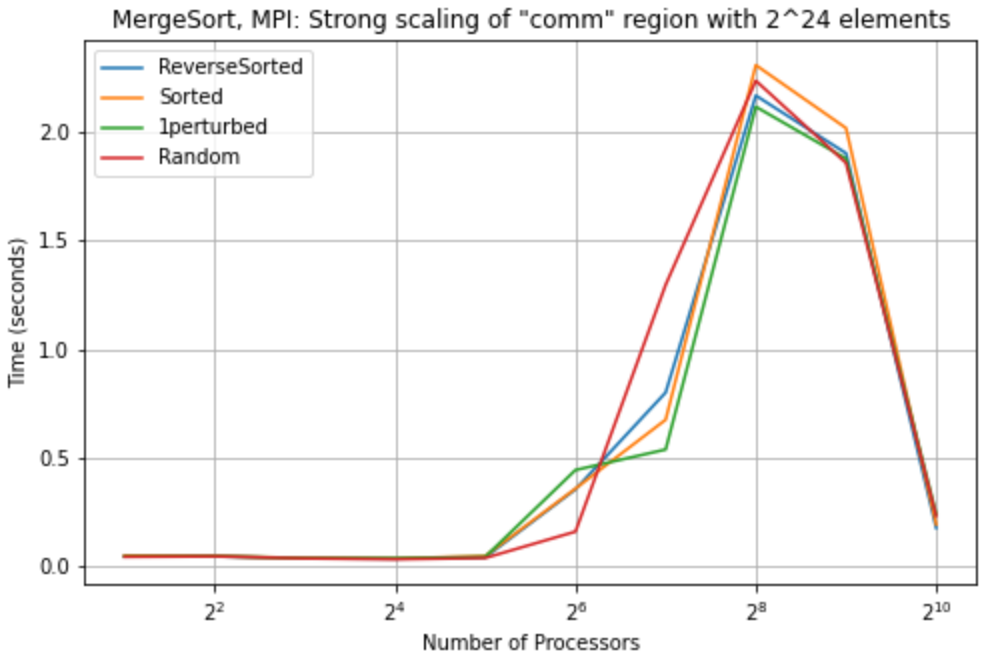
**Graphs:**

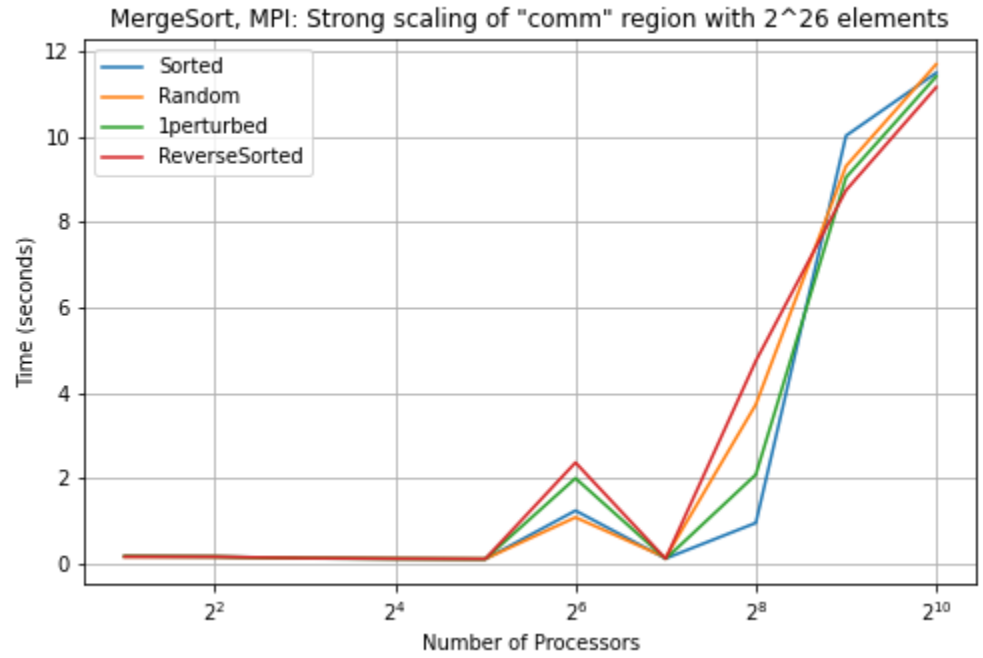

MergeSort, MPI: Strong scaling of "comm" region with 2^16 elements
Sorted
Random
1perturbed
ReverseSorted
Time (seconds)
0.00
0.05
0.10
0.15
0.20
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

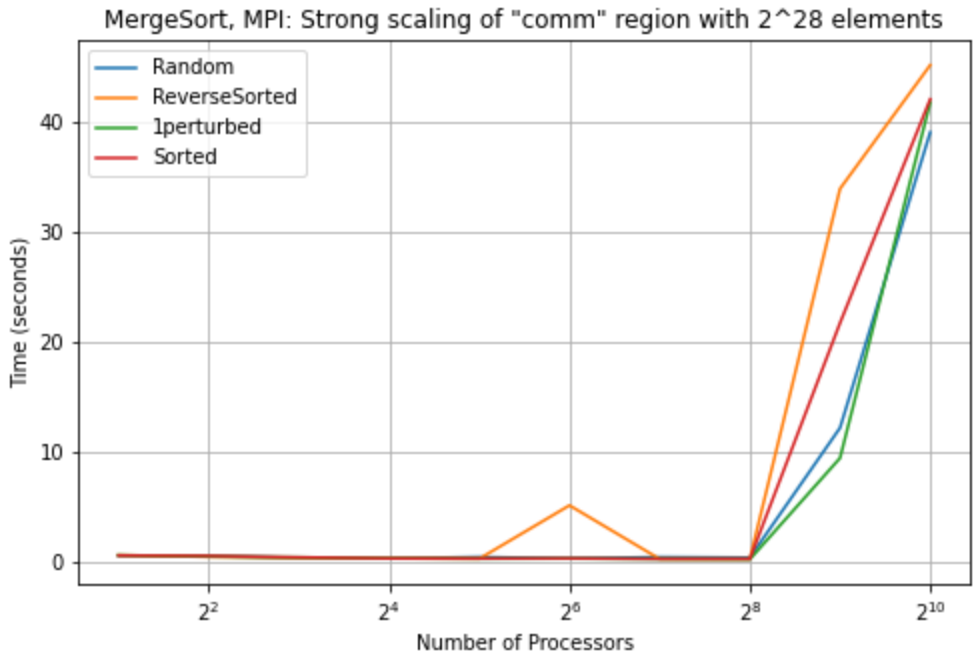

MergeSort, MPI: Strong scaling of "comm" region with 2^18 elements
1perturbed
Sorted
Random
ReverseSorted
Time (seconds)
0.00
0.05
0.10
0.15
0.20
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

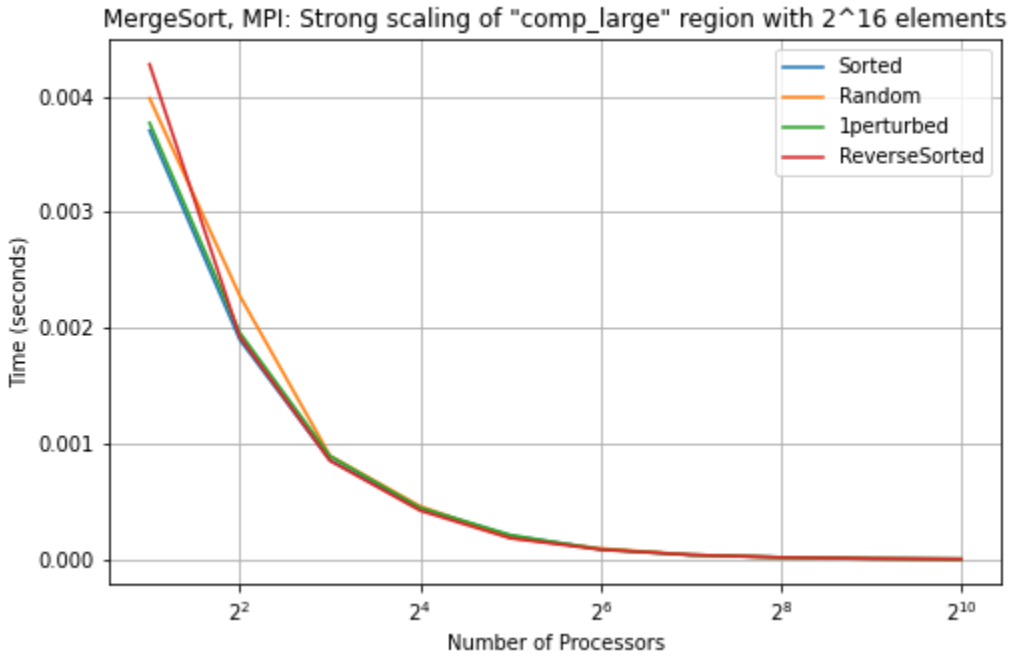

MergeSort, MPI: Strong scaling of "comm" region with 2^20 elements
1perturbed
ReverseSorted
Sorted
Random
Time (seconds)
0.30
0.25
0.20
0.15
0.10
0.05
0.00
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

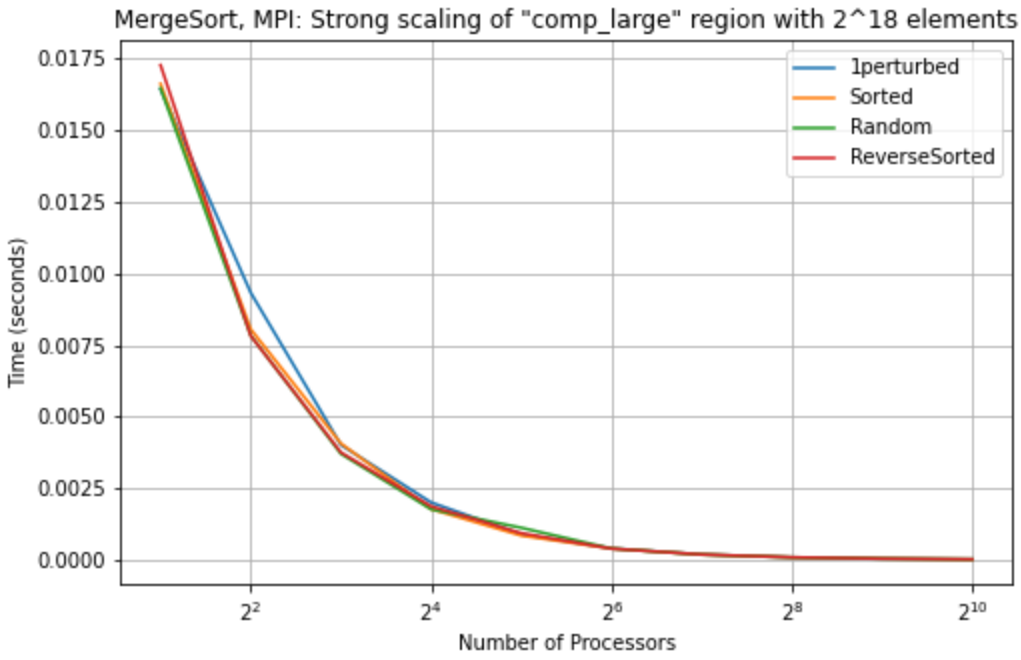

MergeSort, MPI: Strong scaling of "comm" region with 2^22 elements
Sorted
ReverseSorted
Random
1perturbed
Time (seconds)
0.25
0.20
0.15
0.10
0.05
0.00
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

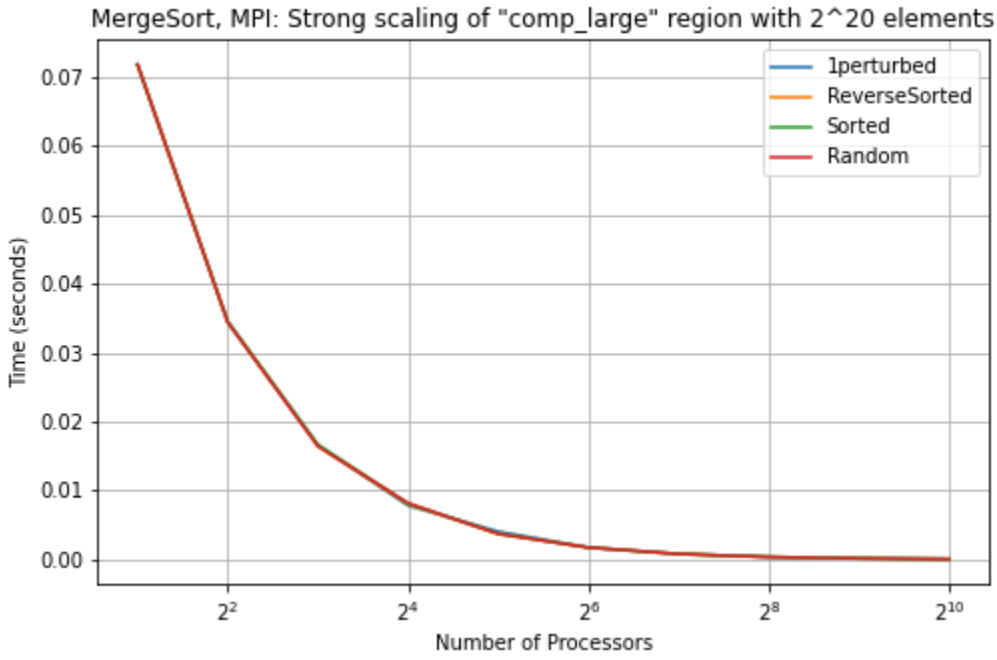

MergeSort, MPI: Strong scaling of "comm" region with 2^24 elements
ReverseSorted
Sorted
1perturbed
Random
Time (seconds)
0.0
0.5
1.0
1.5
2.0
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

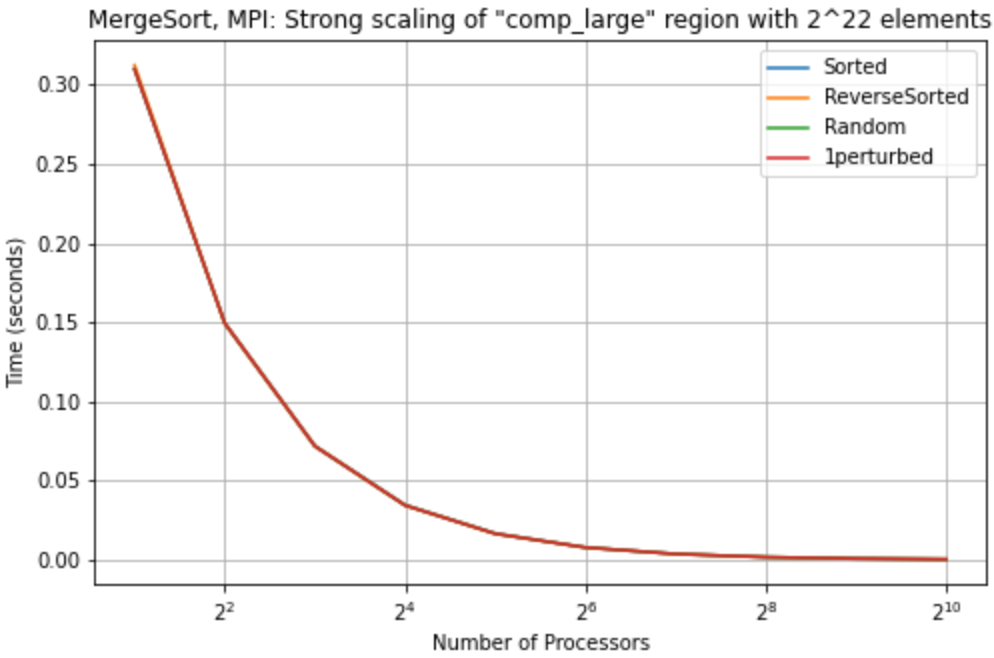

MergeSort, MPI: Strong scaling of "comm" region with 2^26 elements
Sorted
Random
1perturbed
ReverseSorted
Time (seconds)
0
2
4
6
8
10
12
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

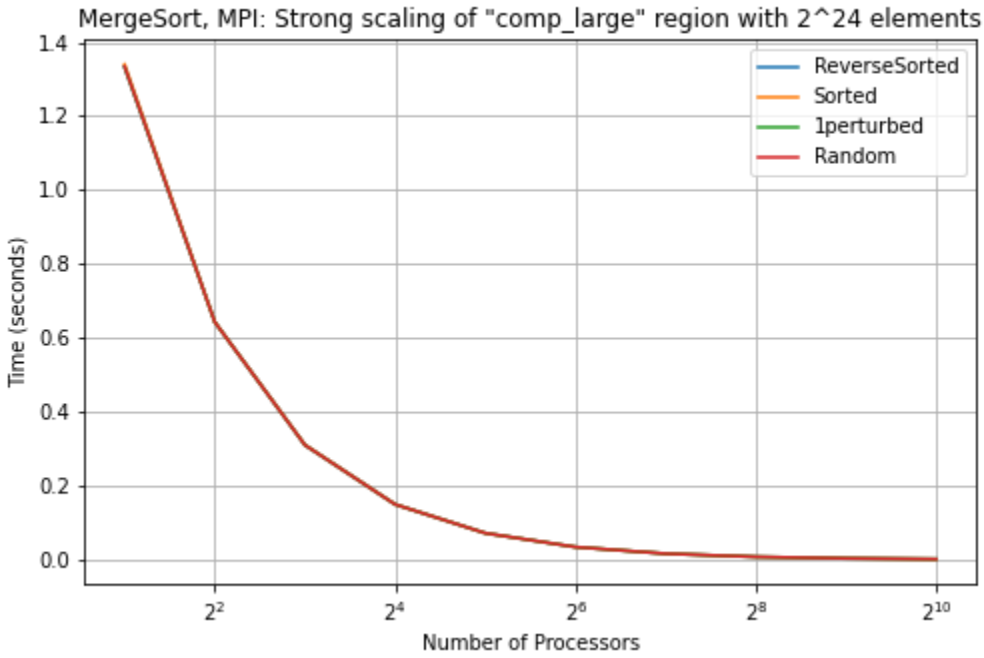

MergeSort, MPI: Strong scaling of "comm" region with 2^28 elements
Random
ReverseSorted
1perturbed
Sorted
Time (seconds)
0
10
20
30
40
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

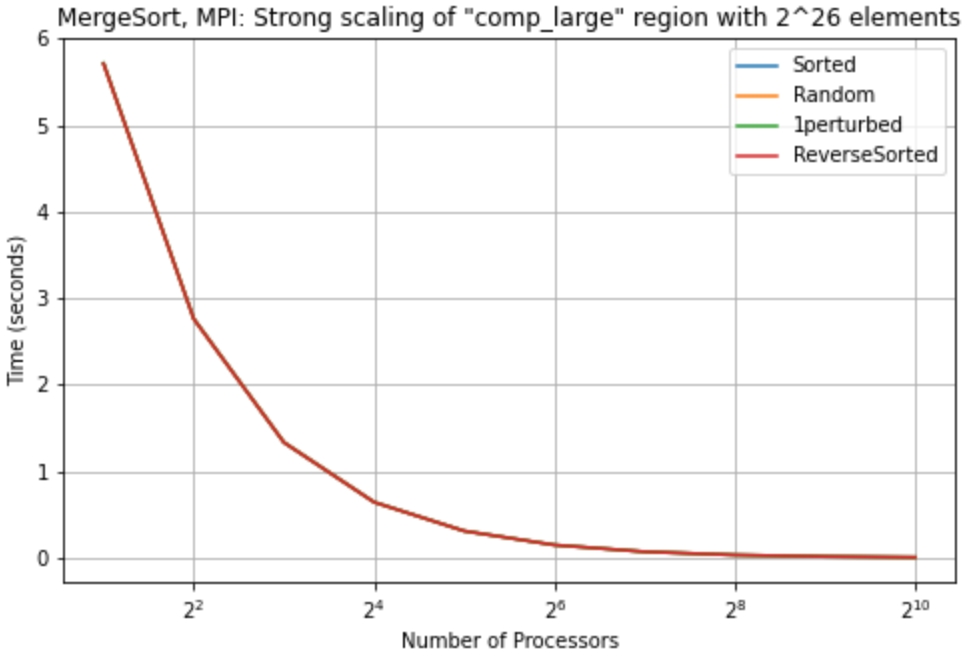

MergeSort, MPI: Strong scaling of "comp_large" region with 2^16 elements
Sorted
Random
1perturbed
ReverseSorted
Time (seconds)
0.000
0.001
0.002
0.003
0.004
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

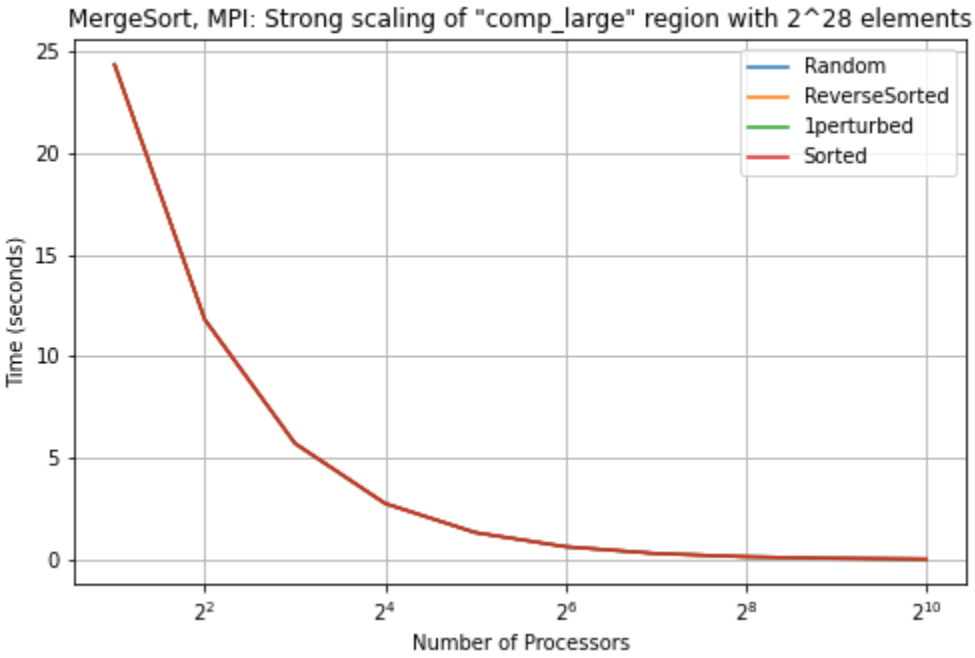

MergeSort, MPI: Strong scaling of "comp_large" region with 2^18 elements
Time (seconds)
Number of Processors
1perturbed
Sorted
Random
ReverseSorted
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$

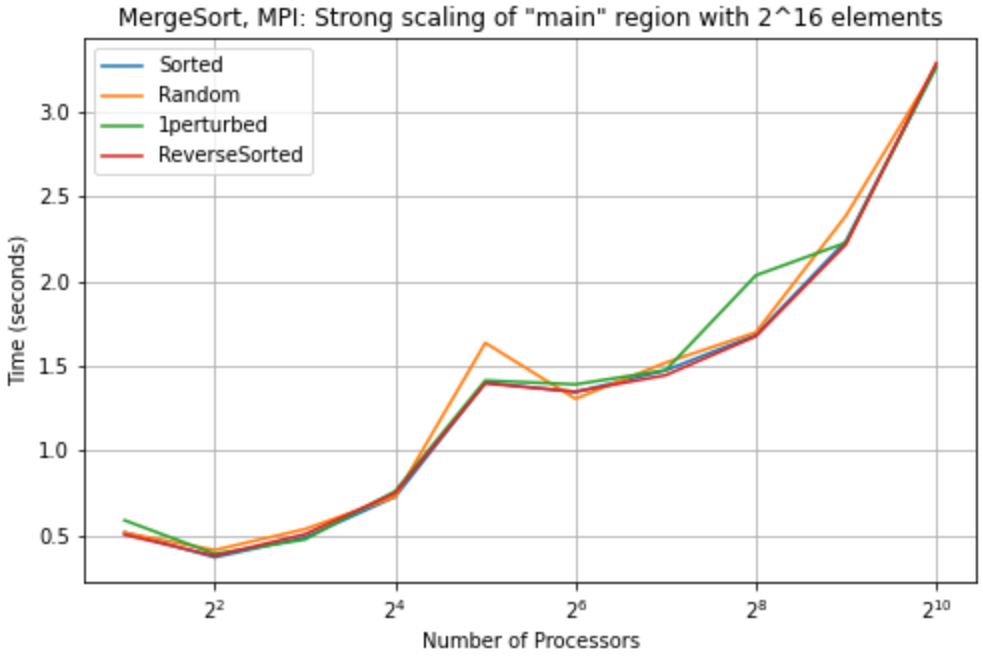

MergeSort, MPI: Strong scaling of "comp_large" region with 2^20 elements
Time (seconds)
Number of Processors
1perturbed
ReverseSorted
Sorted
Random
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$

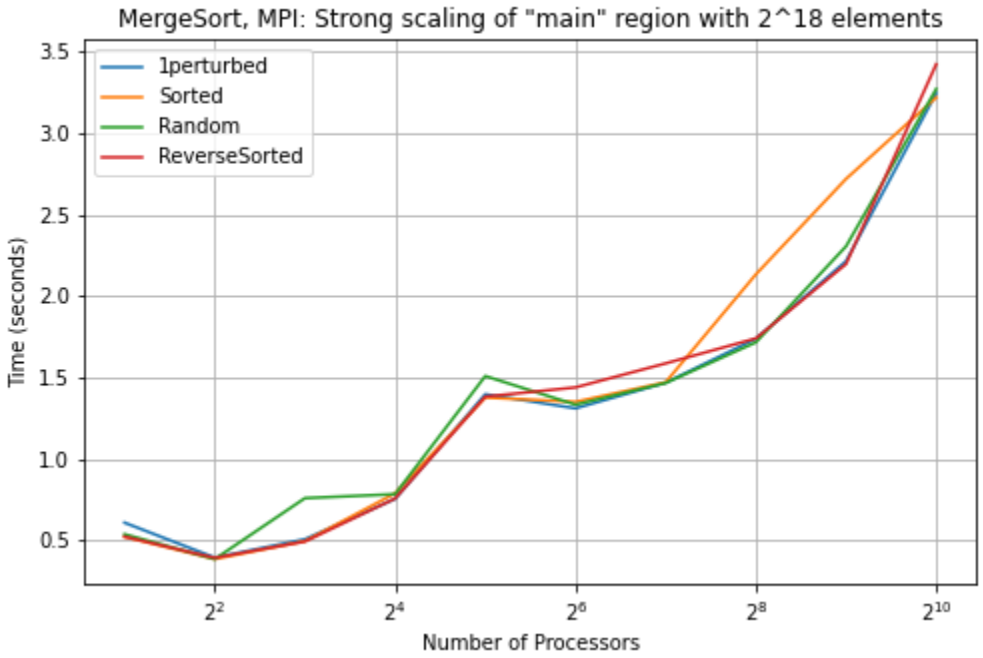

MergeSort, MPI: Strong scaling of "comp_large" region with 2^22 elements
Sorted
ReverseSorted
Random
1perturbed
Time (seconds)
0.30
0.25
0.20
0.15
0.10
0.05
0.00
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

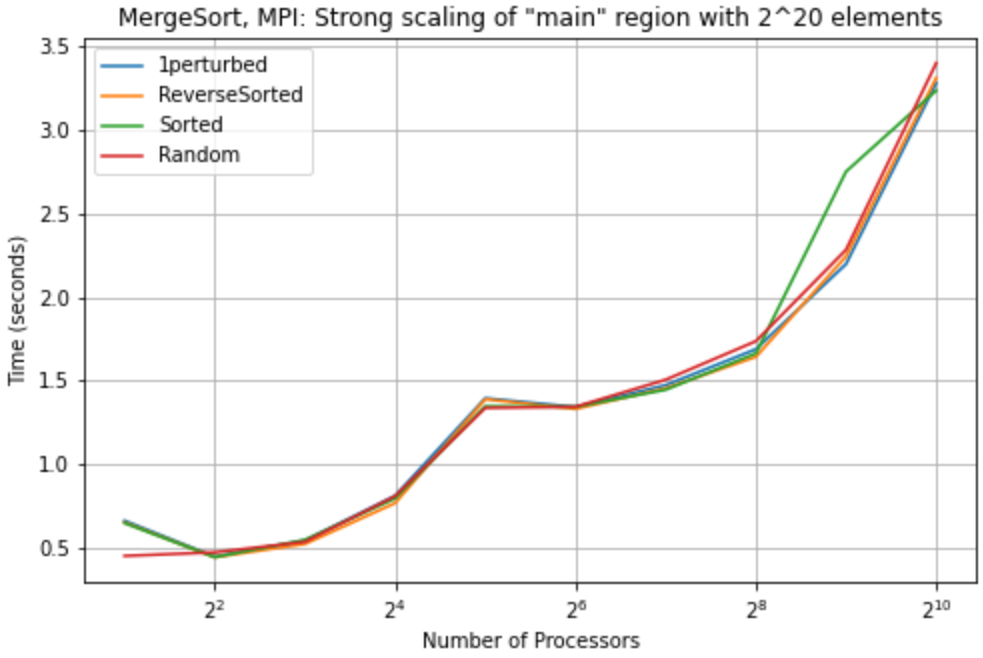

MergeSort, MPI: Strong scaling of "comp_large" region with 2^24 elements
ReverseSorted
Sorted
1perturbed
Random
Time (seconds)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

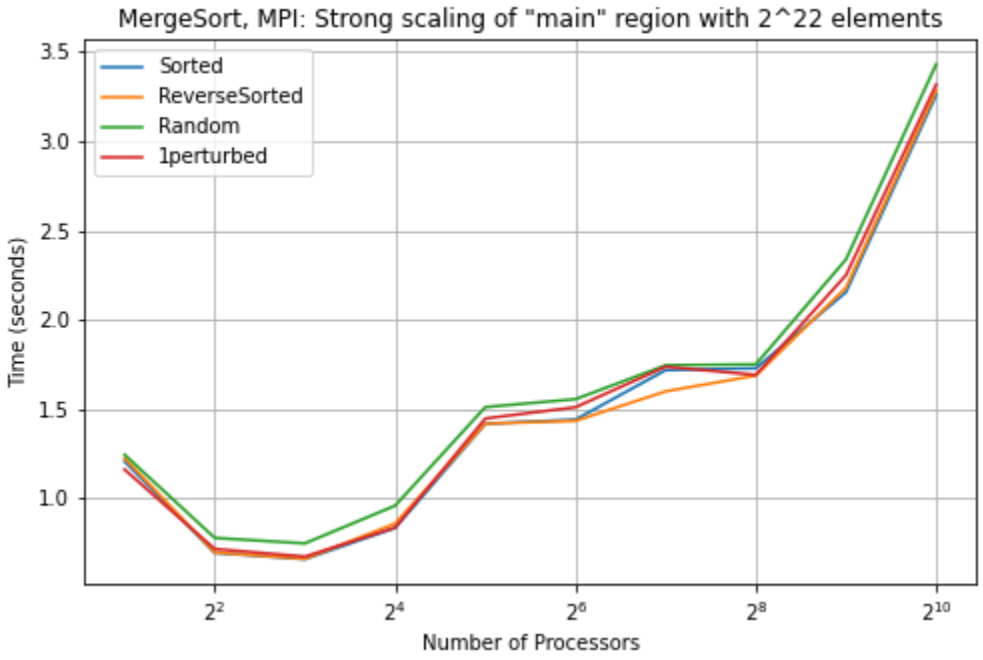

MergeSort, MPI: Strong scaling of "comp_large" region with 2^26 elements
Sorted
Random
1perturbed
ReverseSorted
Time (seconds)
Number of Processors
0
1
2
3
4
5
6
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$

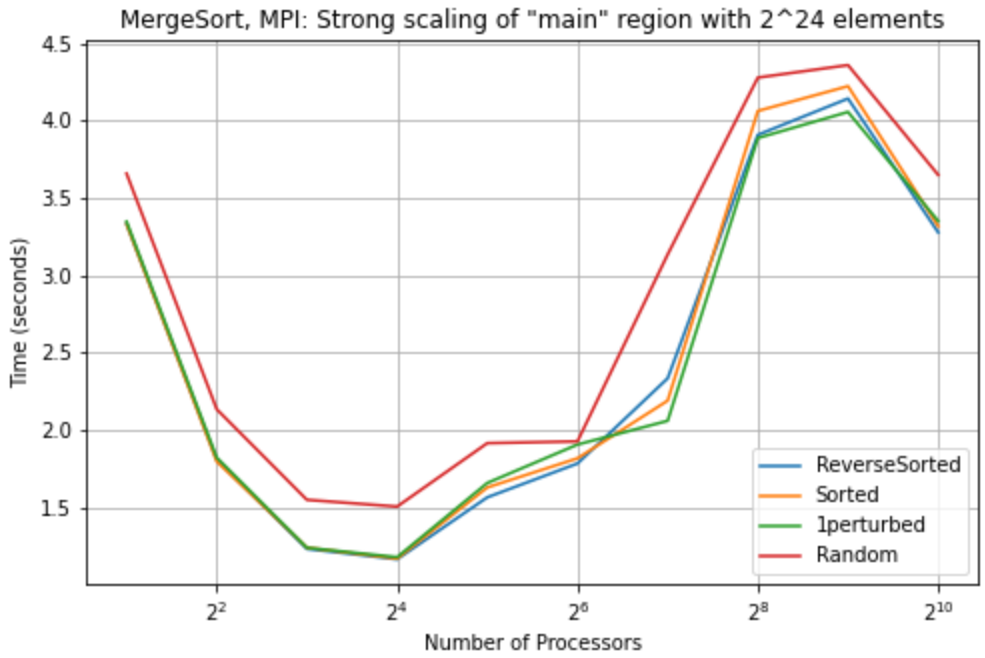

MergeSort, MPI: Strong scaling of "comp_large" region with 2^28 elements
Random
ReverseSorted
1perturbed
Sorted
Time (seconds)
Number of Processors
0
5
10
15
20
25
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$

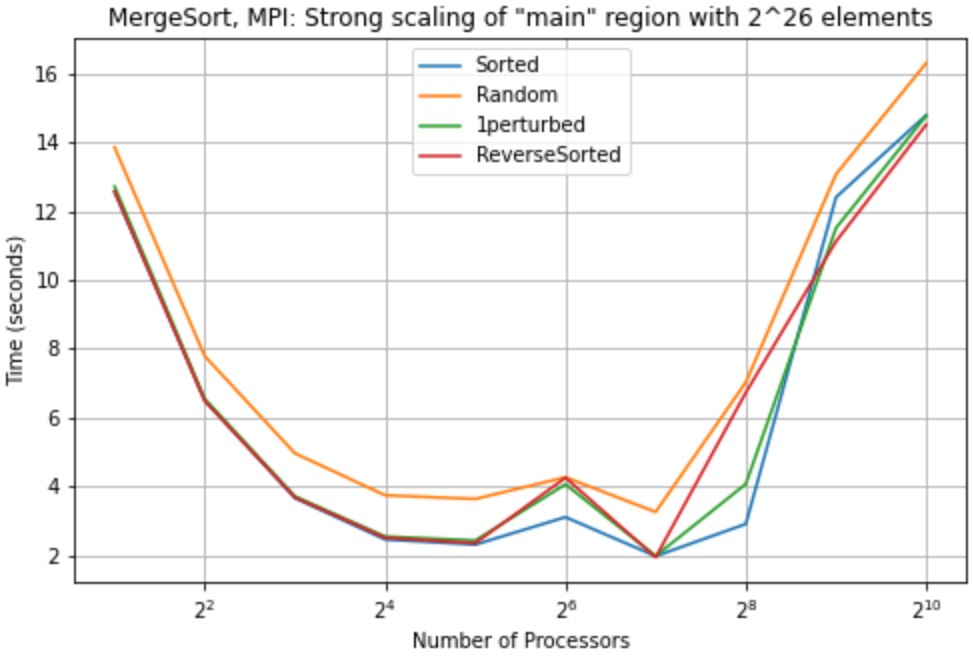

MergeSort, MPI: Strong scaling of "main" region with 2^16 elements
Sorted
Random
1perturbed
ReverseSorted
Time (seconds)
3.0
2.5
2.0
1.5
1.0
0.5
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

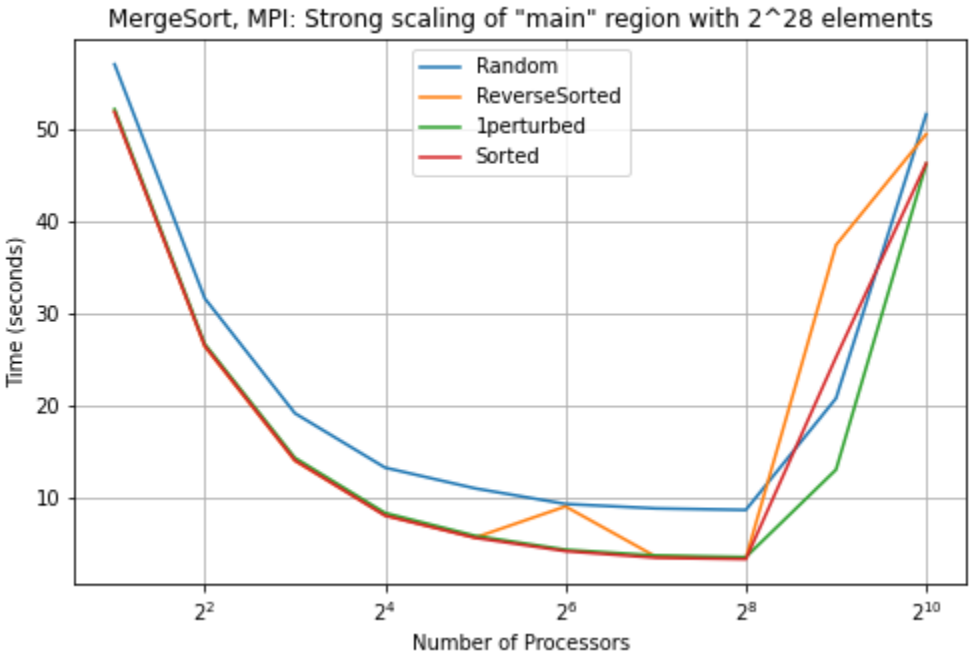

MergeSort, MPI: Strong scaling of "main" region with 2^18 elements
1perturbed
Sorted
Random
ReverseSorted
Time (seconds)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

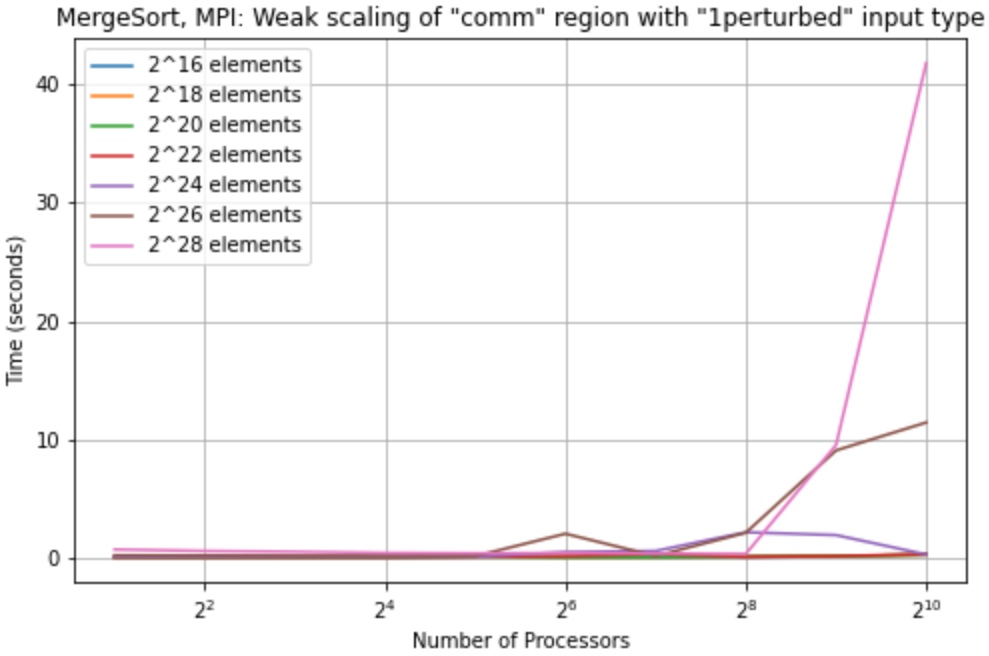

MergeSort, MPI: Strong scaling of "main" region with 2^20 elements
1perturbed
ReverseSorted
Sorted
Random
Time (seconds)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

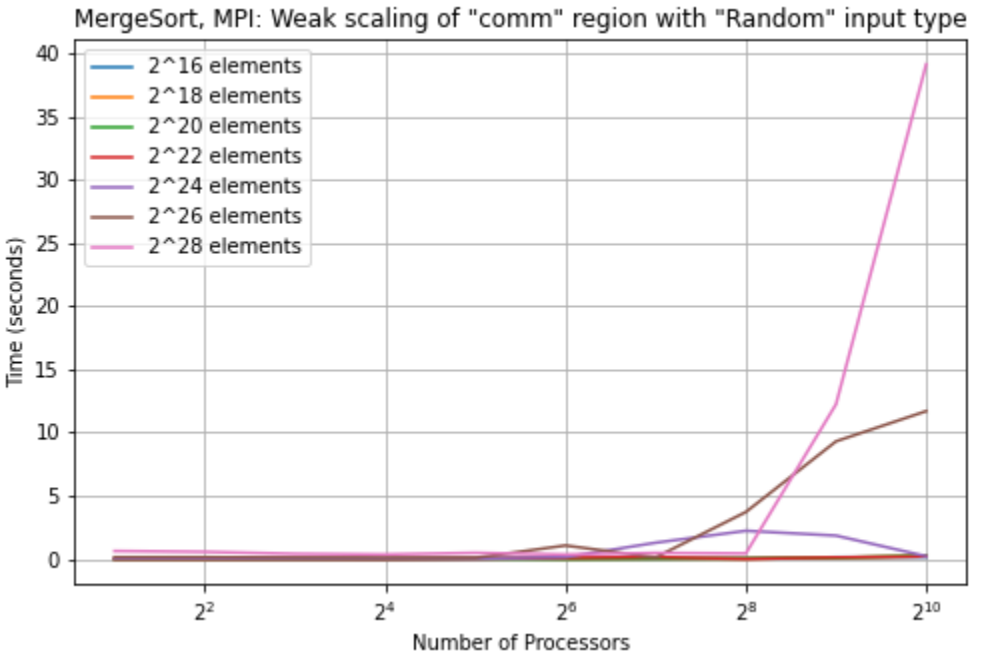

MergeSort, MPI: Strong scaling of "main" region with 2^22 elements
Sorted
ReverseSorted
Random
1perturbed
Time (seconds)
3.5
3.0
2.5
2.0
1.5
1.0
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

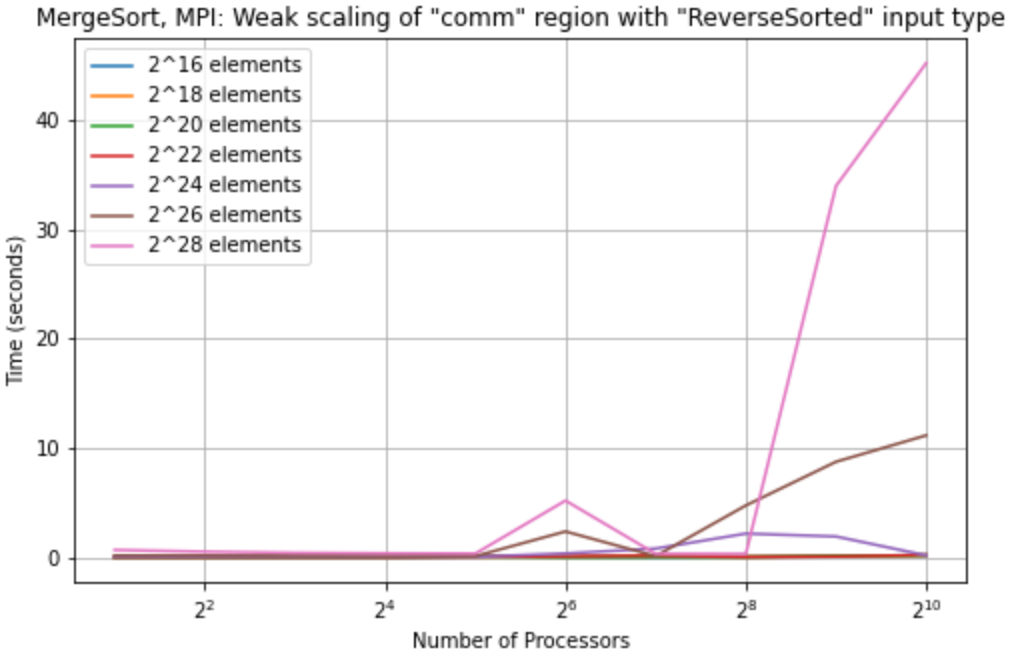

MergeSort, MPI: Strong scaling of "main" region with 2^24 elements
Time (seconds)
4.5
4.0
3.5
3.0
2.5
2.0
1.5
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors
ReverseSorted
Sorted
1perturbed
Random

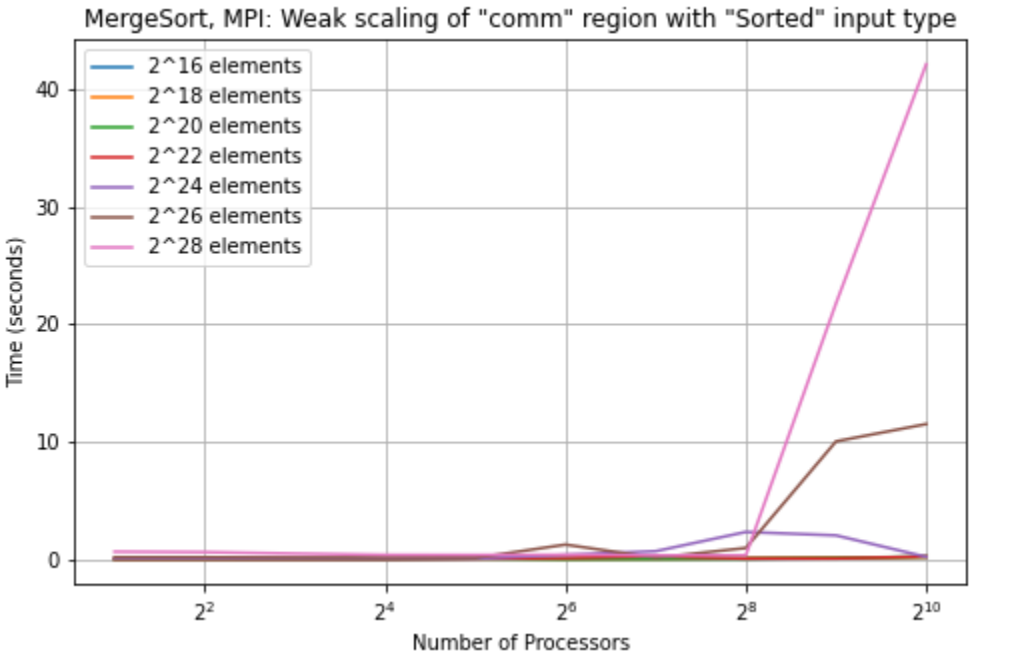

MergeSort, MPI: Strong scaling of "main" region with 2^26 elements
Sorted
Random
1perturbed
ReverseSorted
Time (seconds)
Number of Processors

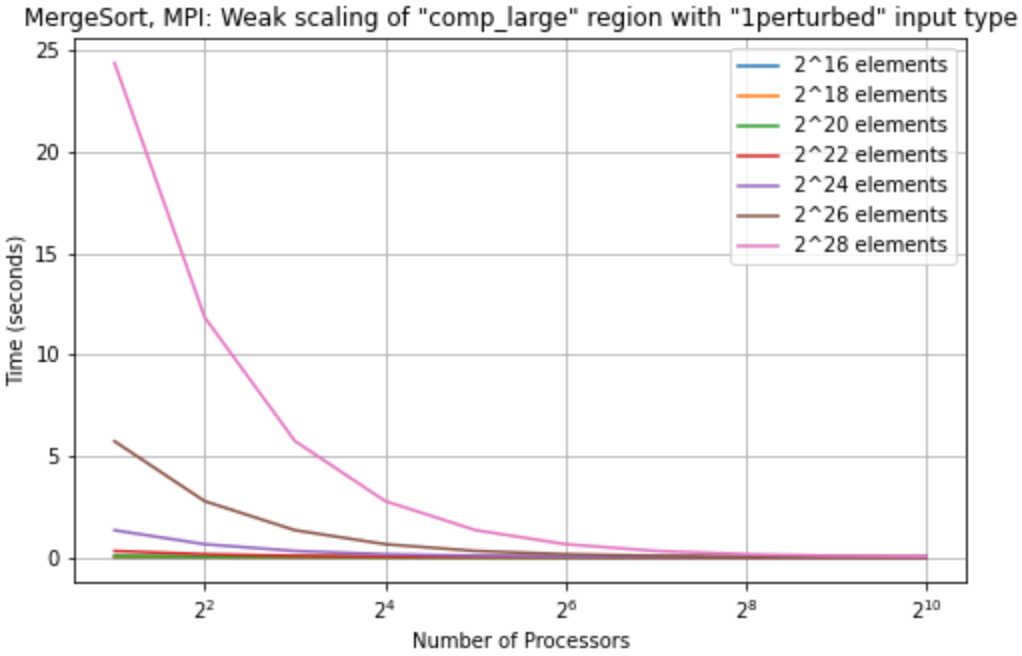

MergeSort, MPI: Strong scaling of "main" region with 2^28 elements
Random
ReverseSorted
1perturbed
Sorted
Time (seconds)
Number of Processors

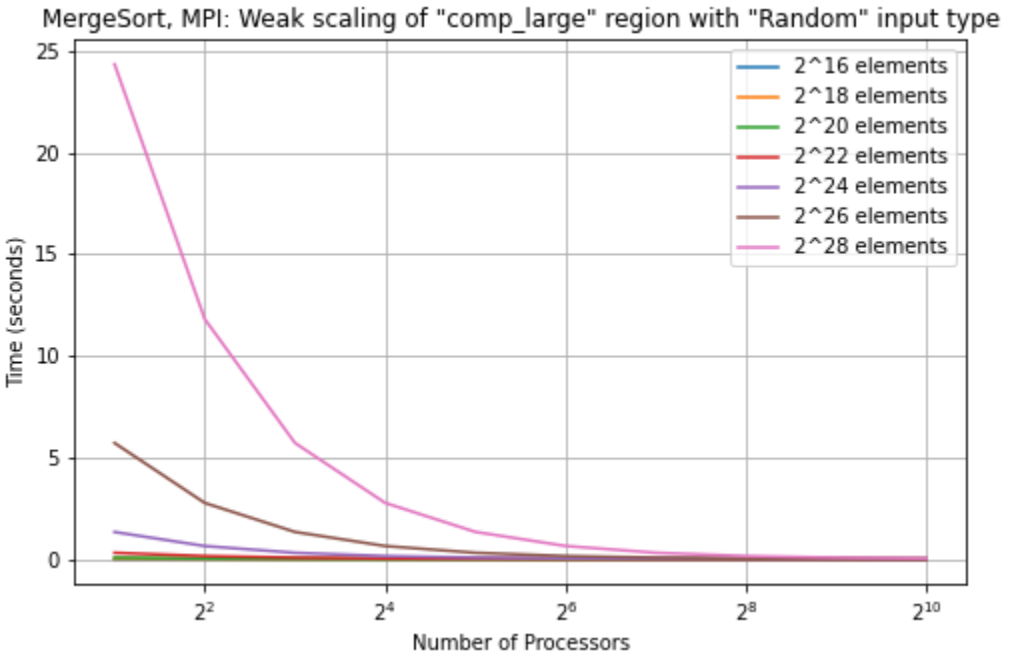

MergeSort, MPI: Weak scaling of "comm" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0
10
20
30
40
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

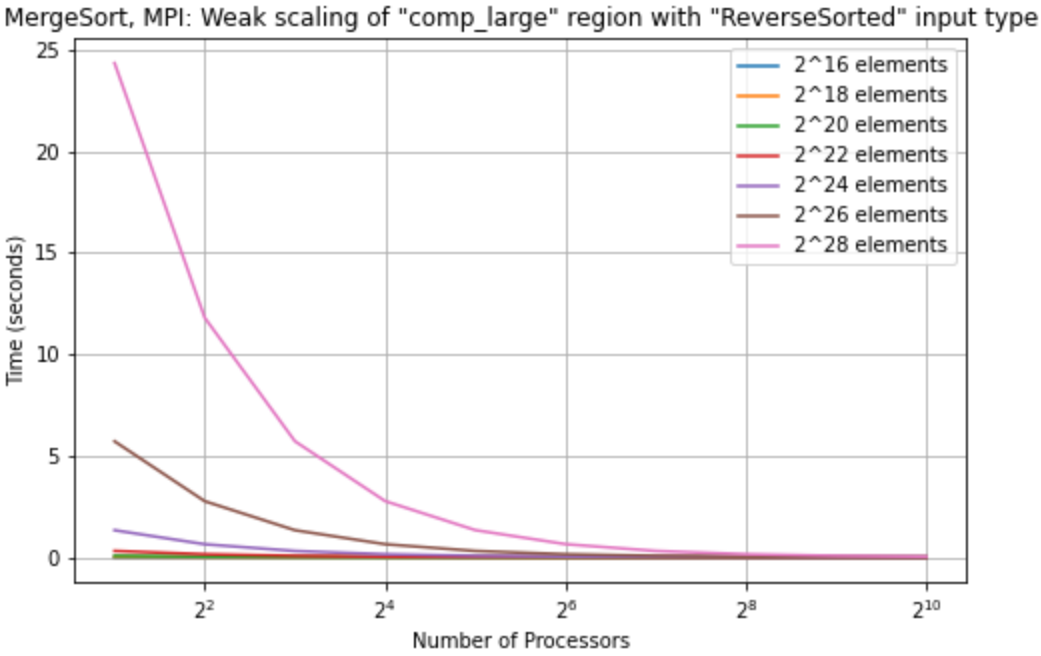

MergeSort, MPI: Weak scaling of "comm" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0
5
10
15
20
25
30
35
40
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

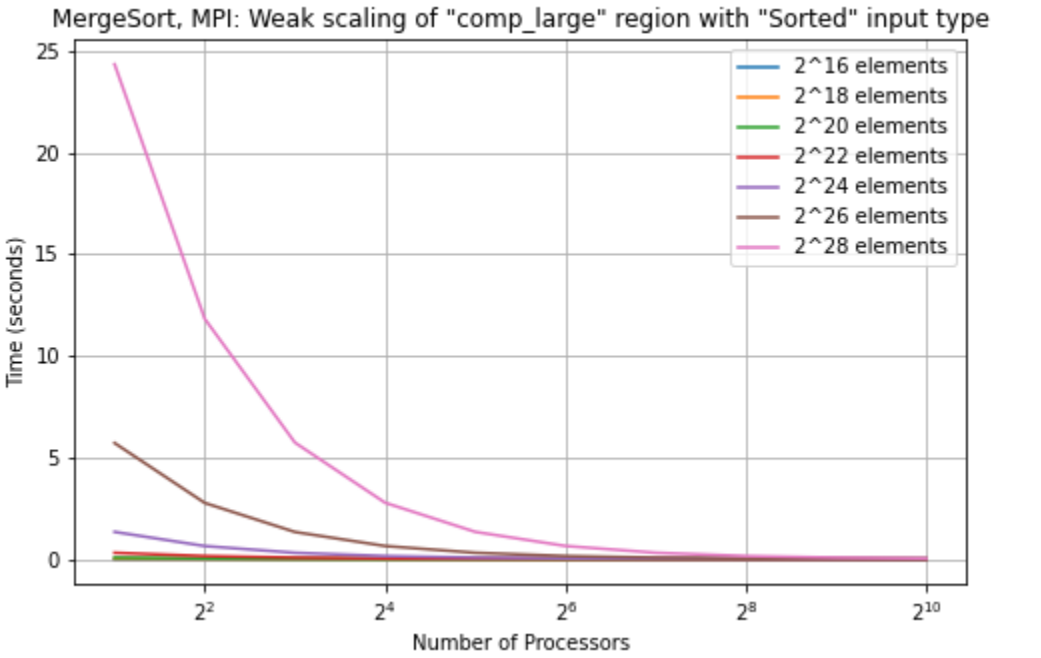

MergeSort, MPI: Weak scaling of "comm" region with "ReverseSorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0
10
20
30
40
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

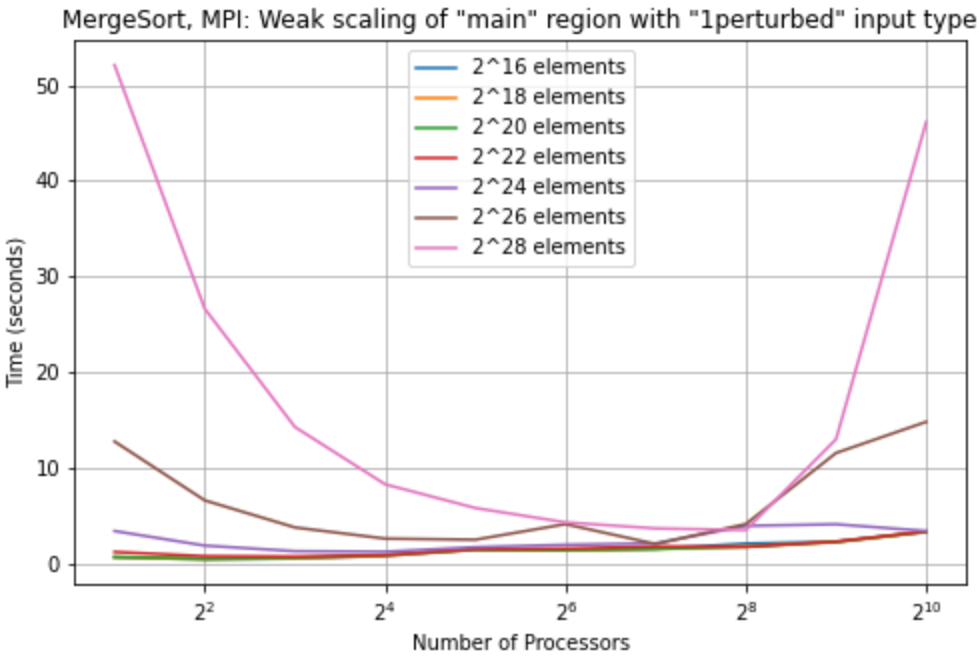

MergeSort, MPI: Weak scaling of "comm" region with "Sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0
10
20
30
40
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

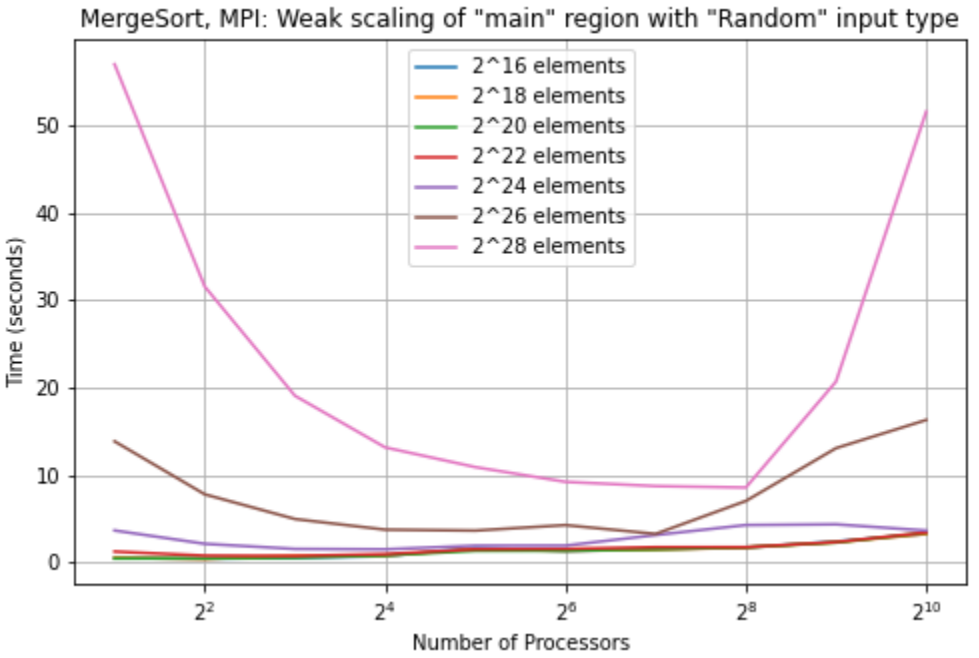

MergeSort, MPI: Weak scaling of "comp_large" region with "1perturbed" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

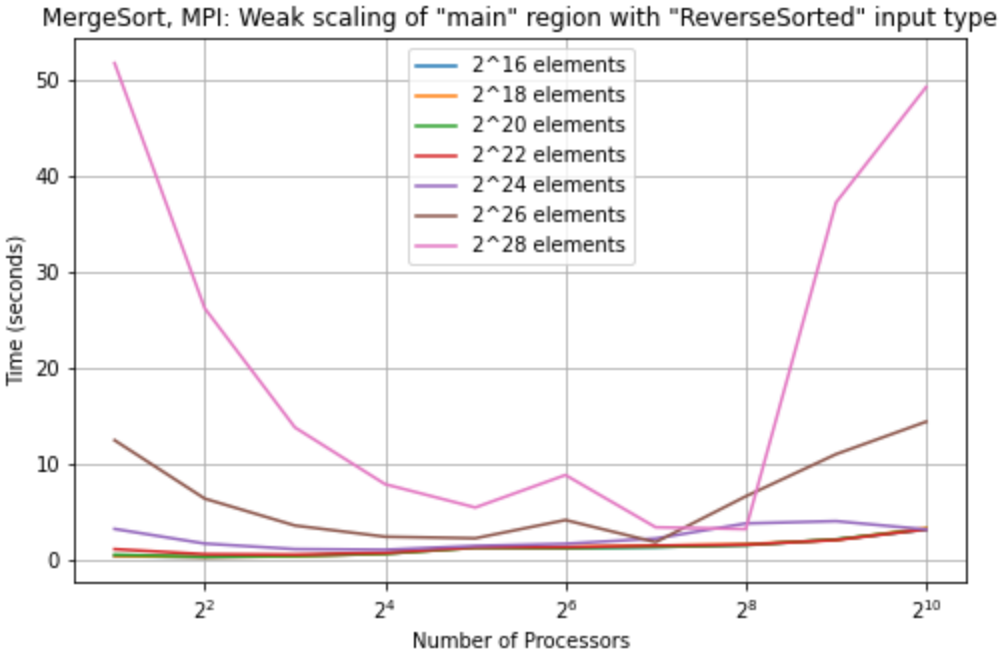

MergeSort, MPI: Weak scaling of "comp_large" region with "Random" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

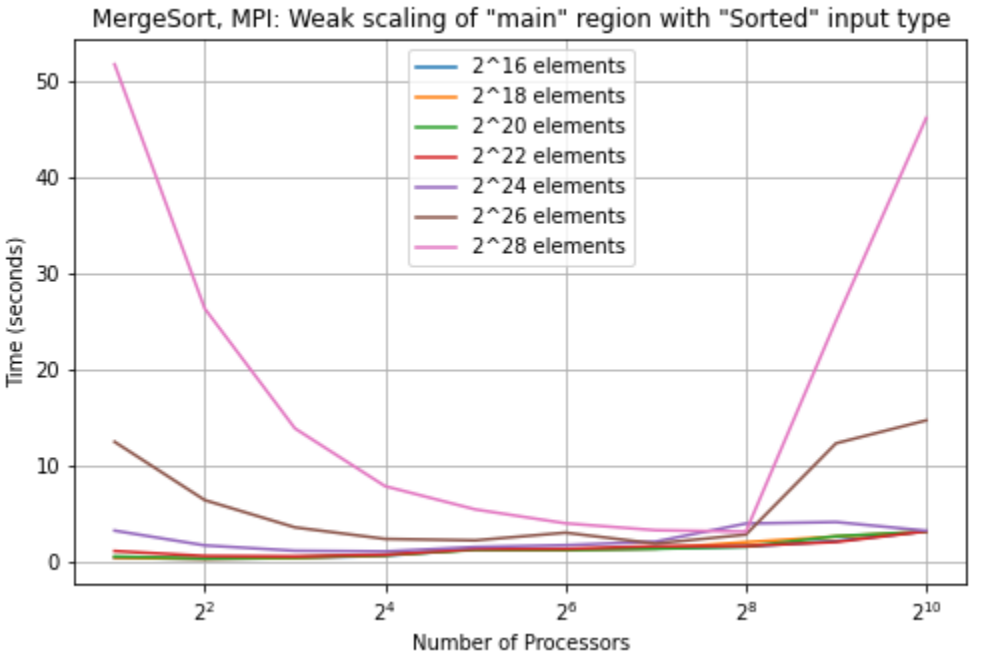

MergeSort, MPI: Weak scaling of "comp_large" region with "ReverseSorted" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

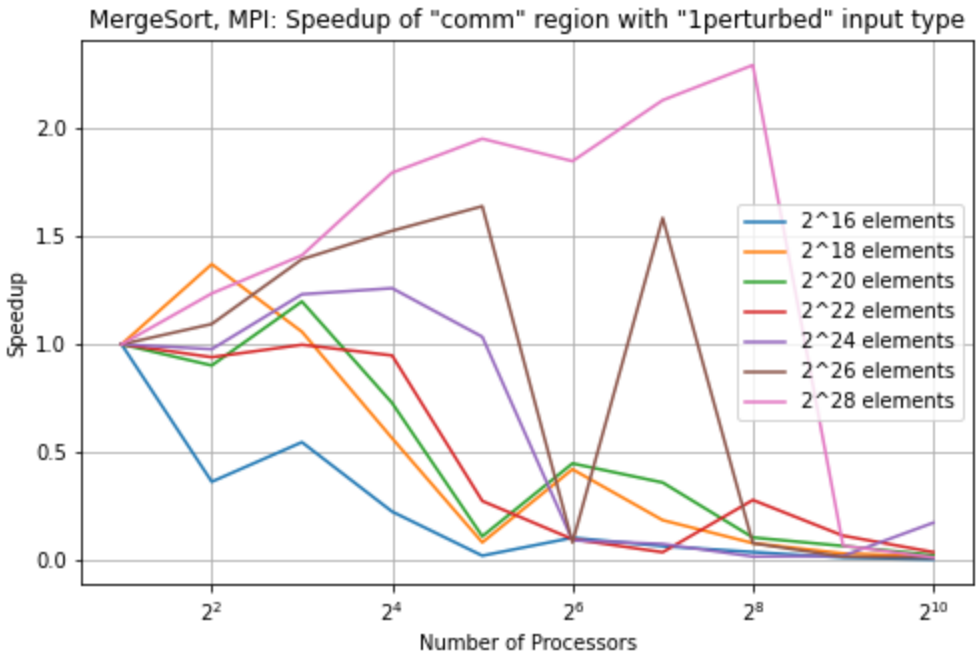

MergeSort, MPI: Weak scaling of "comp_large" region with "Sorted" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

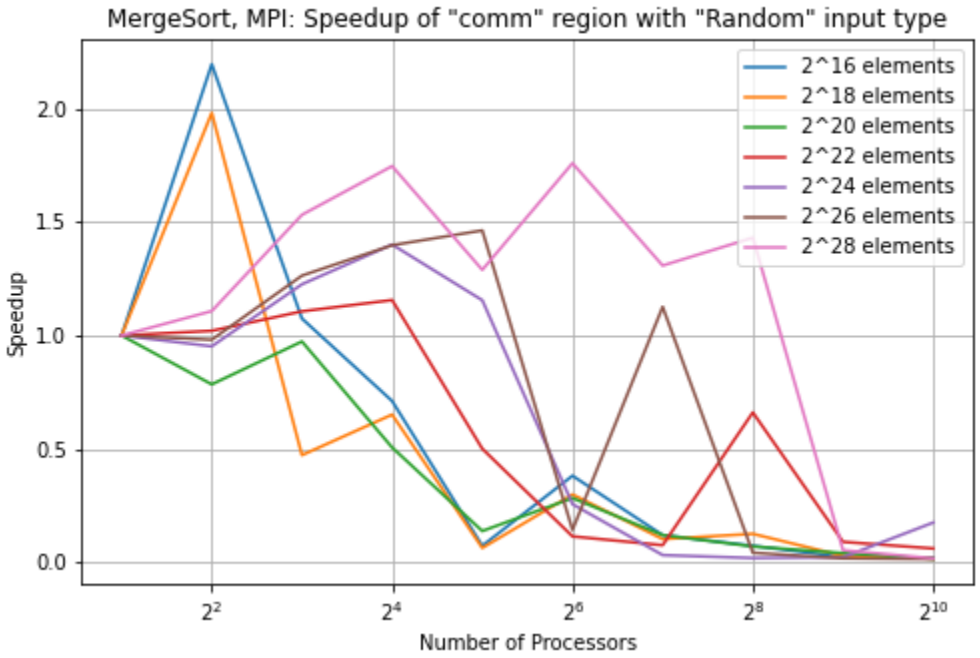

MergeSort, MPI: Weak scaling of "main" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0
10
20
30
40
50
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

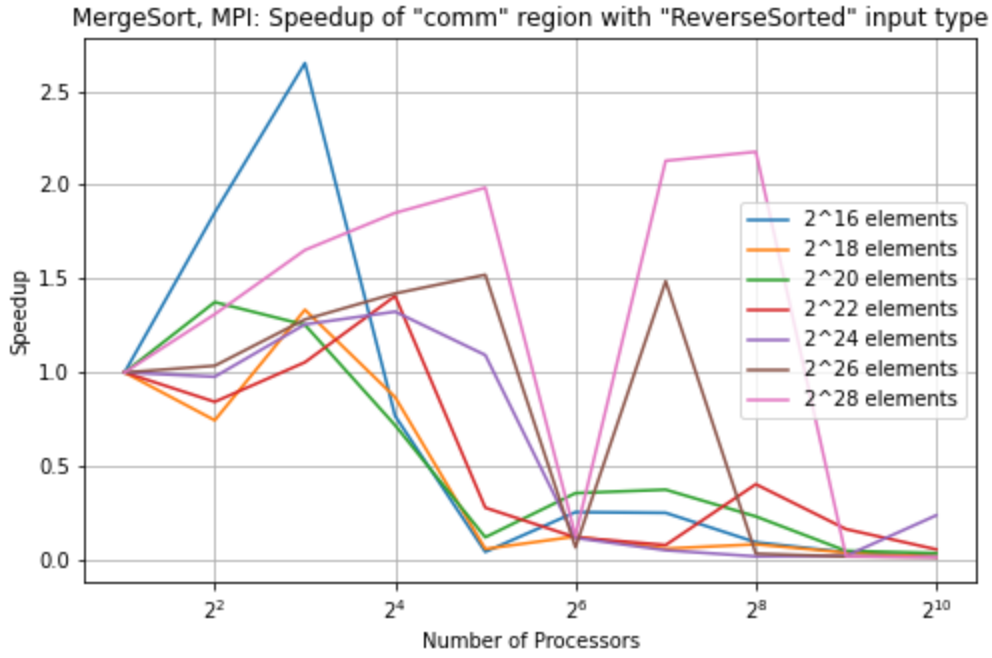

MergeSort, MPI: Weak scaling of "main" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0
10
20
30
40
50
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

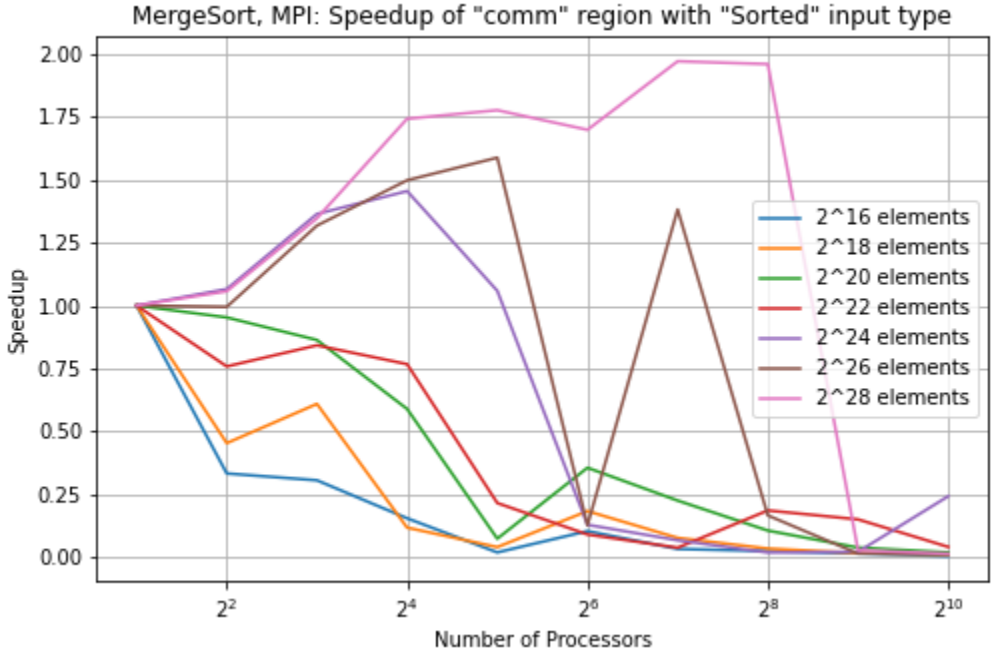

MergeSort, MPI: Weak scaling of "main" region with "ReverseSorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0
10
20
30
40
50
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

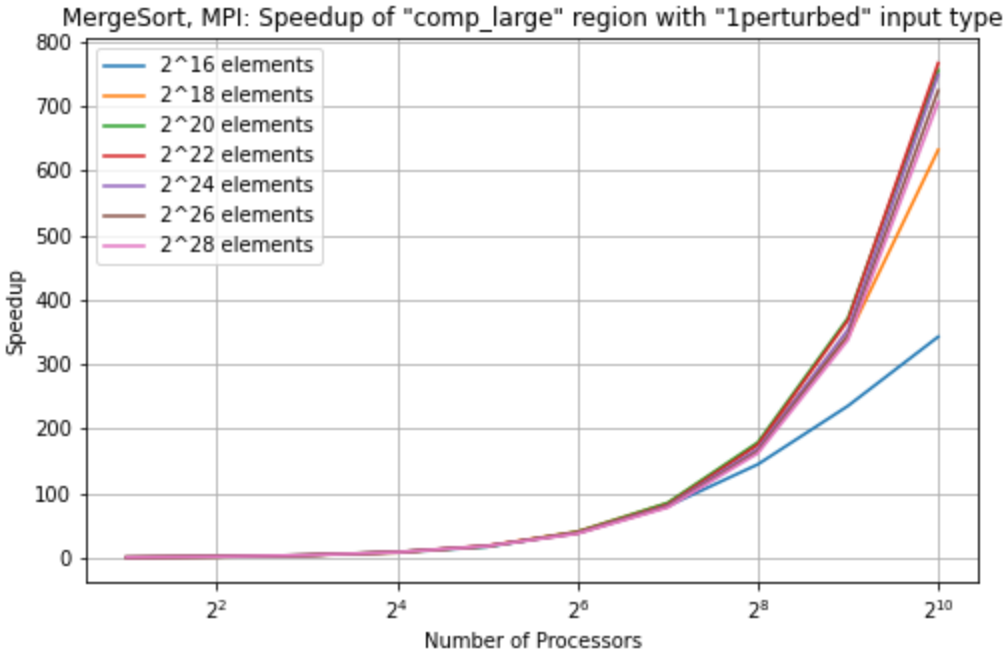

MergeSort, MPI: Weak scaling of "main" region with "Sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0
10
20
30
40
50
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

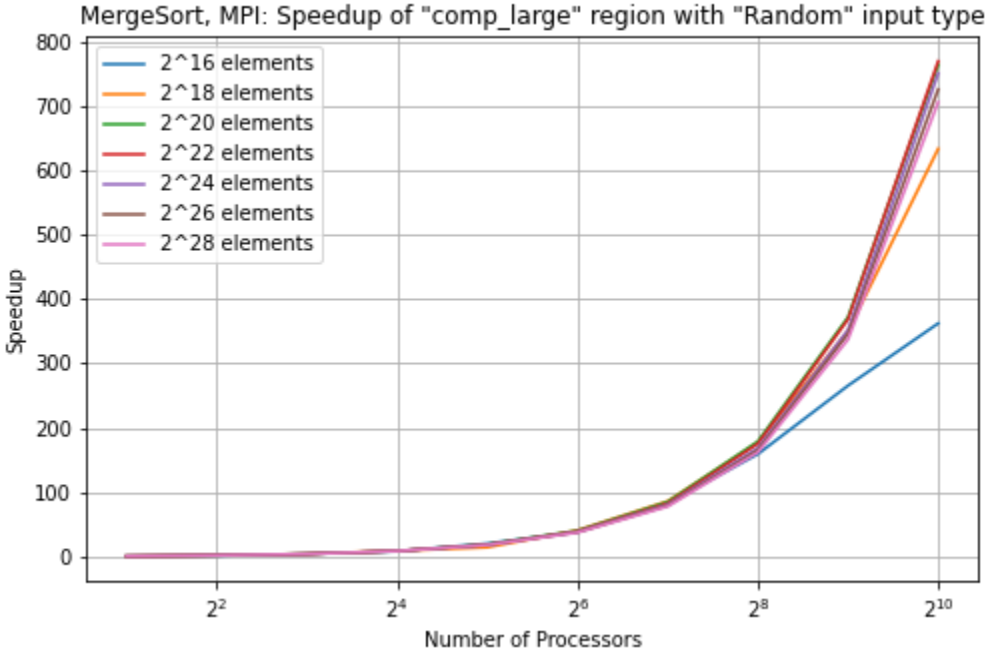

MergeSort, MPI: Speedup of "comm" region with "1perturbed" input type
Speedup
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

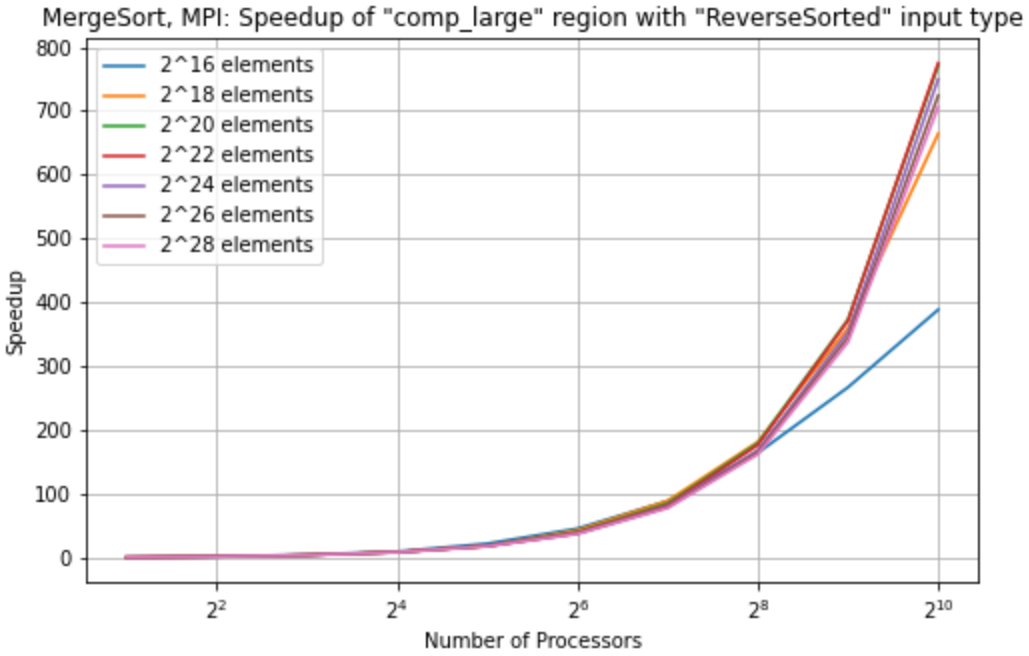

MergeSort, MPI: Speedup of "comm" region with "Random" input type
Speedup
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

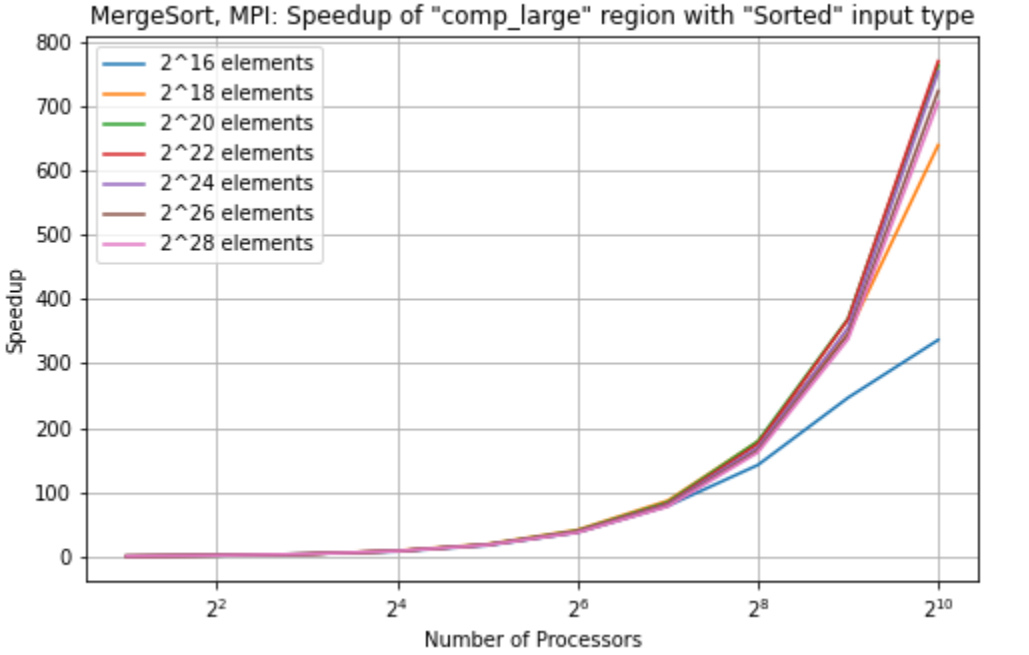

MergeSort, MPI: Speedup of "comm" region with "ReverseSorted" input type
Speedup
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

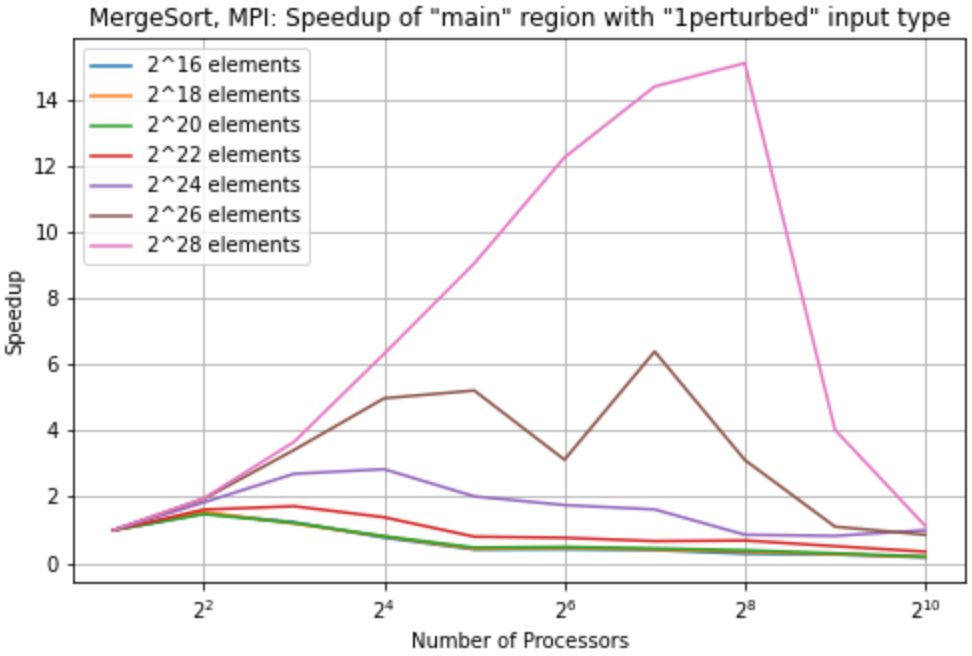

MergeSort, MPI: Speedup of "comm" region with "Sorted" input type
Speedup
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

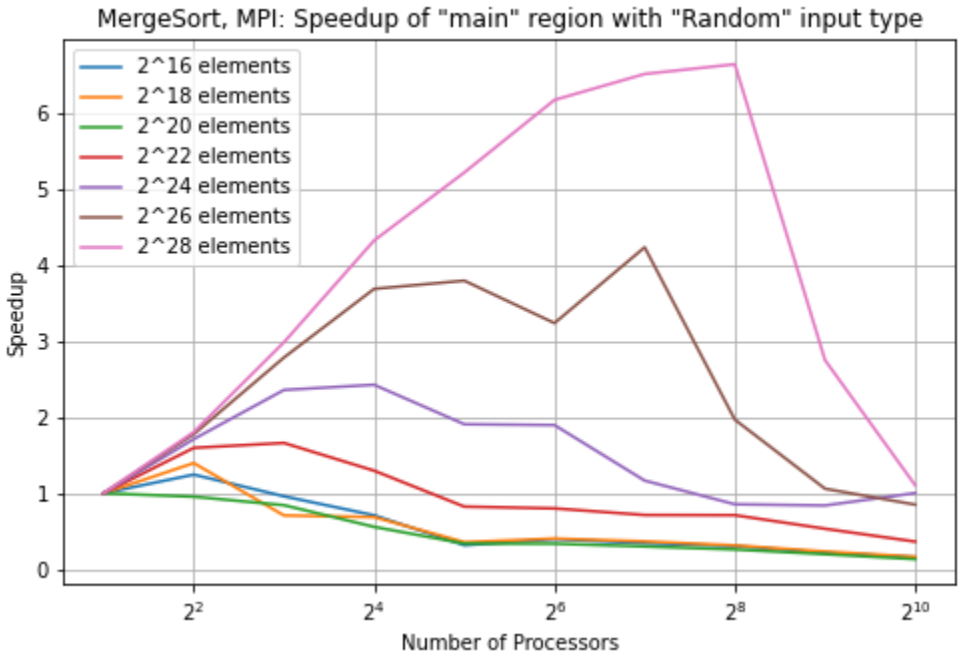

MergeSort, MPI: Speedup of "comp_large" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors

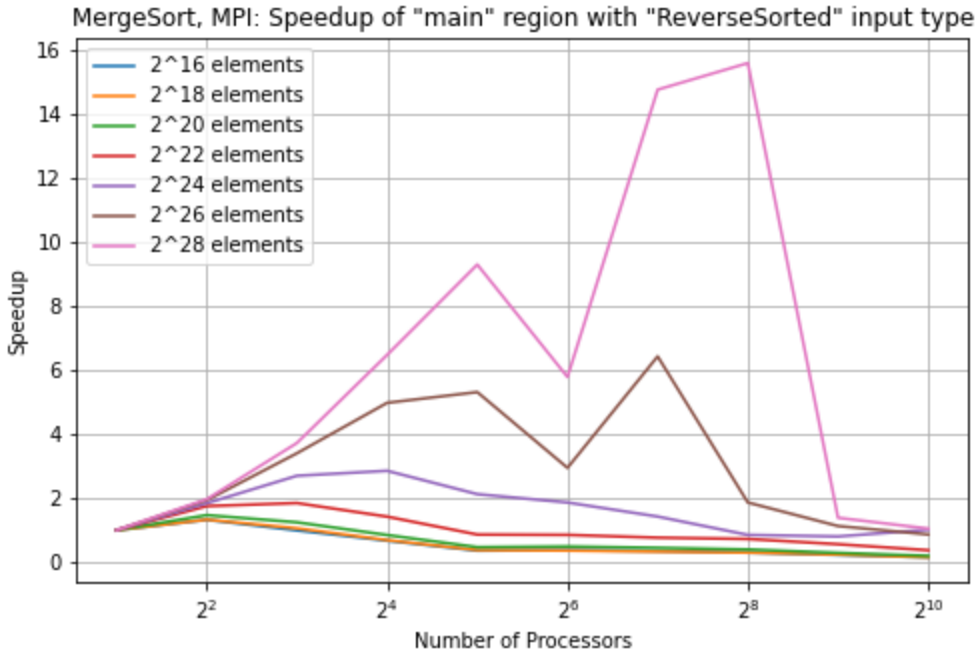

MergeSort, MPI: Speedup of "comp_large" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors

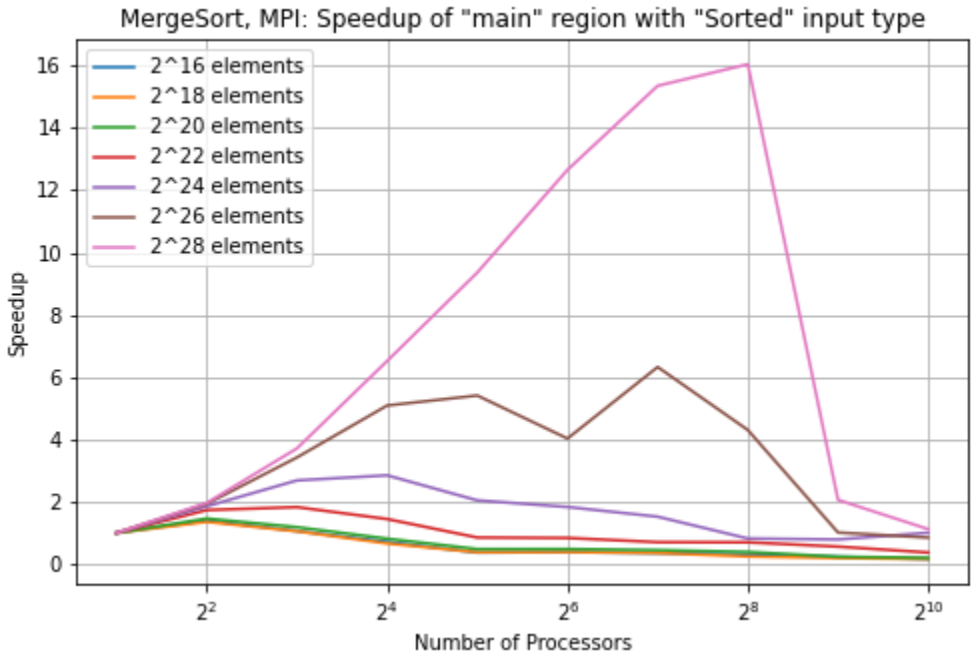

MergeSort, MPI: Speedup of "comp_large" region with "ReverseSorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors


MergeSort, MPI: Speedup of "comp_large" region with "Sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors


MergeSort, MPI: Speedup of "main" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0
2
4
6
8
10
12
14
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors


MergeSort, MPI: Speedup of "main" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0
1
2
3
4
5
6
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors



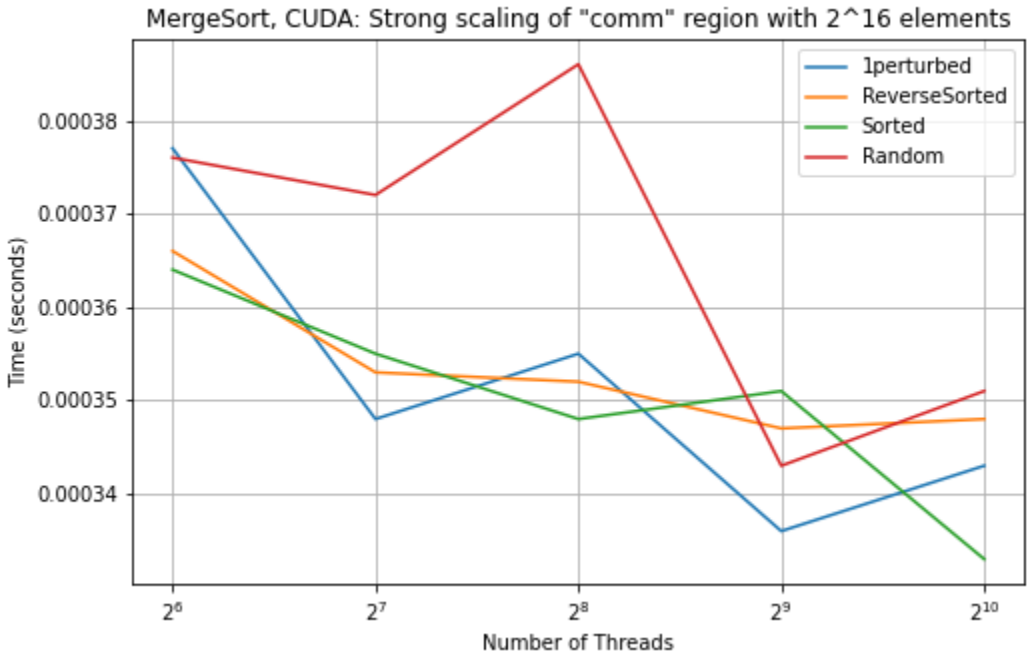



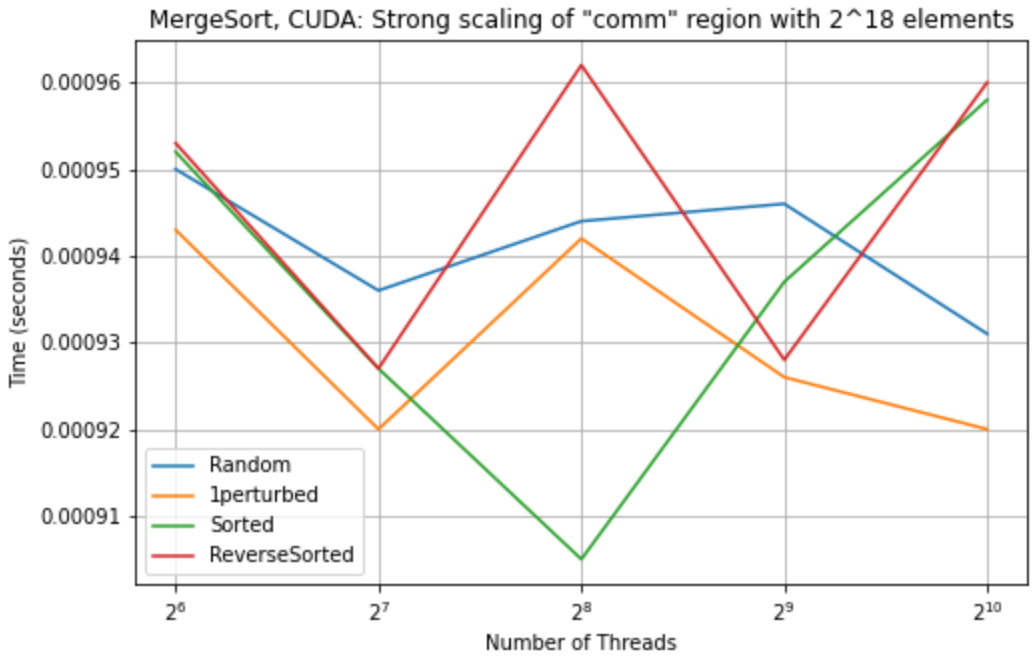

One would expect that as the number of processors increases, the runtime of the mergesort algorithm runtime decreases up until a point of diminishing returns. This is because the communication time will increase, with more workers splitting up the overall task. This holds true, and it is clear that Computation time and communication time both contribute to the overall time as the number of processors increases. It seems that, in

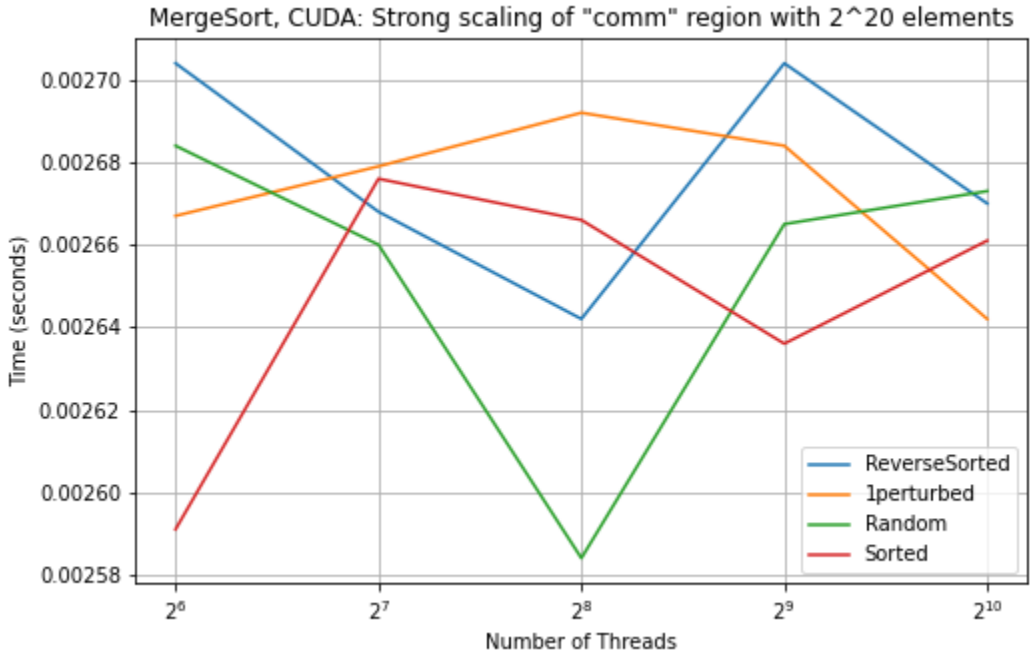
terms of communication, 2^8 processors is the ideal efficiency of this algorithm implementation at the large array input sizes, as after that, communication time drastically increases and makes the overall algorithm slower, regardless of the gains in computation time. Computation time itself has a very clear logarithmic trend of reducing time as processors increase.The computation time has a very large speedup at maximum processors, while communication speedup varies greatly. Overall, the algorithm's main function reaches a relatively small speedup of around 16 at largest input sizes, and less than 1 at smaller input sizes and high processors, which could signal some issues with the implementation of the MPI mergesort being distributed among the workers.

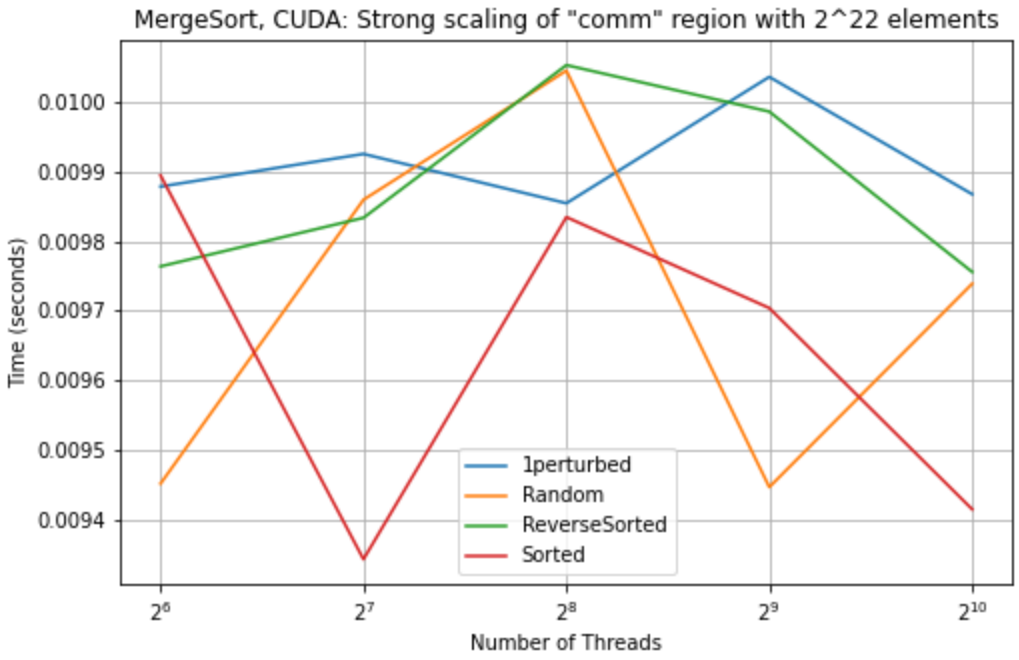
# Merge Sort CUDA

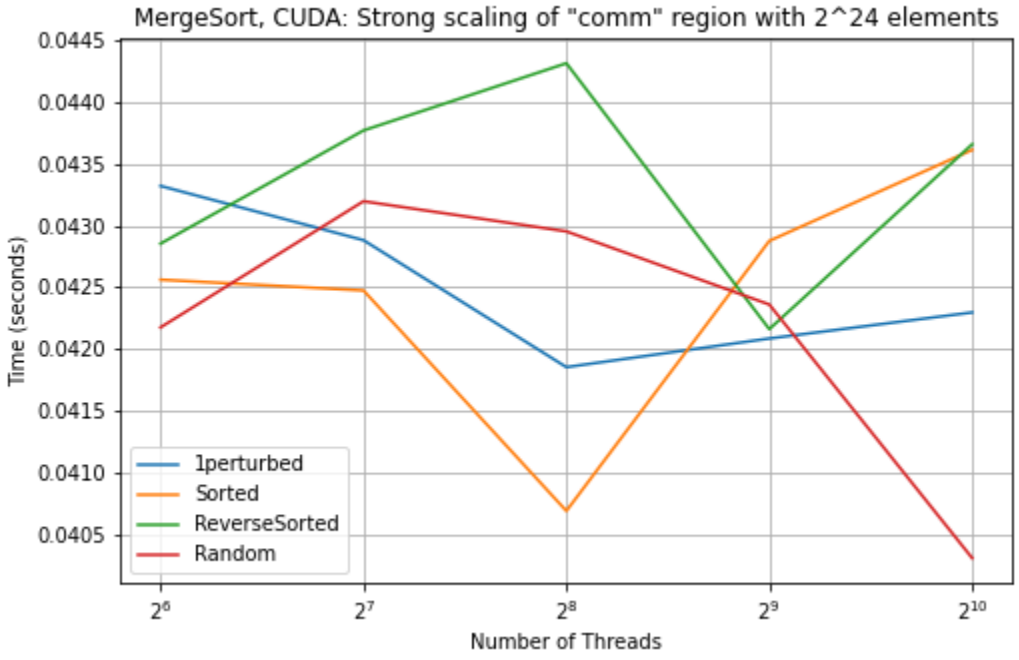
**Graphs**

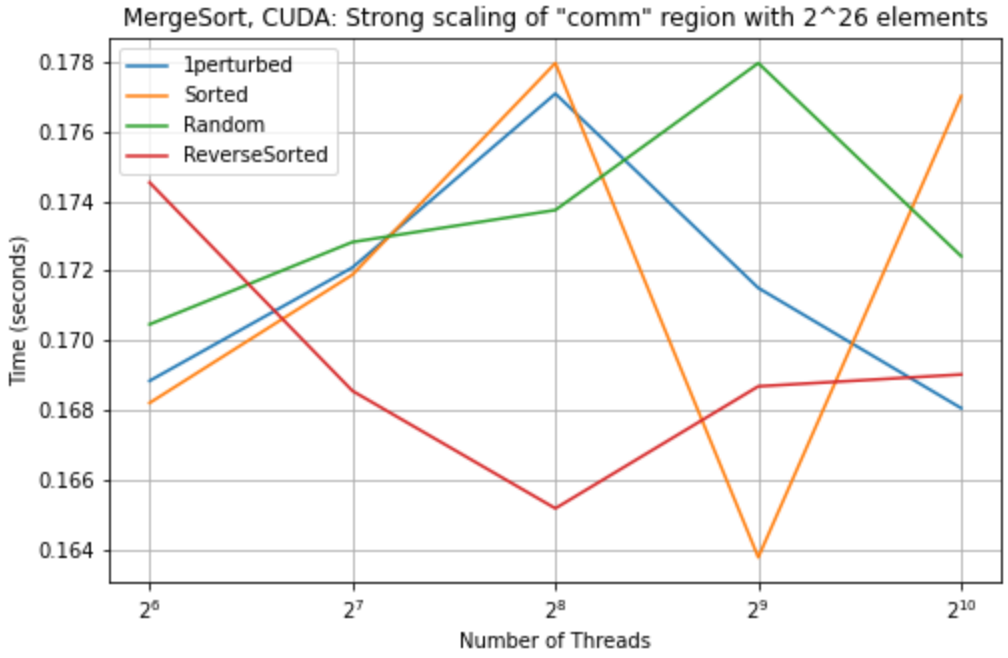


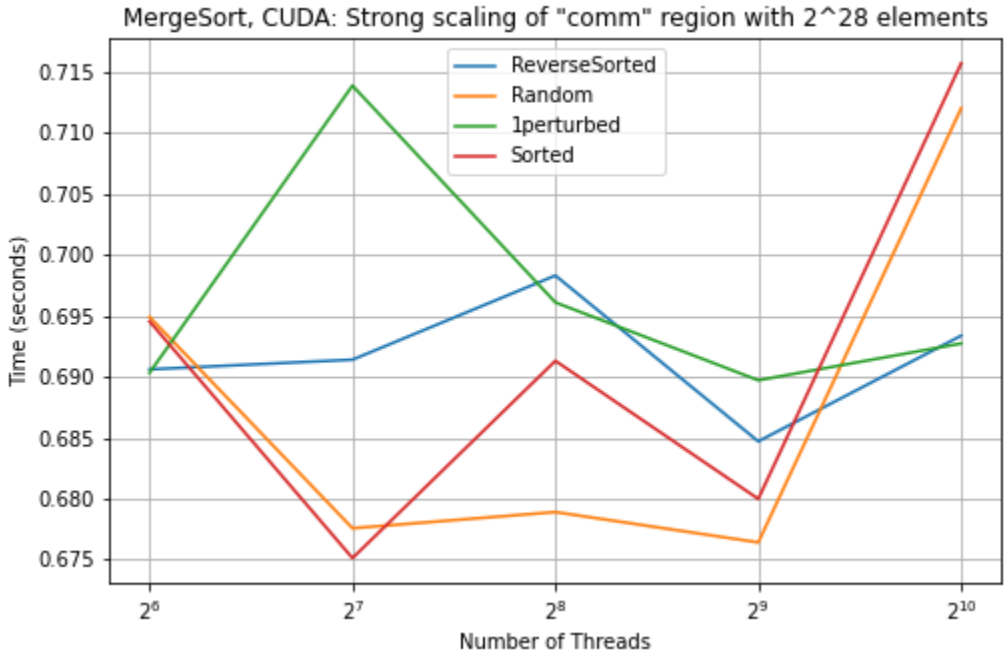

MergeSort, CUDA: Strong scaling of "comm" region with 2^18 elements
Time (seconds)
0.00096
0.00095
0.00094
0.00093
0.00092
0.00091
Random
1perturbed
Sorted
ReverseSorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

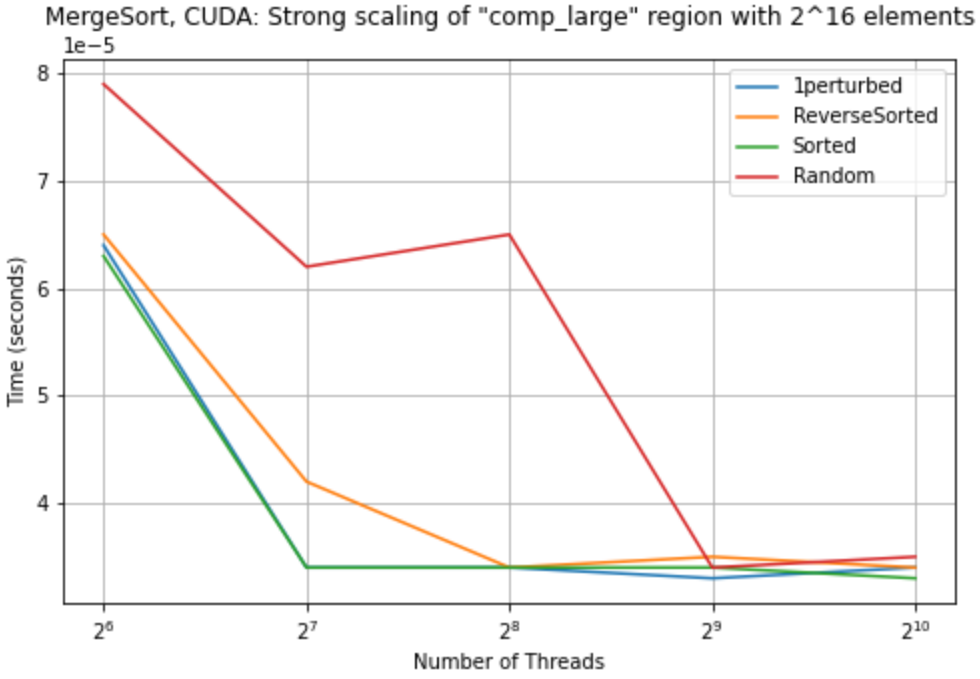

MergeSort, CUDA: Strong scaling of "comm" region with 2^20 elements
Time (seconds)
0.00270
0.00268
0.00266
0.00264
0.00262
0.00260
0.00258
ReverseSorted
1perturbed
Random
Sorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

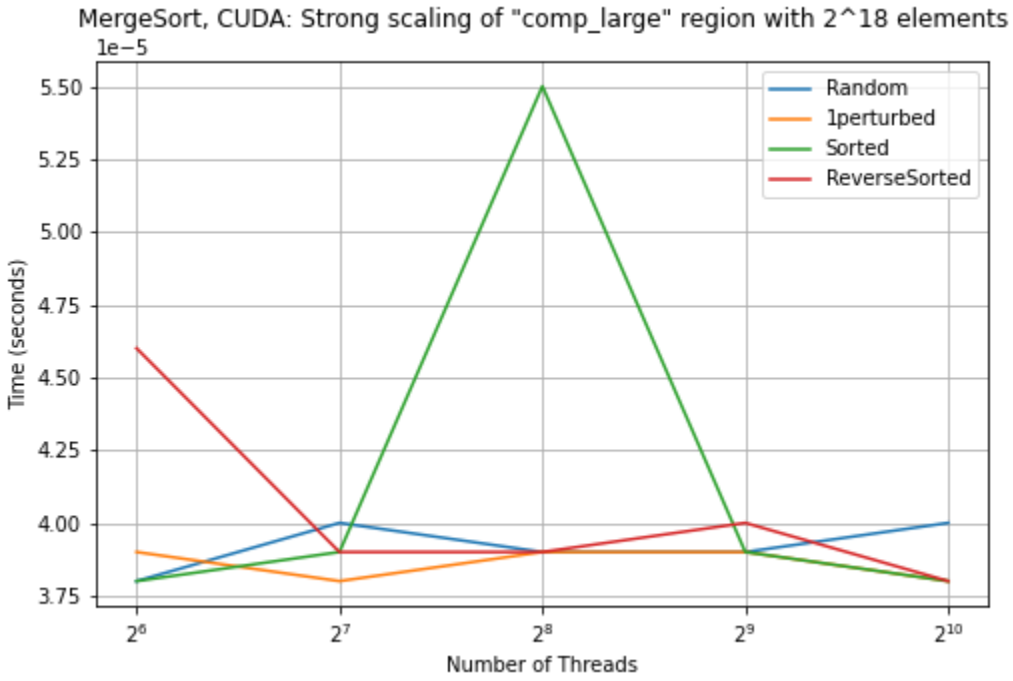

MergeSort, CUDA: Strong scaling of "comm" region with 2^22 elements
Time (seconds)
0.0100
0.0099
0.0098
0.0097
0.0096
0.0095
0.0094
1perturbed
Random
ReverseSorted
Sorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

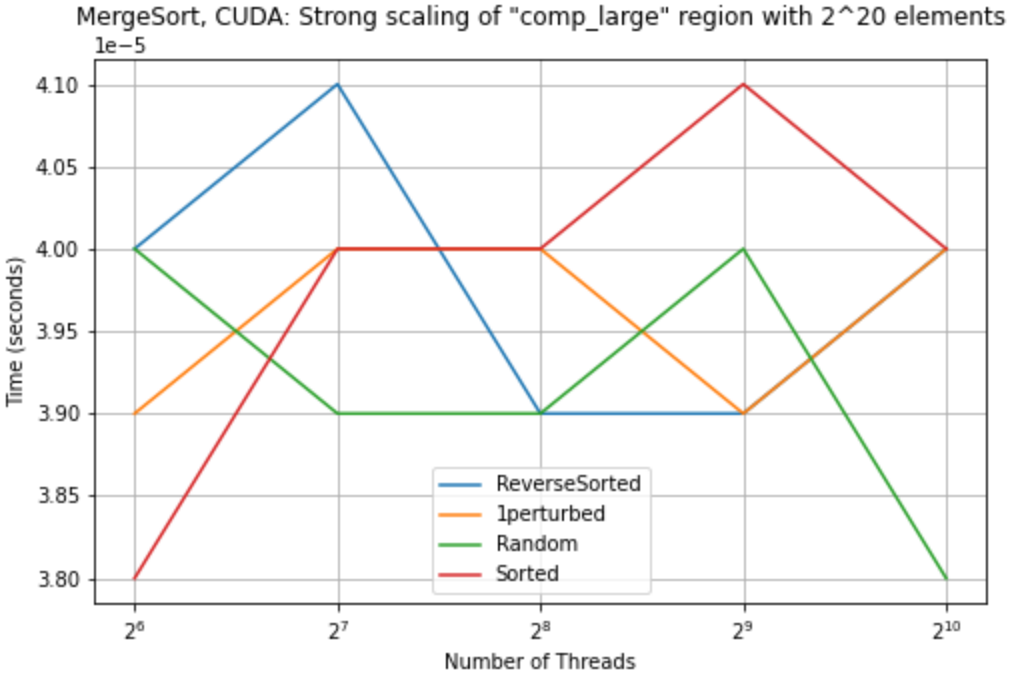

MergeSort, CUDA: Strong scaling of "comm" region with 2^24 elements
Time (seconds)
0.0445
0.0440
0.0435
0.0430
0.0425
0.0420
0.0415
0.0410
0.0405
1perturbed
Sorted
ReverseSorted
Random
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

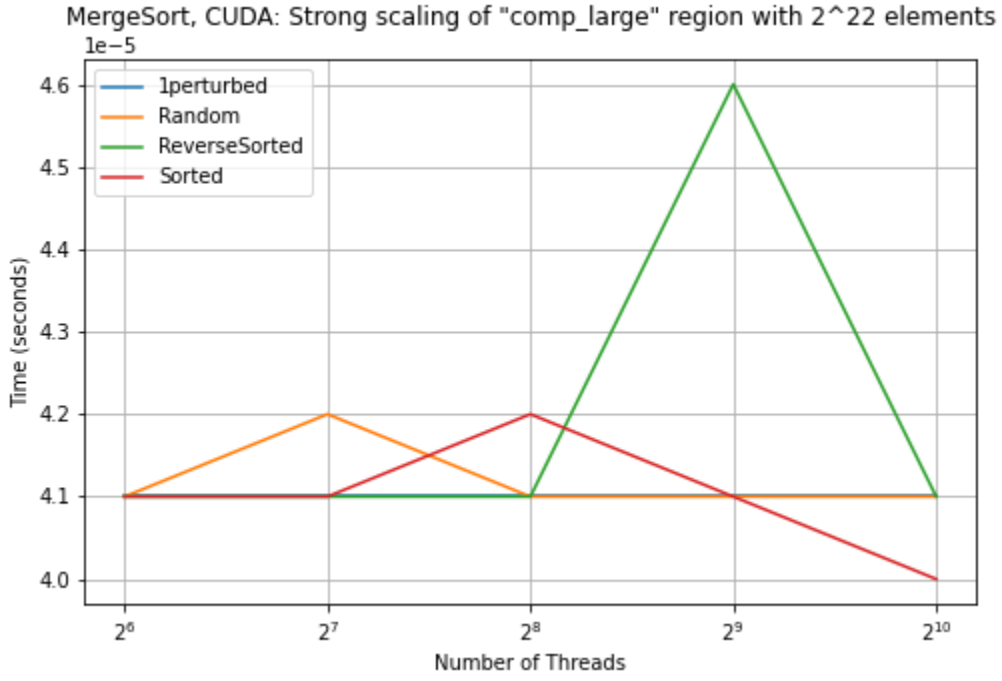

MergeSort, CUDA: Strong scaling of "comm" region with 2^26 elements
1perturbed
Sorted
Random
ReverseSorted
Time (seconds)
0.178
0.176
0.174
0.172
0.170
0.168
0.166
0.164
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

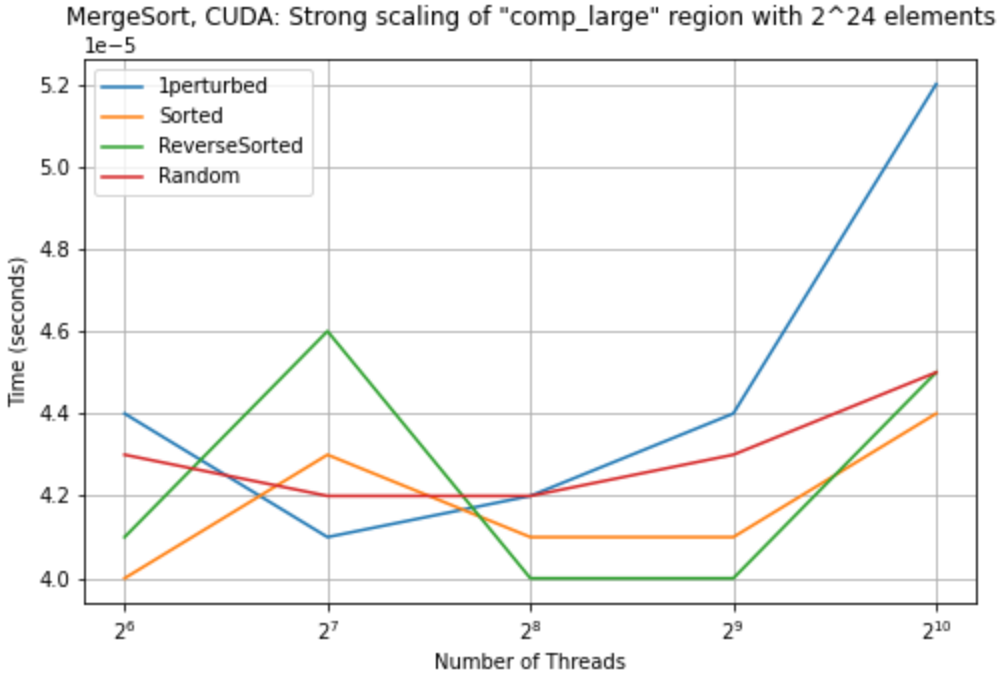

MergeSort, CUDA: Strong scaling of "comm" region with 2^28 elements
ReverseSorted
Random
1perturbed
Sorted
Time (seconds)
0.715
0.710
0.705
0.700
0.695
0.690
0.685
0.680
0.675
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

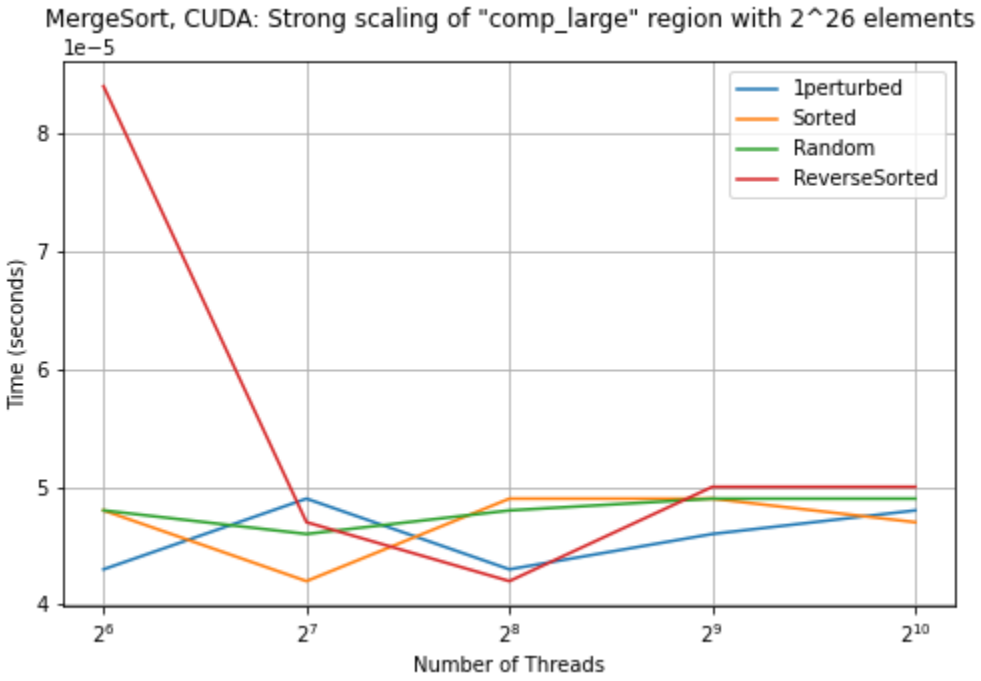

MergeSort, CUDA: Strong scaling of "comp_large" region with 2^16 elements
1e−5
8
7
6
5
4
Time (seconds)
1perturbed
ReverseSorted
Sorted
Random
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

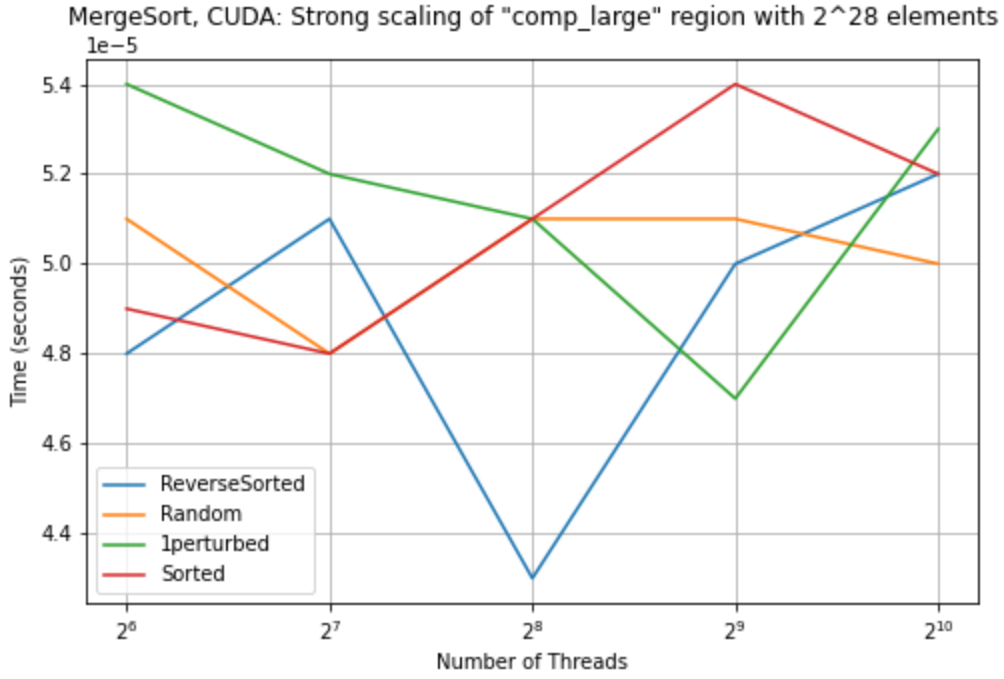

MergeSort, CUDA: Strong scaling of "comp_large" region with 2^18 elements
1e−5
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
Time (seconds)
Random
1perturbed
Sorted
ReverseSorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

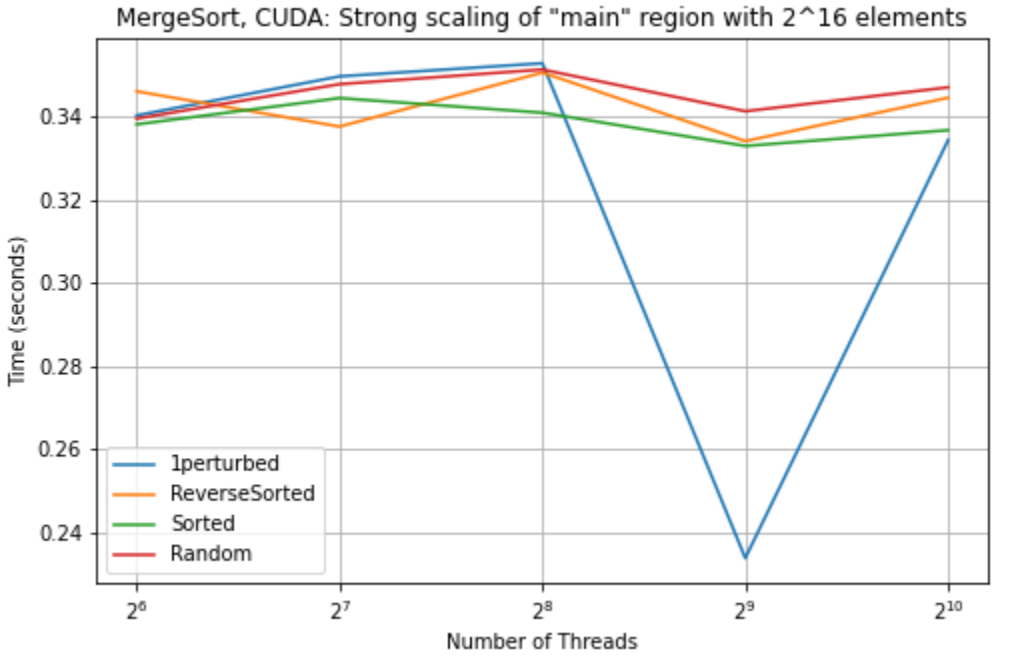

MergeSort, CUDA: Strong scaling of "comp_large" region with 2^20 elements
1e−5
4.10
4.05
4.00
3.95
3.90
3.85
3.80
Time (seconds)
ReverseSorted
1perturbed
Random
Sorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

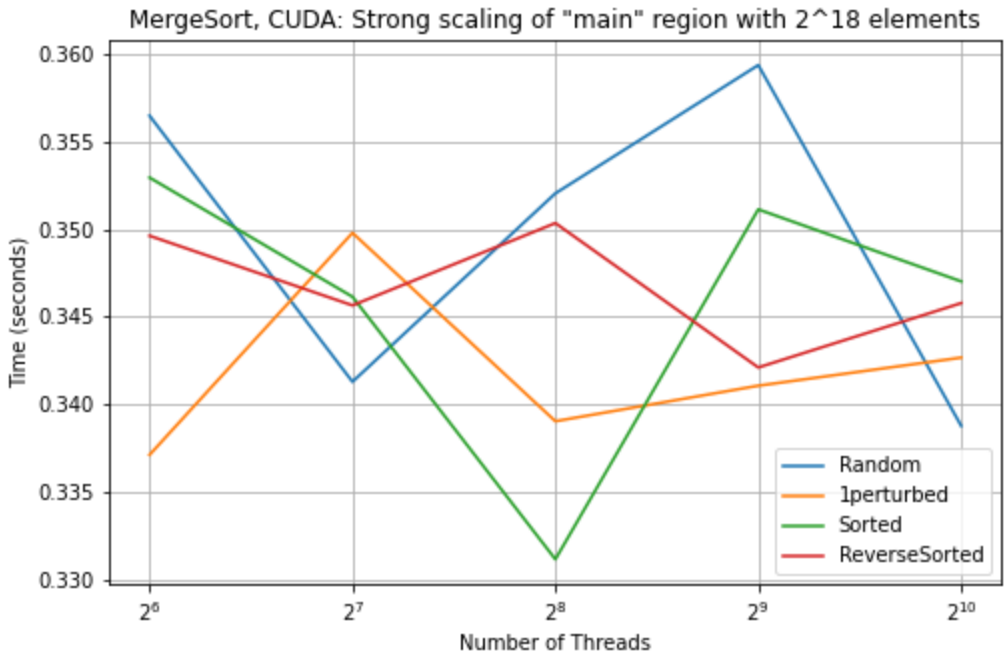

MergeSort, CUDA: Strong scaling of "comp_large" region with 2^22 elements
1e−5
4.6
4.5
4.4
4.3
4.2
4.1
4.0
Time (seconds)
1perturbed
Random
ReverseSorted
Sorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

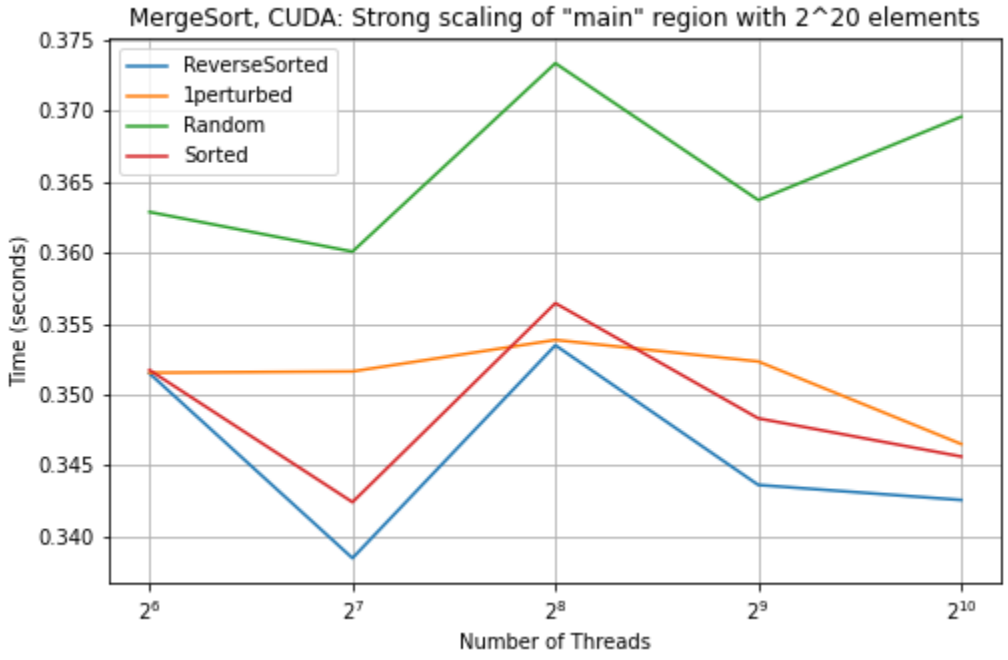

MergeSort, CUDA: Strong scaling of "comp_large" region with 2^24 elements
1e−5
1perturbed
Sorted
ReverseSorted
Random
Time (seconds)
Number of Threads

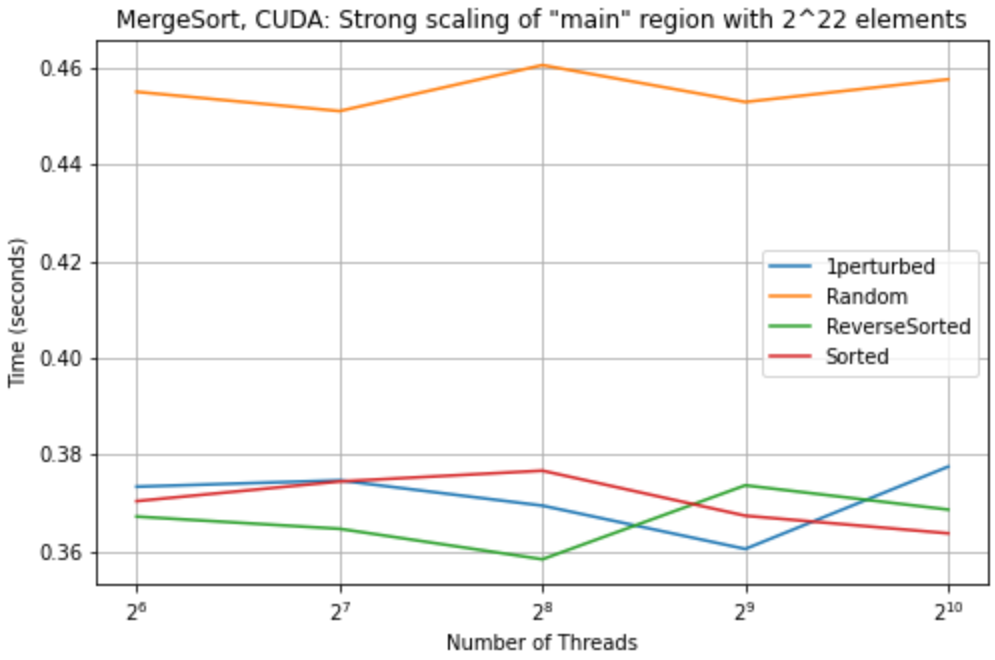

MergeSort, CUDA: Strong scaling of "comp_large" region with 2^26 elements
1e−5
1perturbed
Sorted
Random
ReverseSorted
Time (seconds)
Number of Threads

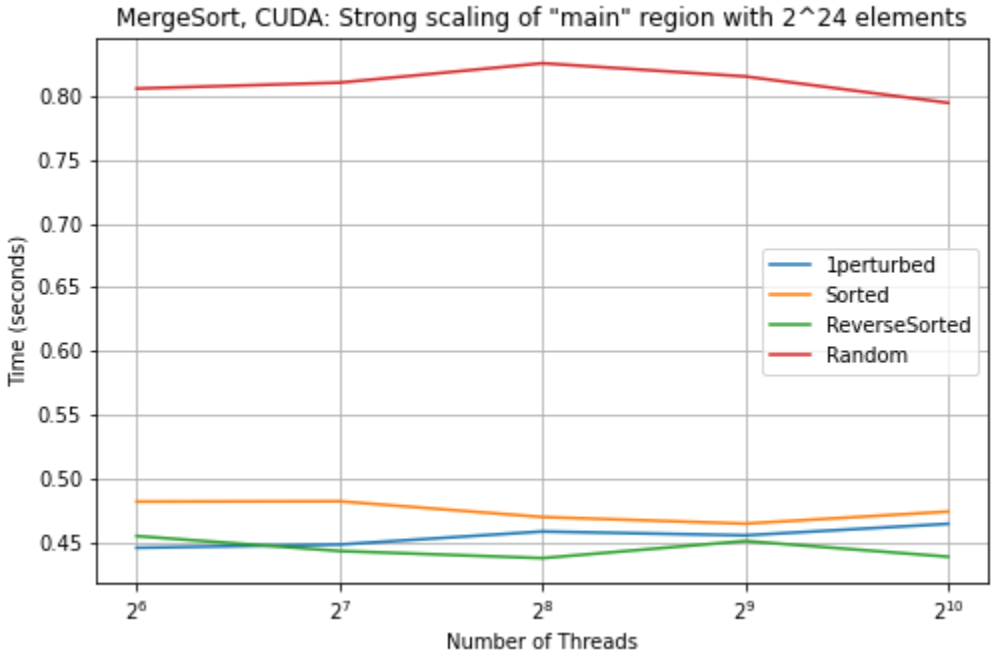

MergeSort, CUDA: Strong scaling of "comp_large" region with 2^28 elements
1e−5
5.4
5.2
5.0
4.8
4.6
4.4
Time (seconds)
ReverseSorted
Random
1perturbed
Sorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

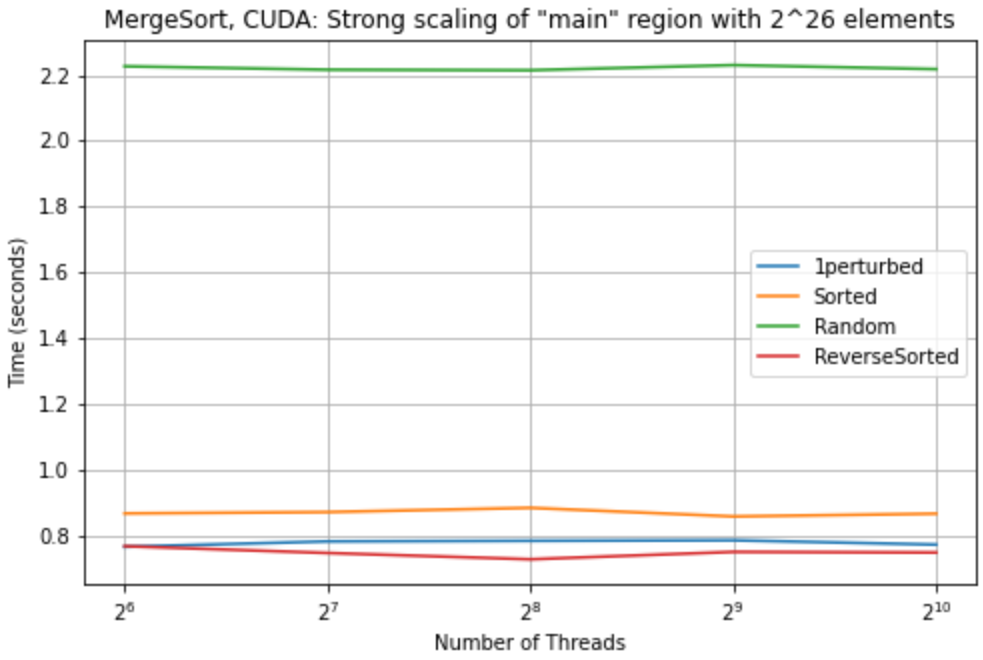

MergeSort, CUDA: Strong scaling of "main" region with 2^16 elements
0.34
0.32
0.30
0.28
0.26
0.24
Time (seconds)
1perturbed
ReverseSorted
Sorted
Random
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

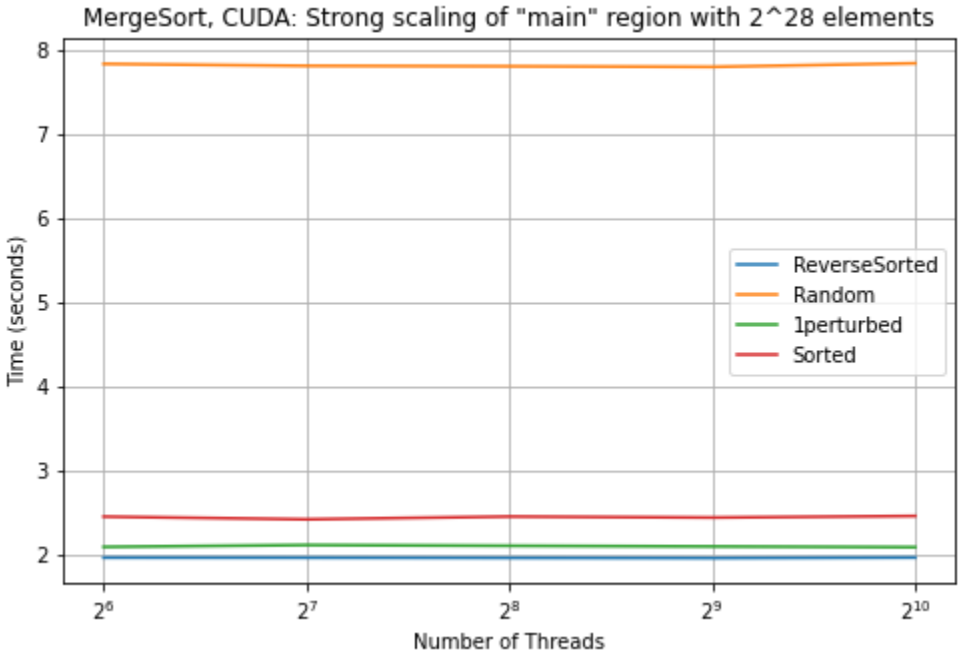

MergeSort, CUDA: Strong scaling of "main" region with 2^18 elements
Time (seconds)
0.360
0.355
0.350
0.345
0.340
0.335
0.330
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
Random
1perturbed
Sorted
ReverseSorted

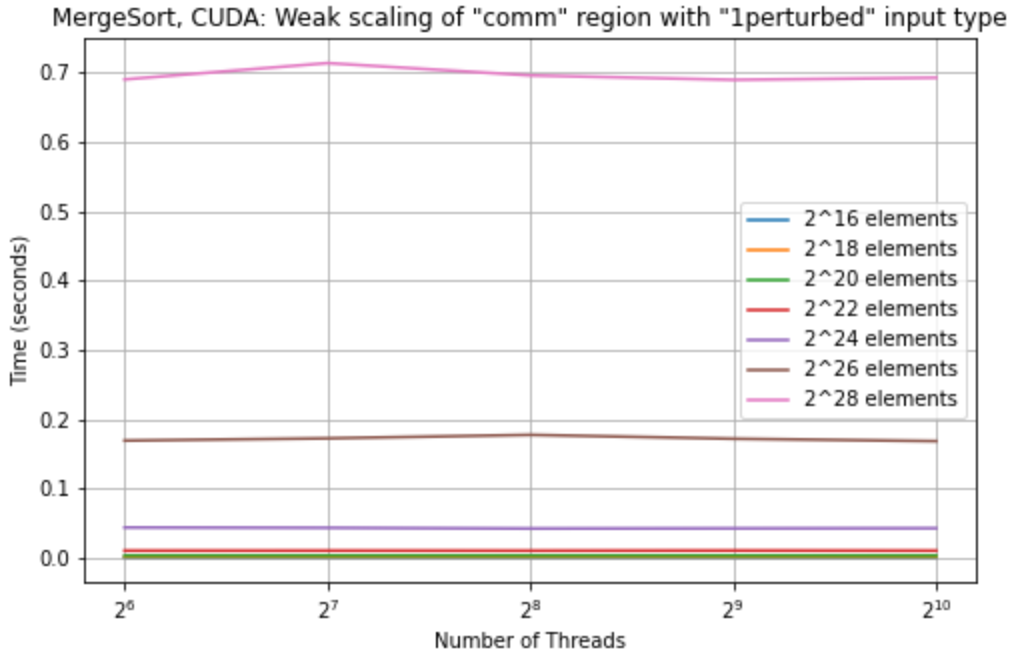

MergeSort, CUDA: Strong scaling of "main" region with 2^20 elements
Time (seconds)
0.375
0.370
0.365
0.360
0.355
0.350
0.345
0.340
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
ReverseSorted
1perturbed
Random
Sorted

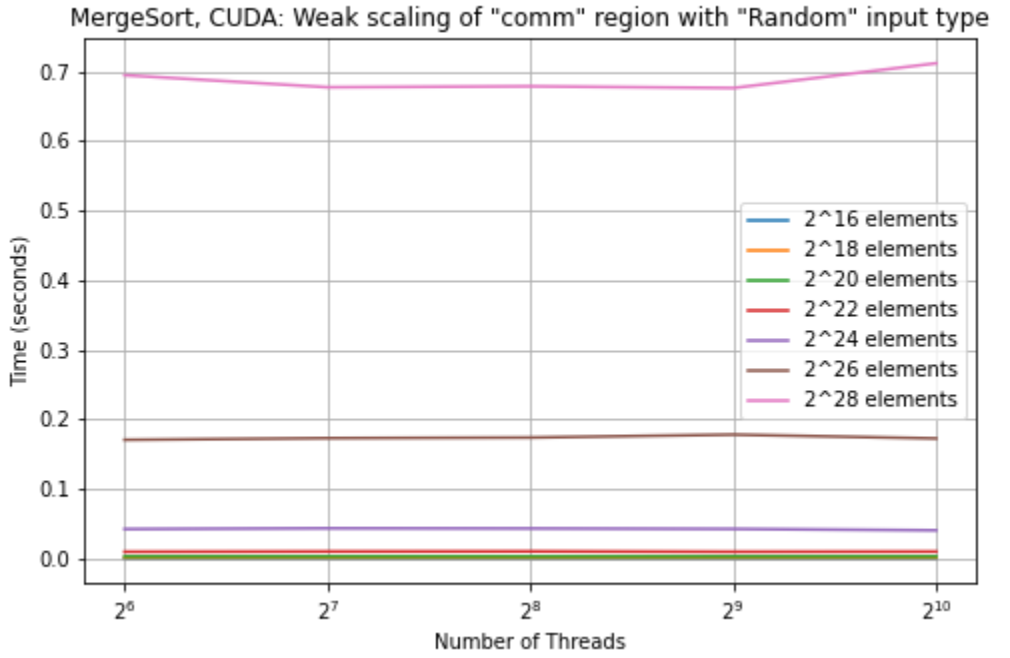

MergeSort, CUDA: Strong scaling of "main" region with 2^22 elements
Time (seconds)
0.46
0.44
0.42
0.40
0.38
0.36
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
1perturbed
Random
ReverseSorted
Sorted

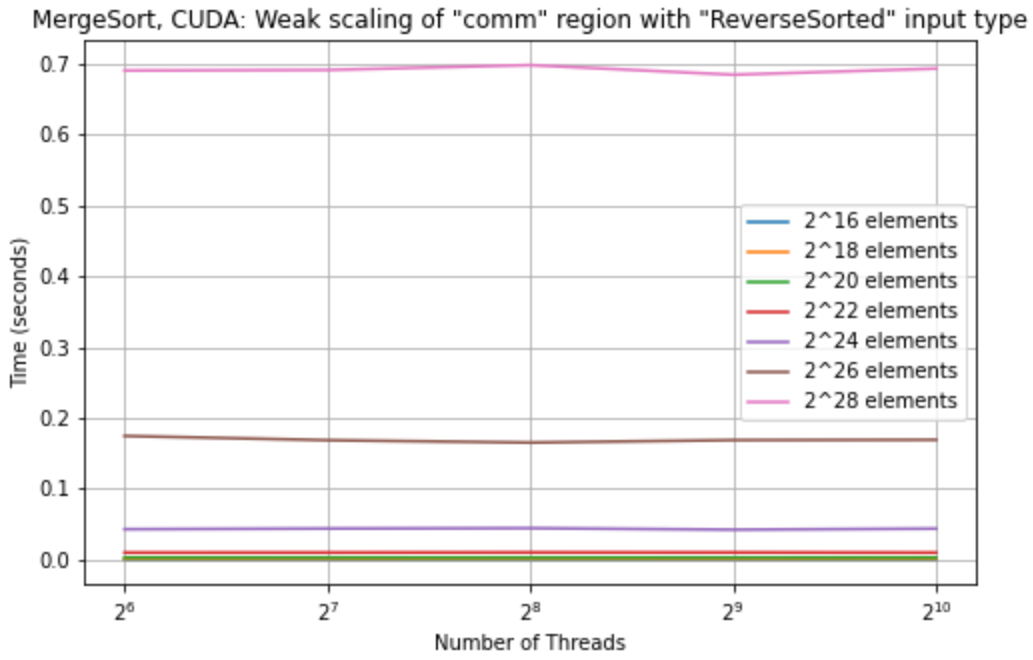

MergeSort, CUDA: Strong scaling of "main" region with 2^24 elements
Time (seconds)
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
1perturbed
Sorted
ReverseSorted
Random

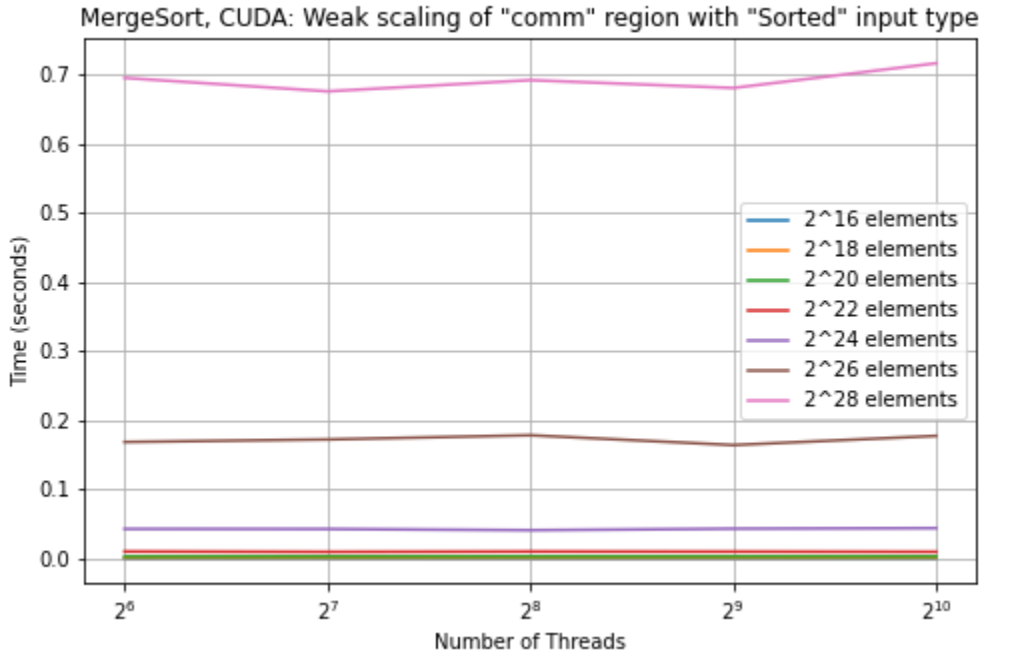

MergeSort, CUDA: Strong scaling of "main" region with 2^26 elements
Time (seconds)
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
1perturbed
Sorted
Random
ReverseSorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

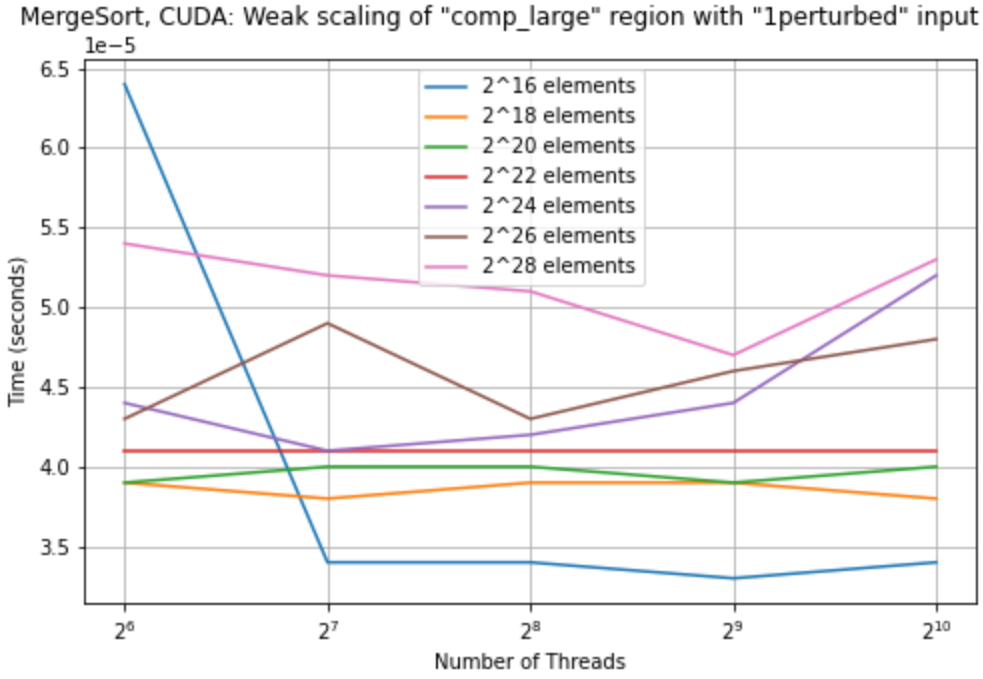

MergeSort, CUDA: Strong scaling of "main" region with 2^28 elements
Time (seconds)
8
7
6
5
4
3
2
ReverseSorted
Random
1perturbed
Sorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

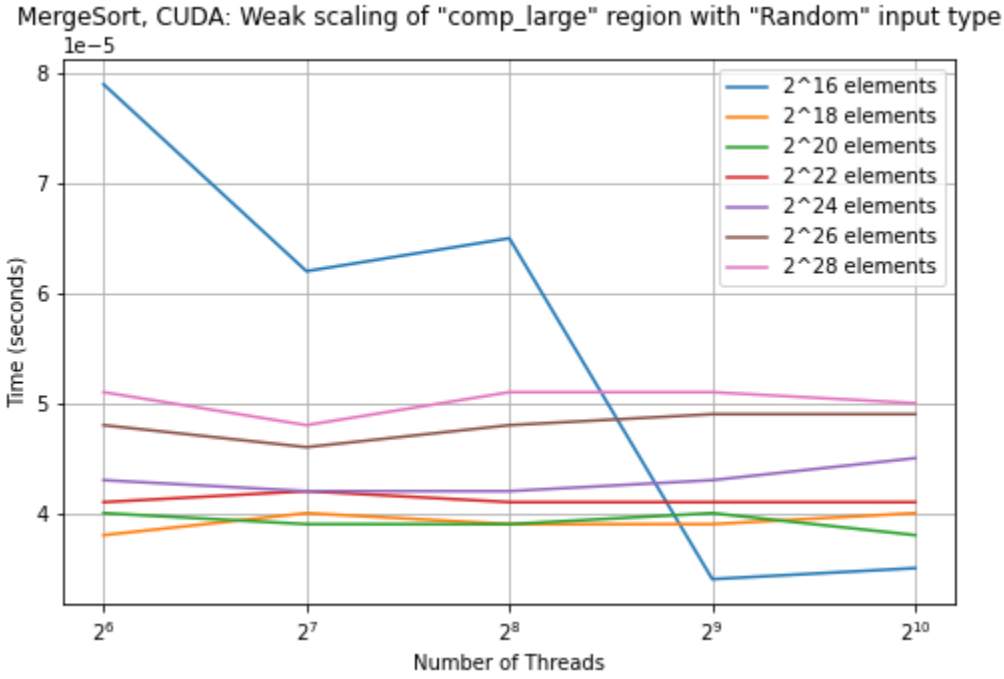

MergeSort, CUDA: Weak scaling of "comm" region with "1perturbed" input type
Time (seconds)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements


MergeSort, CUDA: Weak scaling of "comm" region with "Random" input type
Time (seconds)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

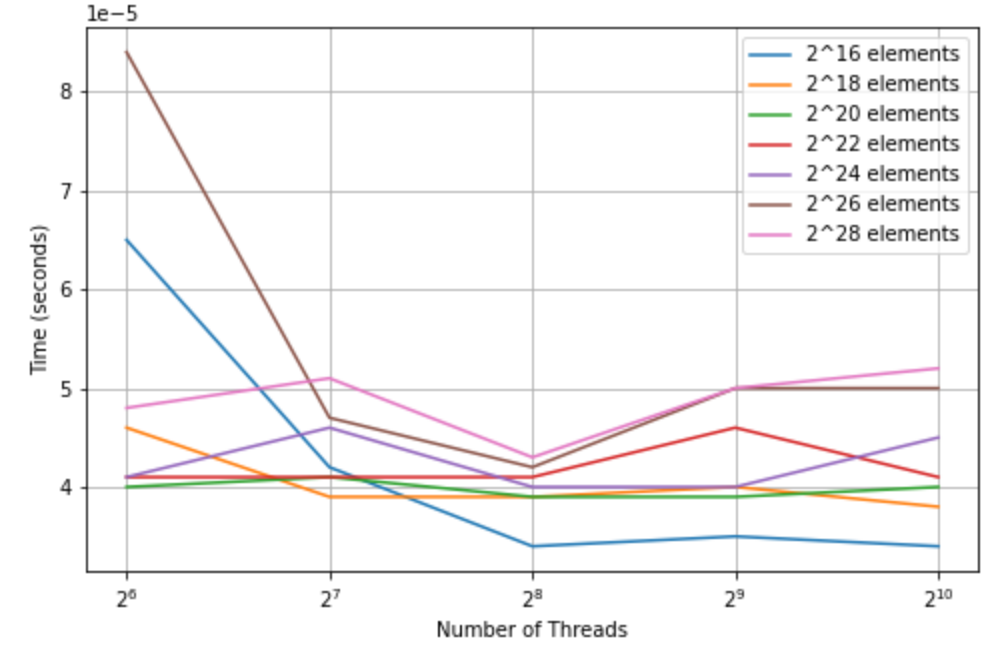

MergeSort, CUDA: Weak scaling of "comm" region with "ReverseSorted" input type
Time (seconds)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements


MergeSort, CUDA: Weak scaling of "comm" region with "Sorted" input type
Time (seconds)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

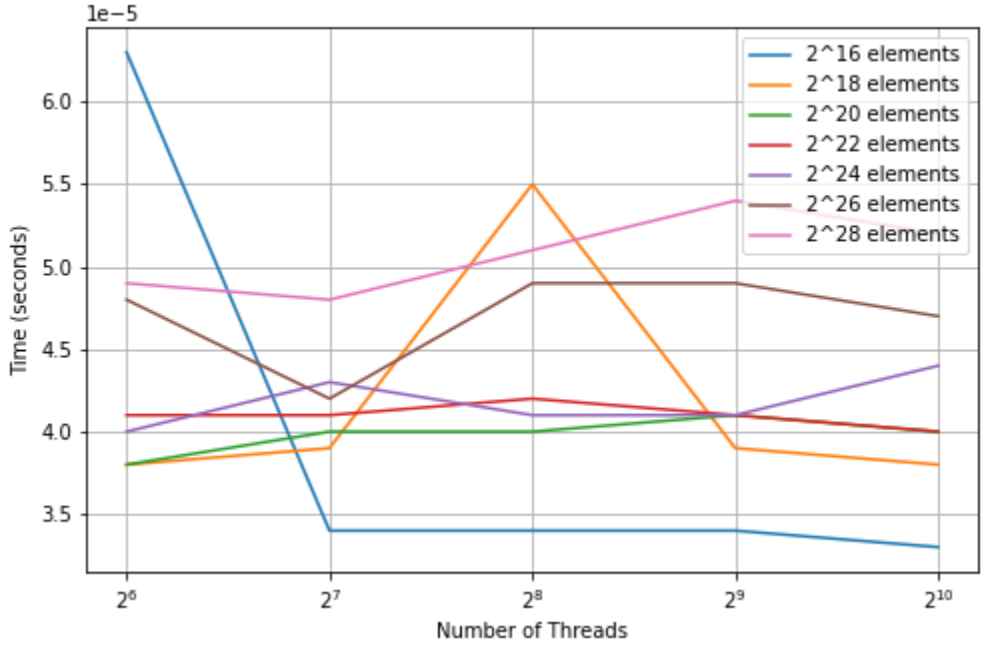

MergeSort, CUDA: Weak scaling of "comp_large" region with "1perturbed" input type
1e−5
6.5
6.0
5.5
5.0
4.5
4.0
3.5
Time (seconds)
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

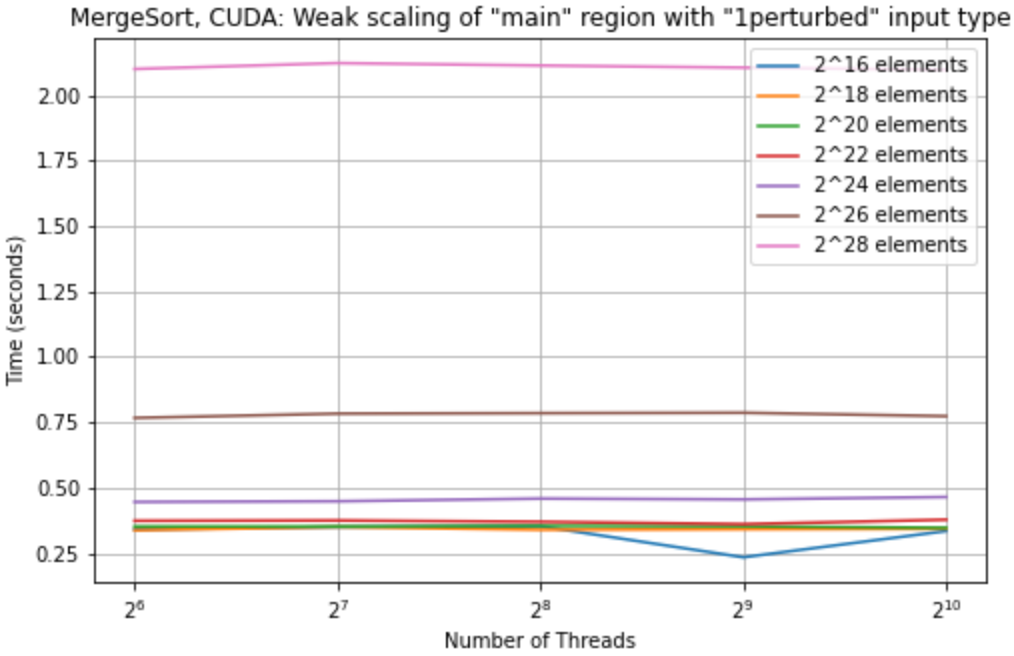

MergeSort, CUDA: Weak scaling of "comp_large" region with "Random" input type
1e−5
8
7
6
5
4
Time (seconds)
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

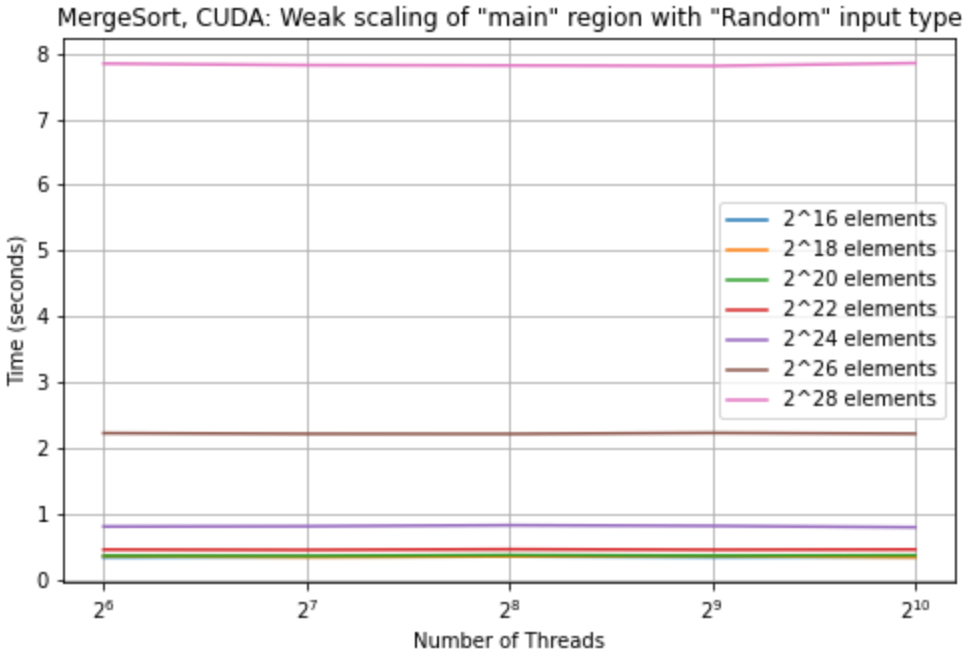
MergeSort, CUDA: Weak scaling of "comp_large" region with "ReverseSorted" input type

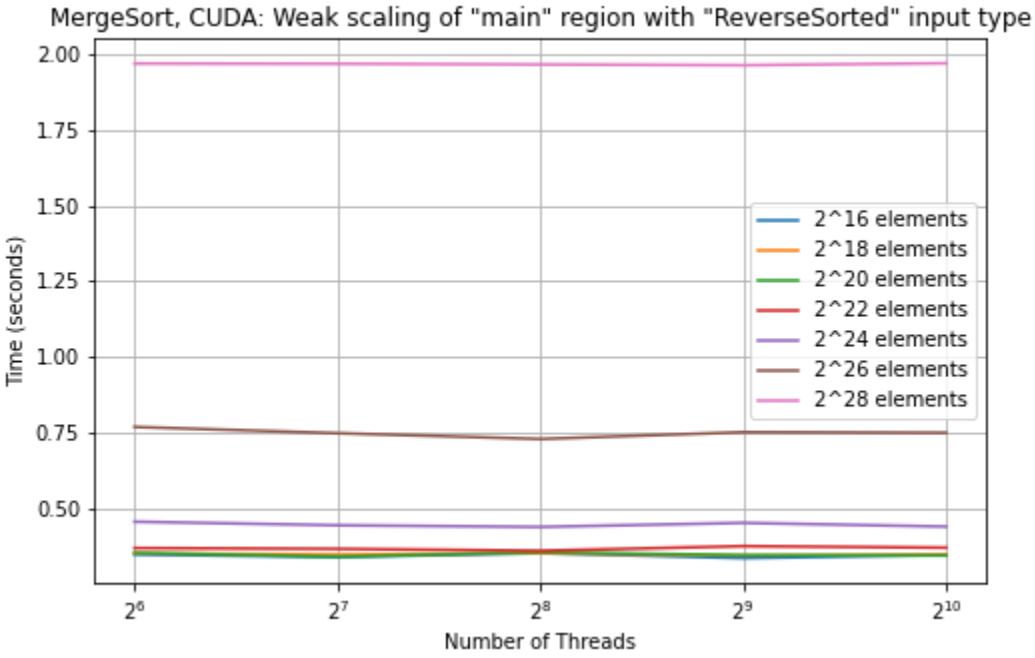


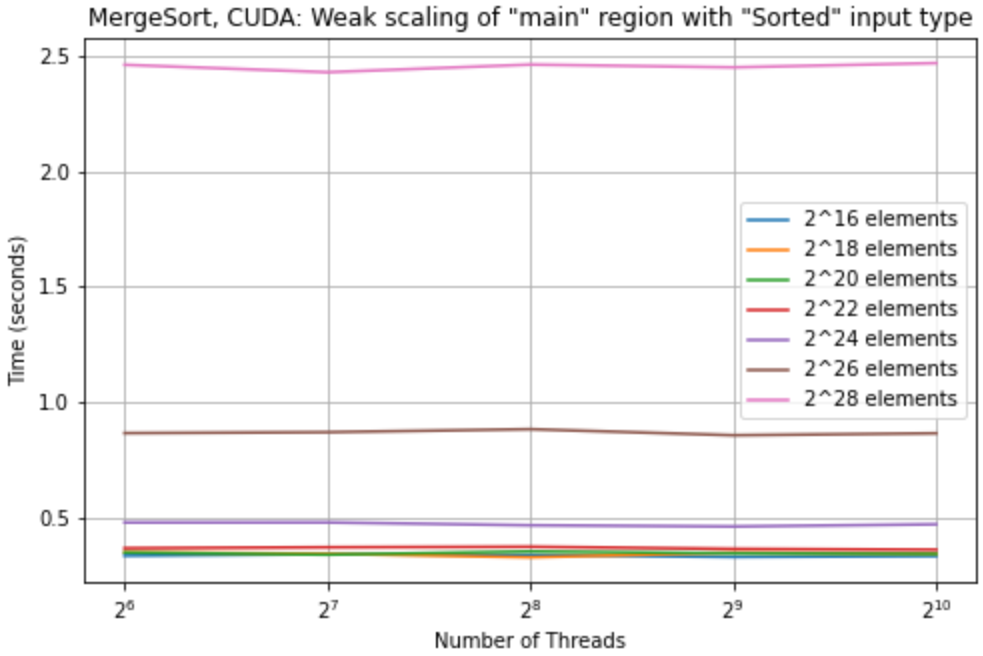

MergeSort, CUDA: Weak scaling of "comp_large" region with "Sorted" input type

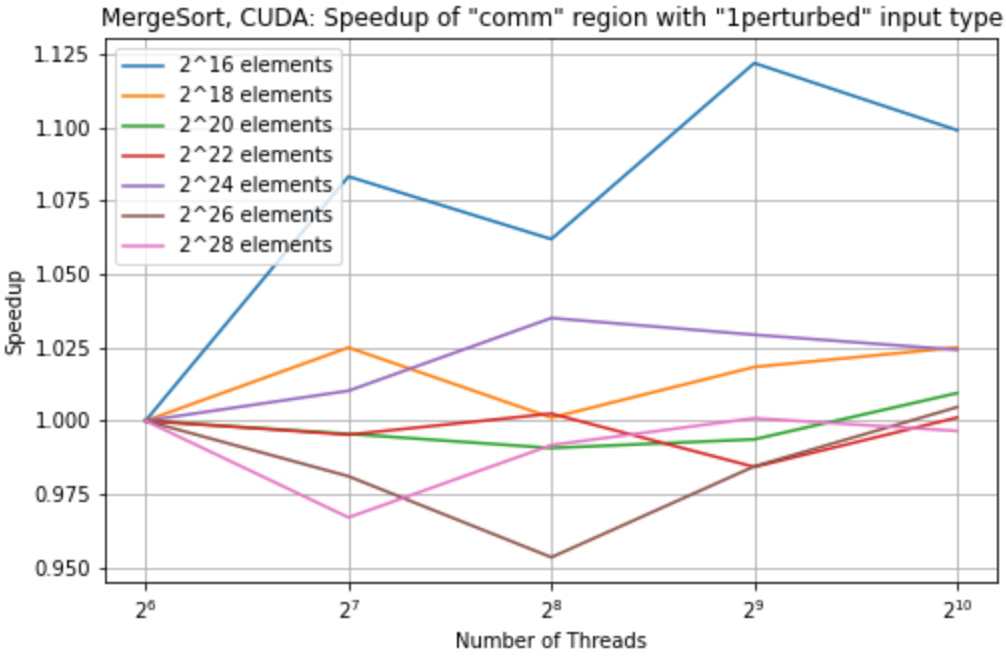


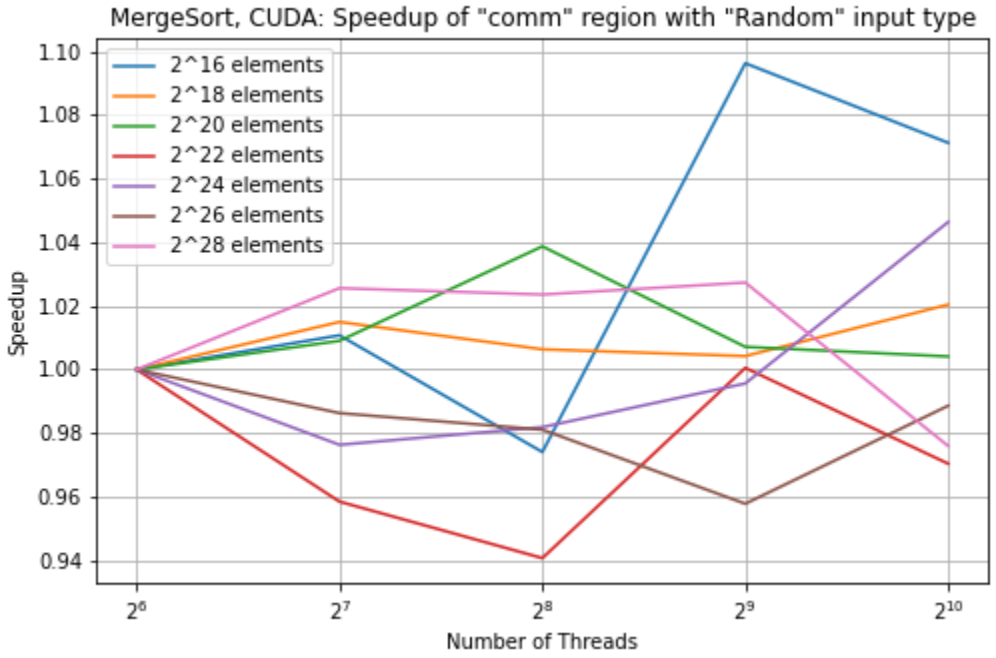

MergeSort, CUDA: Weak scaling of "main" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

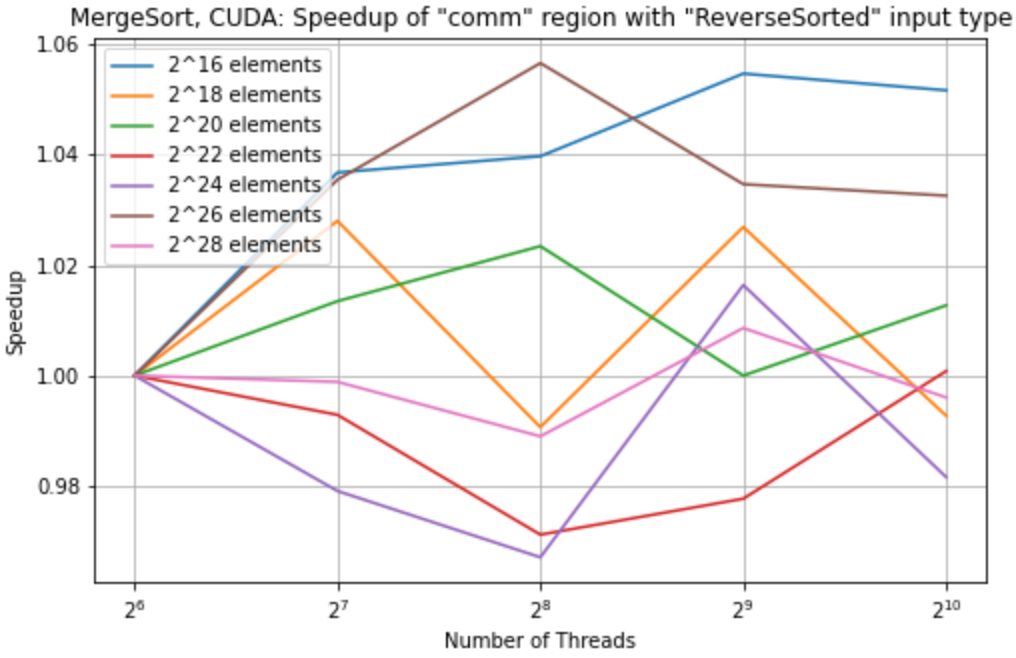

MergeSort, CUDA: Weak scaling of "main" region with "Random" input type
Time (seconds)
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

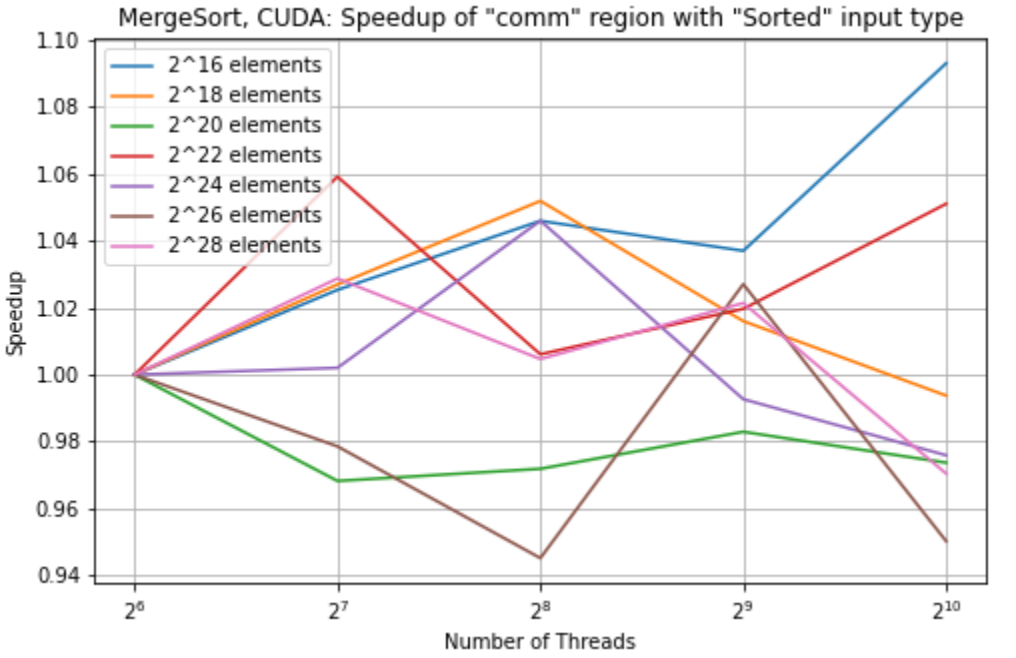

MergeSort, CUDA: Weak scaling of "main" region with "ReverseSorted" input type
Time (seconds)
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

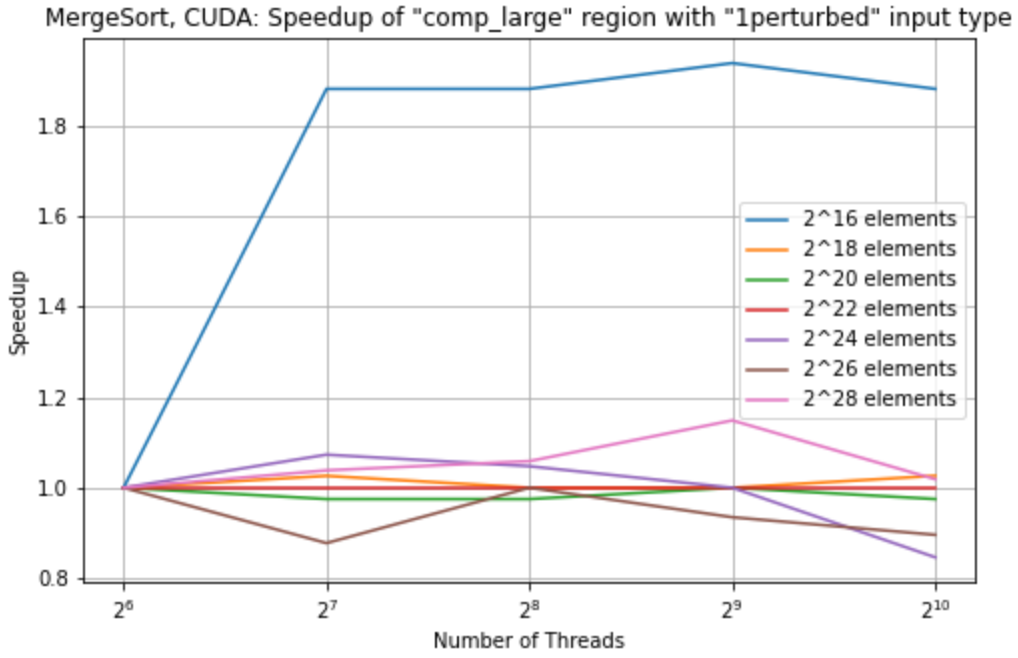

MergeSort, CUDA: Weak scaling of "main" region with "Sorted" input type
Time (seconds)
2.5
2.0
1.5
1.0
0.5
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

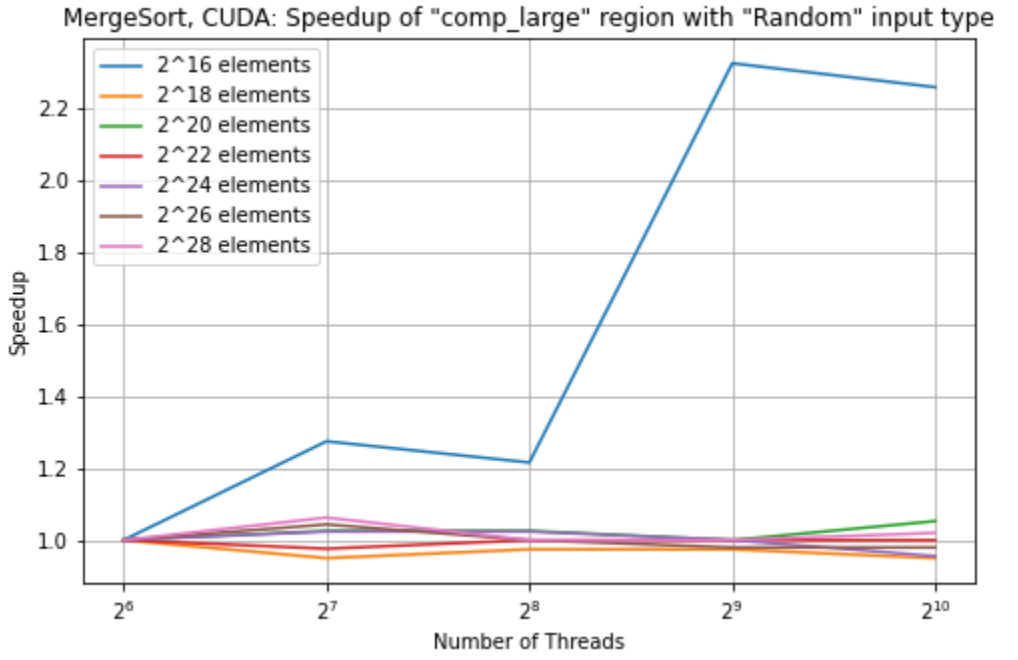

MergeSort, CUDA: Speedup of "comm" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
1.125
1.100
1.075
1.050
1.025
1.000
0.975
0.950
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

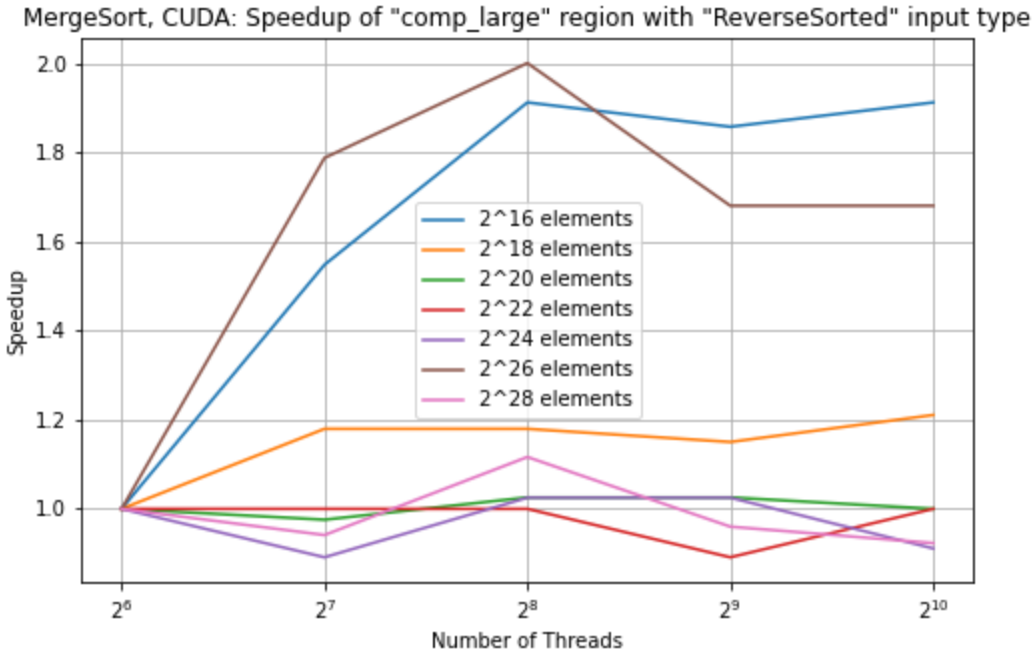

MergeSort, CUDA: Speedup of "comm" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
1.10
1.08
1.06
1.04
1.02
1.00
0.98
0.96
0.94
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

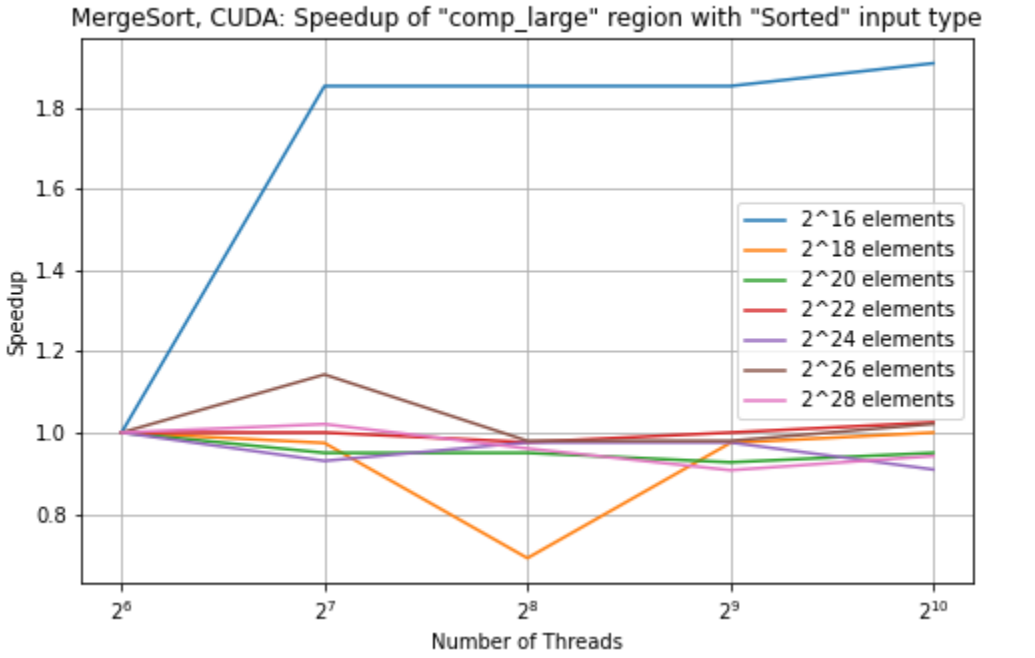

MergeSort, CUDA: Speedup of "comm" region with "ReverseSorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
1.06
1.04
1.02
1.00
0.98
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

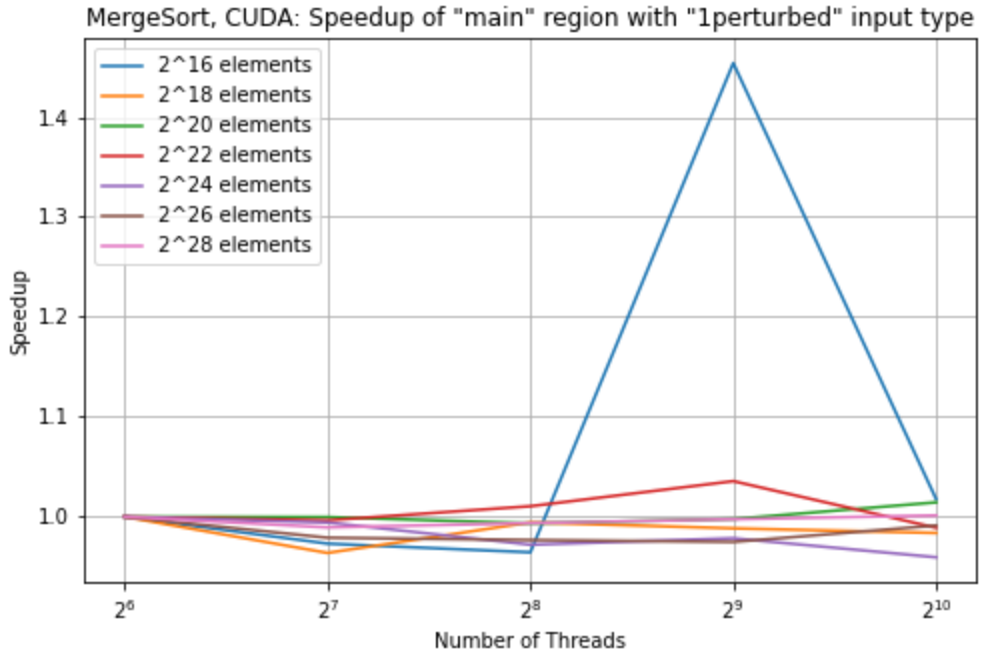

MergeSort, CUDA: Speedup of "comm" region with "Sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
1.10
1.08
1.06
1.04
1.02
1.00
0.98
0.96
0.94
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

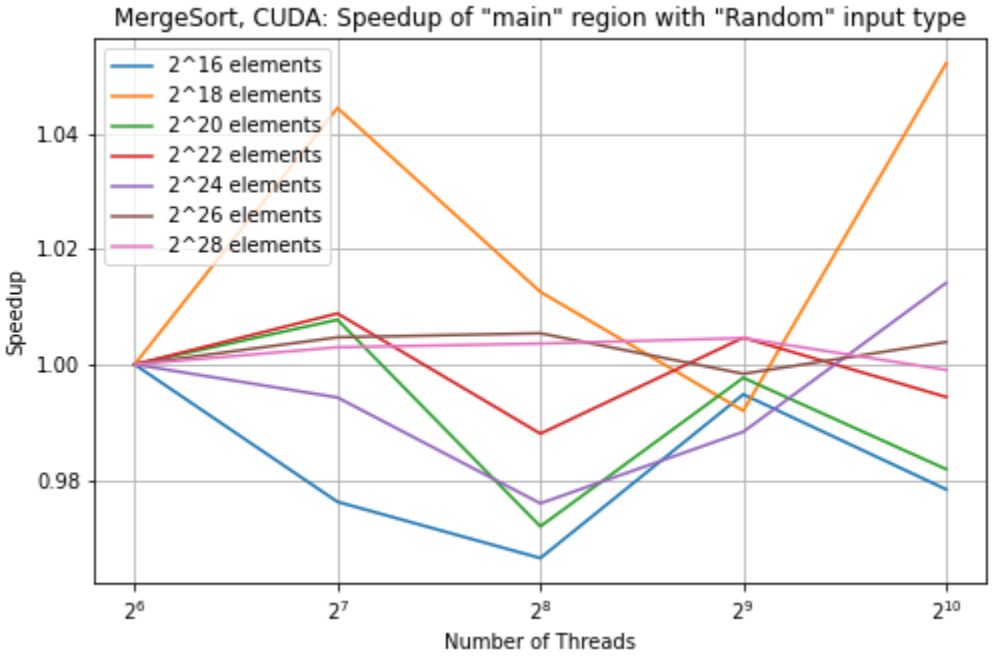

MergeSort, CUDA: Speedup of "comp_large" region with "1perturbed" input type
Speedup
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

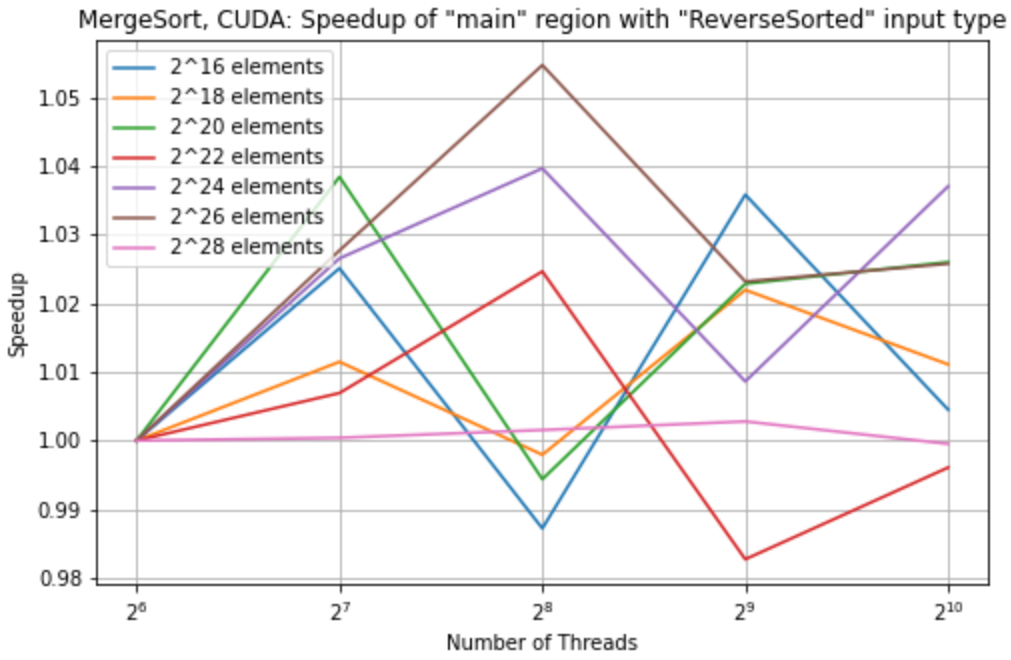

MergeSort, CUDA: Speedup of "comp_large" region with "Random" input type
Speedup
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

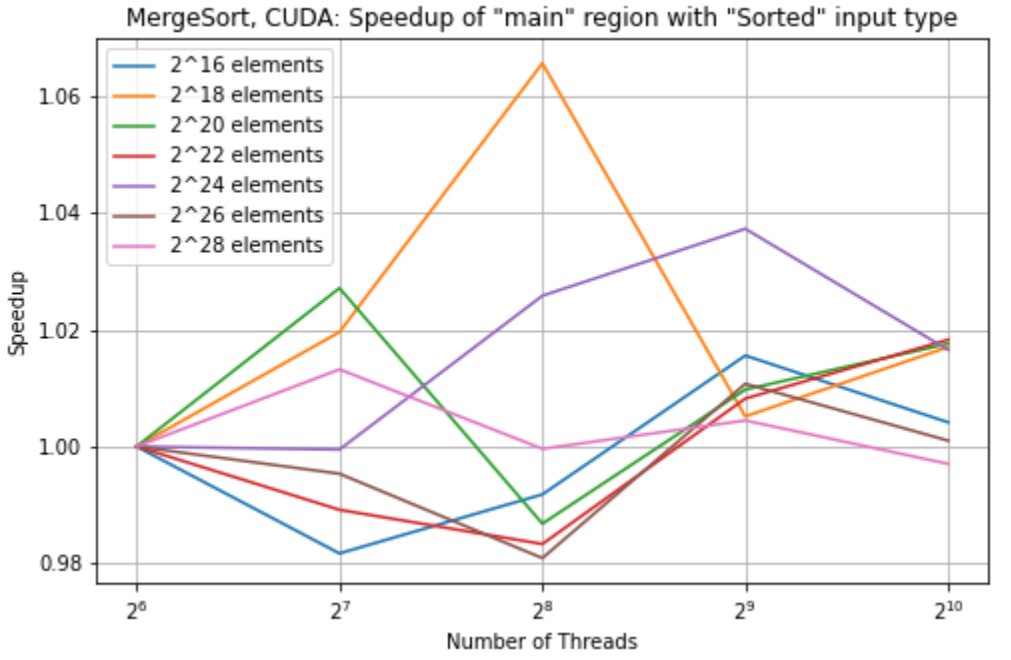

MergeSort, CUDA: Speedup of "comp_large" region with "ReverseSorted" input type
Speedup
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements


MergeSort, CUDA: Speedup of "comp_large" region with "Sorted" input type
Speedup
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements


MergeSort, CUDA: Speedup of "main" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
1.4
1.3
1.2
1.1
1.0
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads


MergeSort, CUDA: Speedup of "main" region with "Random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
1.04
1.02
1.00
0.98
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

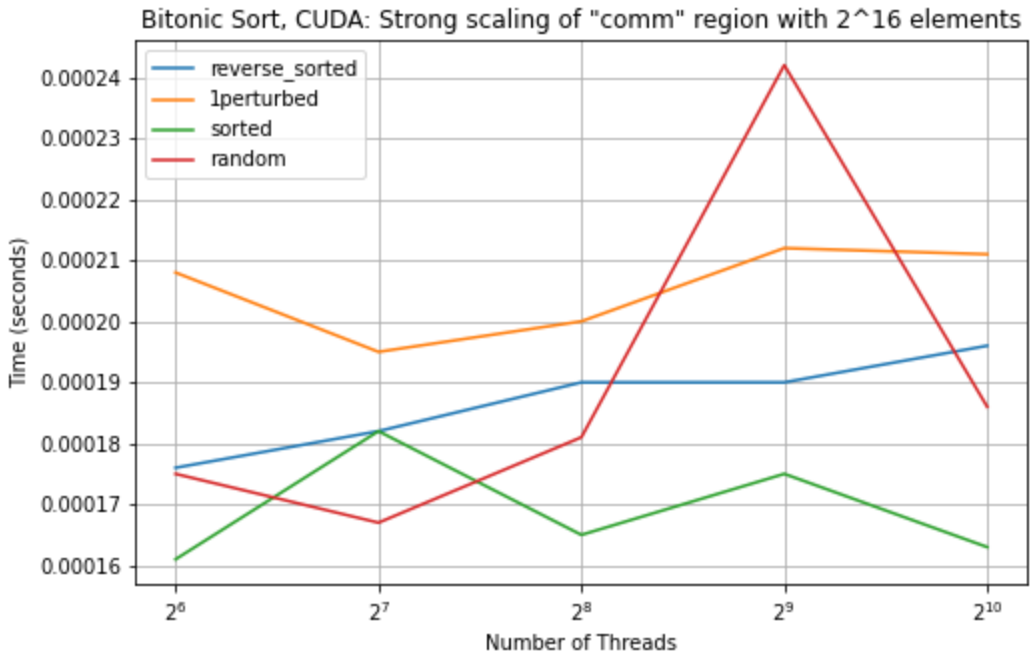


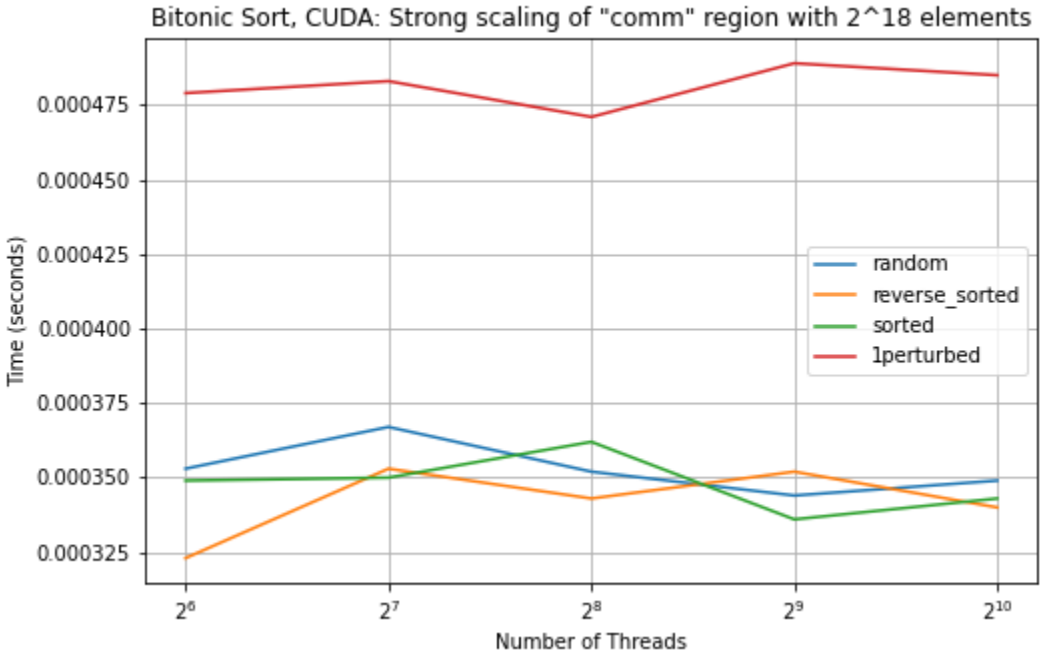



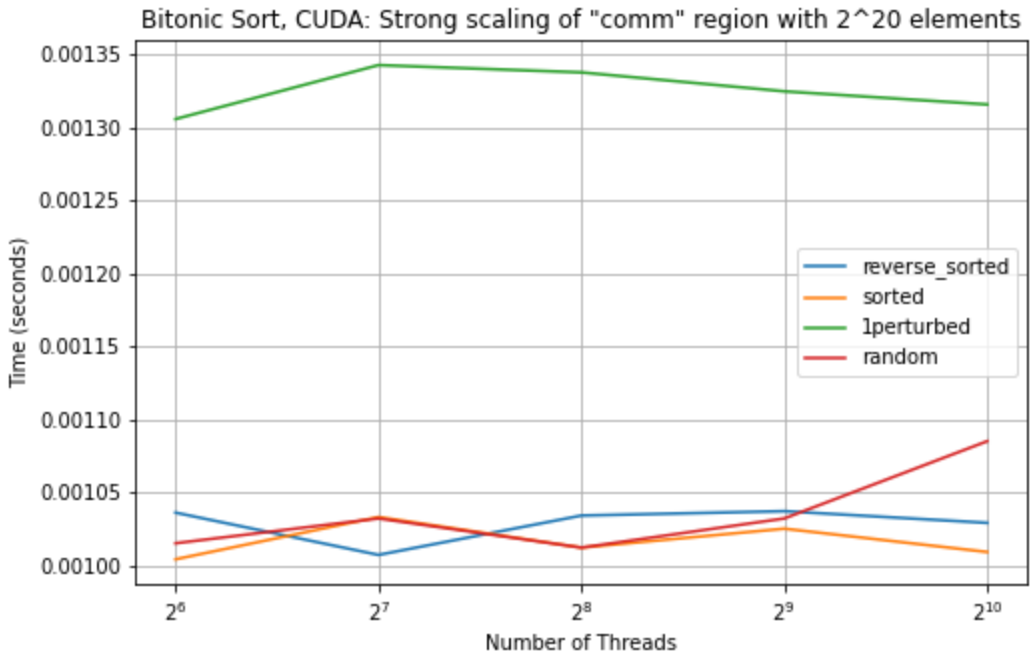

As the number of threads increases, the runtime of the mergesort CUDA algorithm varies greatly. There is not a clear trend of decreasing runtime with increasing threads from these graphs, which may signal an implementation issue with CUDA mergesort in this code. For the smallest array input size,  there is a general trend of time reduction in

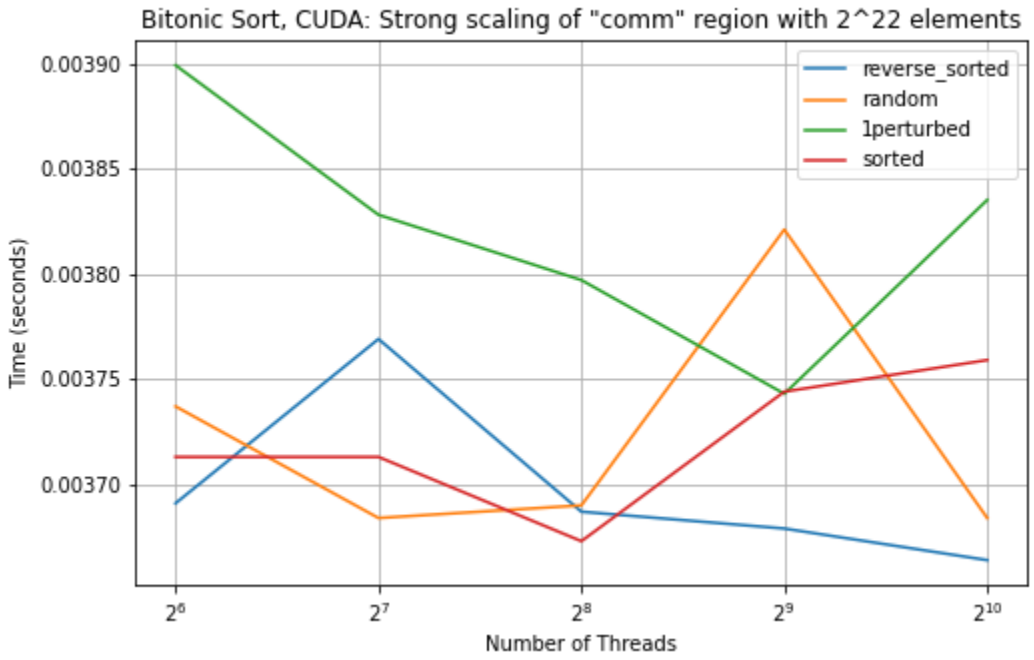
the strong scaling graphs, however, the higher array input sizes fail to show clear trends. There are linear lines in the weak scaling of the algorithm at various input arrays, signaling that regardless of the thread size, the algorithm runtime will stay consistent. This shows that the splitting of the tasks between threads is not happening efficiently, or the sequential portions of the algorithm are still having the greatest impact on overall performance. For some array input sizes, there is minimal speedup of around 1, which again might indicate that the mergesort CUDA implementation is not effectively splitting up the tasks.

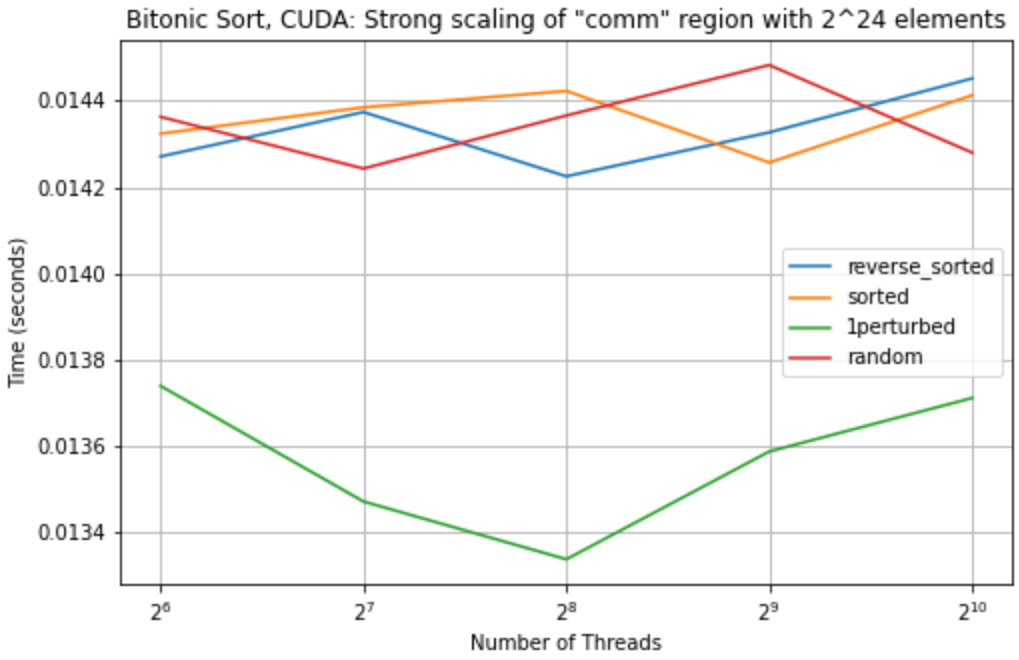
## Bitonic Sort CUDA

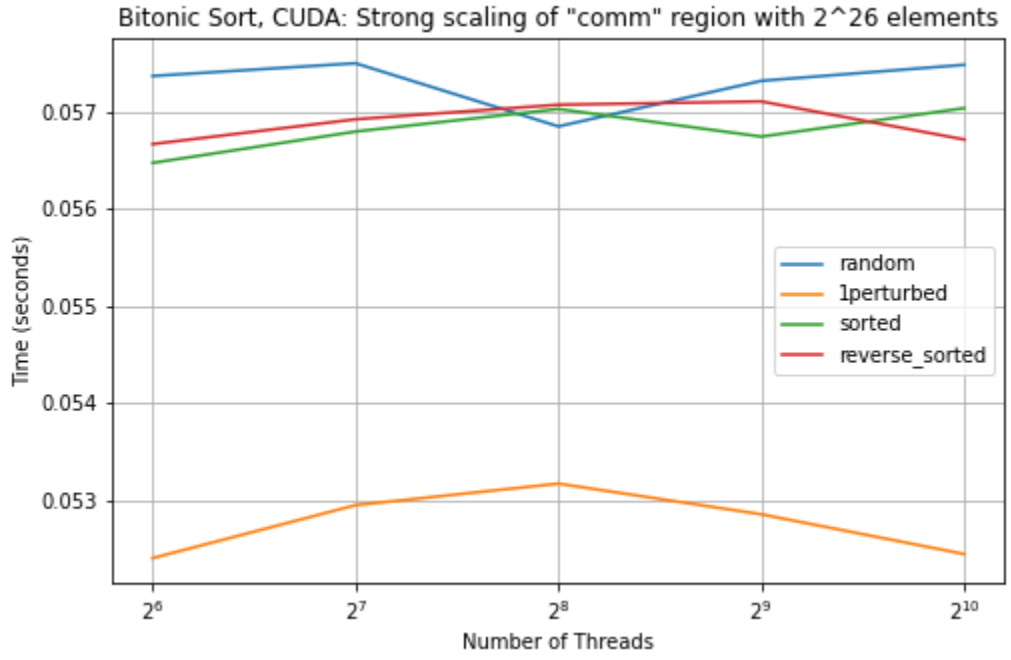

Bitonic Sort, CUDA: Strong scaling of "comm" region with 2^16 elements
reverse_sorted
1perturbed
sorted
random
Time (seconds)
0.00024
0.00023
0.00022
0.00021
0.00020
0.00019
0.00018
0.00017
0.00016
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

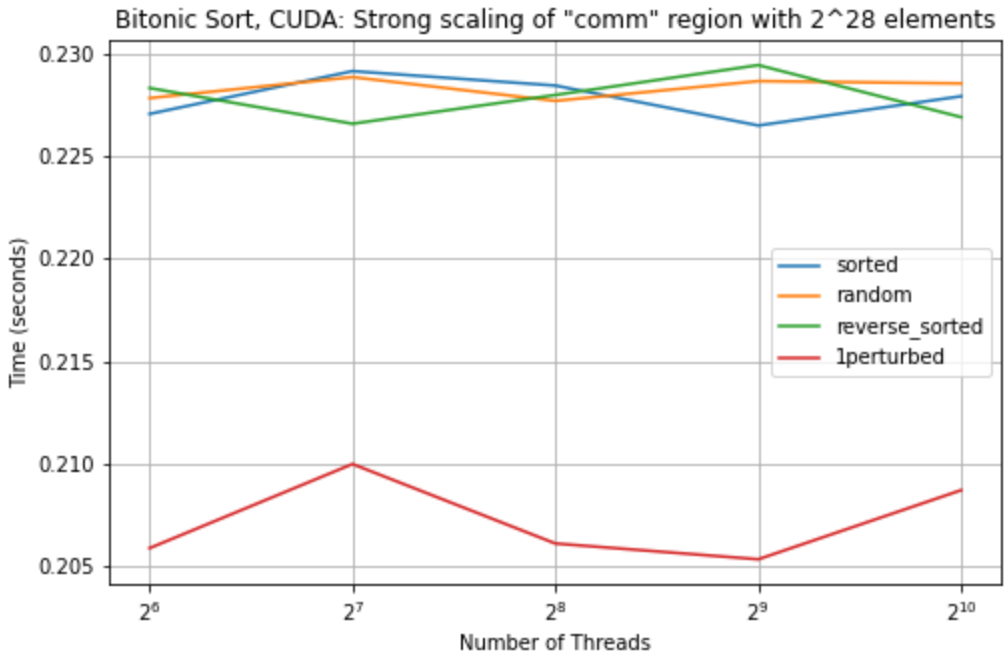

Bitonic Sort, CUDA: Strong scaling of "comm" region with 2^18 elements
random
reverse_sorted
sorted
1perturbed
Time (seconds)
0.000475
0.000450
0.000425
0.000400
0.000375
0.000350
0.000325
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

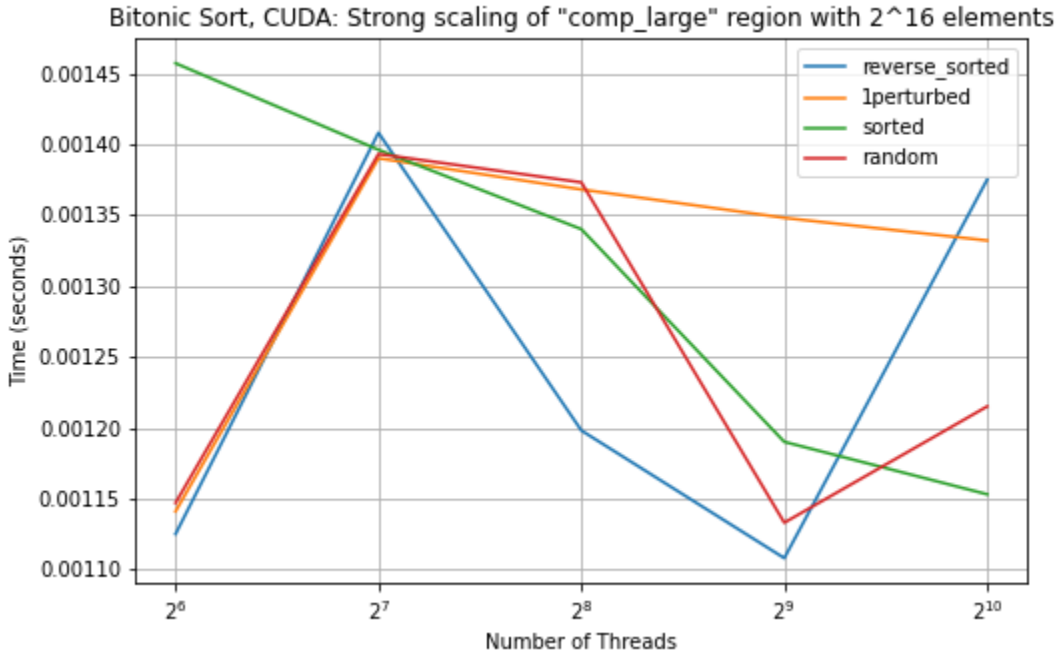

Bitonic Sort, CUDA: Strong scaling of "comm" region with 2^20 elements
Time (seconds)
0.00135
0.00130
0.00125
0.00120
0.00115
0.00110
0.00105
0.00100
reverse_sorted
sorted
1perturbed
random
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

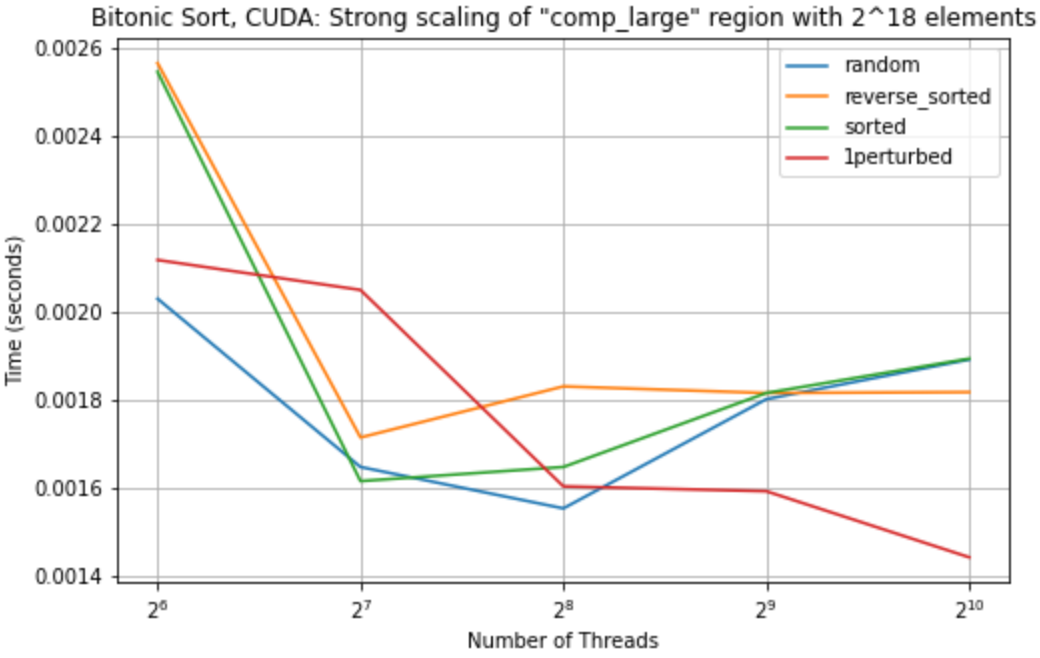

Bitonic Sort, CUDA: Strong scaling of "comm" region with 2^22 elements
Time (seconds)
0.00390
0.00385
0.00380
0.00375
0.00370
reverse_sorted
random
1perturbed
sorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

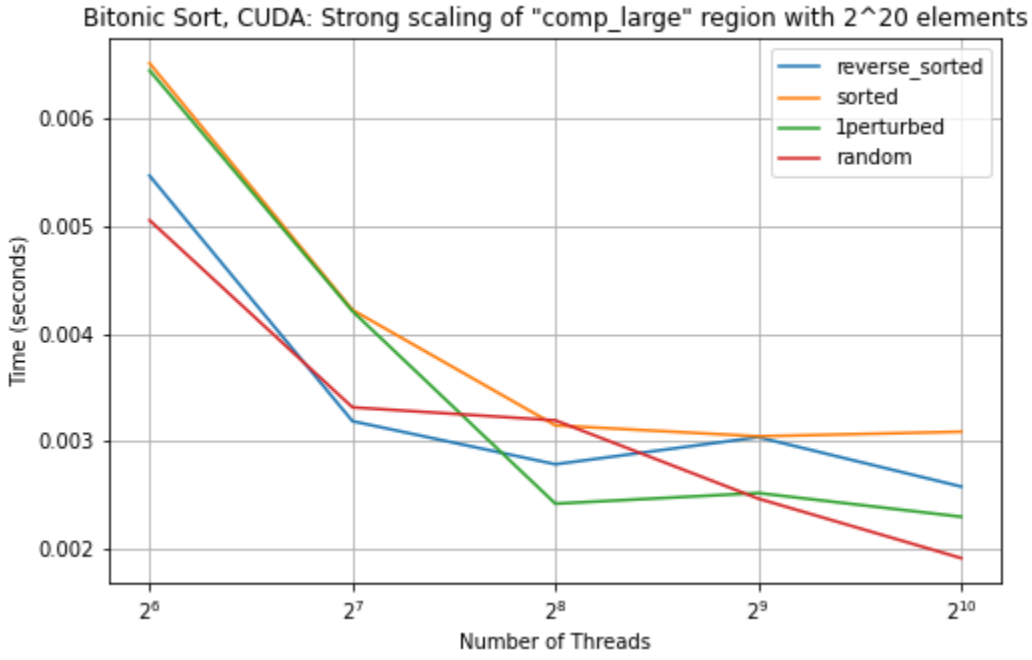

Bitonic Sort, CUDA: Strong scaling of "comm" region with 2^24 elements
Time (seconds)
0.0144
0.0142
0.0140
0.0138
0.0136
0.0134
reverse_sorted
sorted
1perturbed
random
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

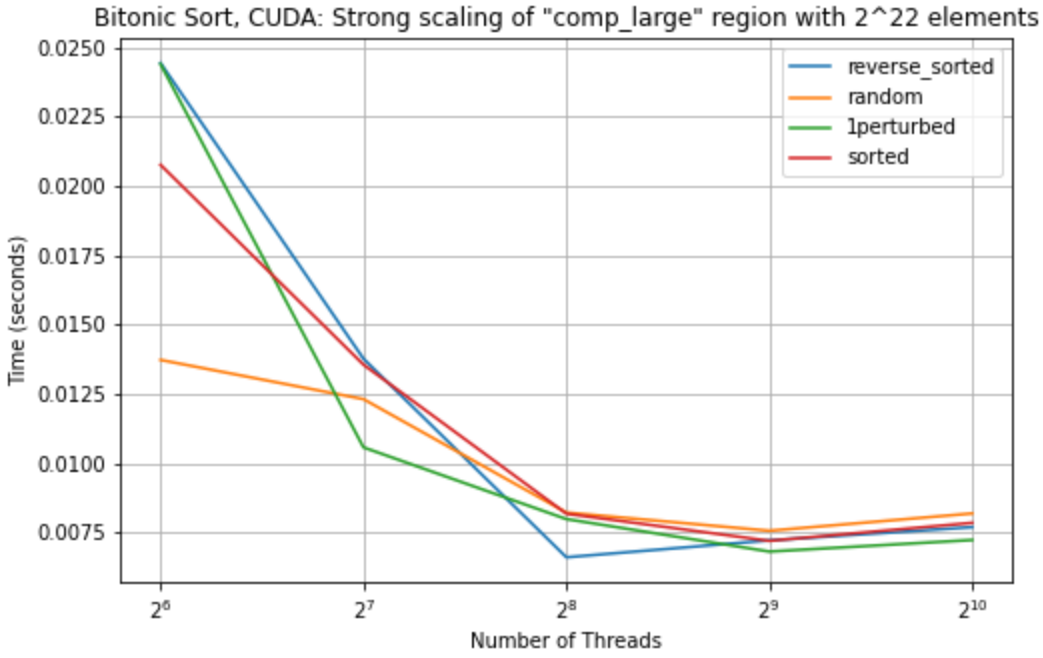

Bitonic Sort, CUDA: Strong scaling of "comm" region with 2^26 elements
Time (seconds)
0.057
0.056
0.055
0.054
0.053
random
1perturbed
sorted
reverse_sorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

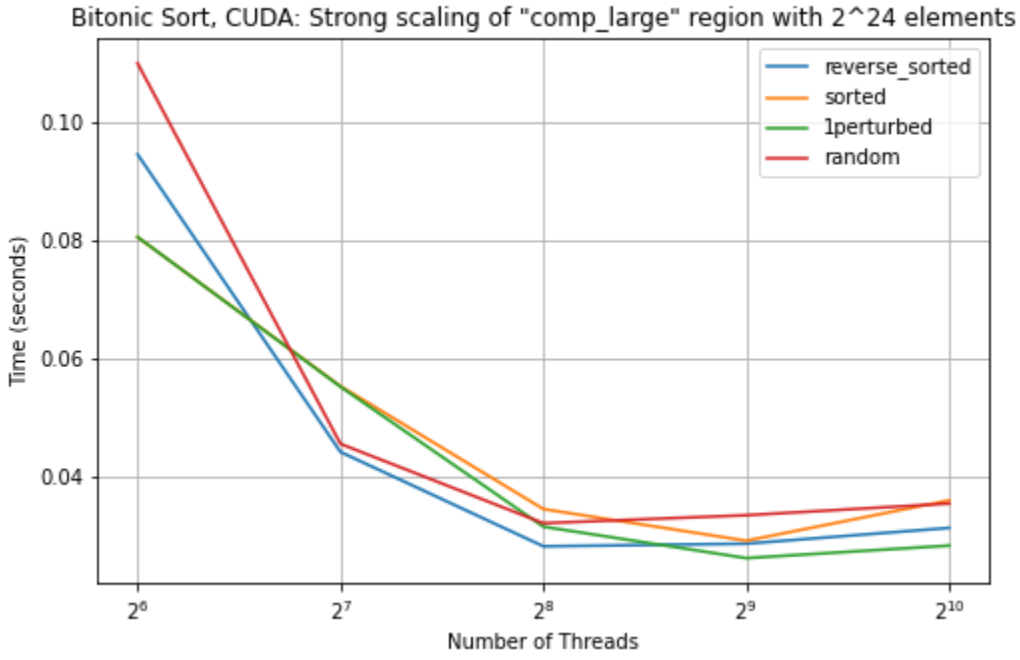

Bitonic Sort, CUDA: Strong scaling of "comm" region with 2^28 elements
Time (seconds)
0.230
0.225
0.220
0.215
0.210
0.205
sorted
random
reverse_sorted
1perturbed
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

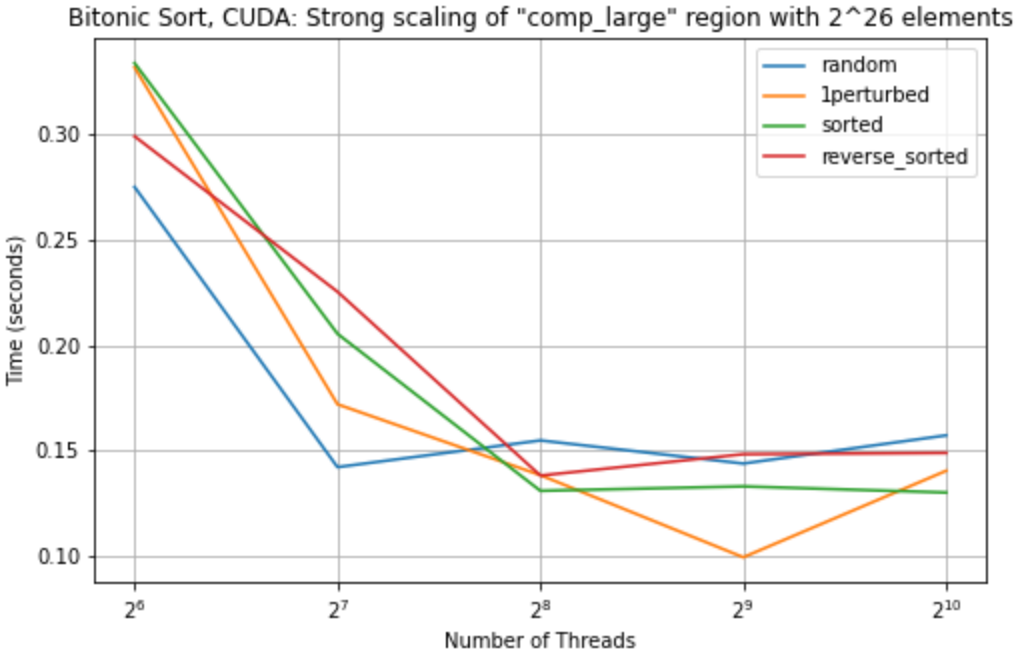

Bitonic Sort, CUDA: Strong scaling of "comp_large" region with 2^16 elements
Time (seconds)
0.00145
0.00140
0.00135
0.00130
0.00125
0.00120
0.00115
0.00110
reverse_sorted
1perturbed
sorted
random
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

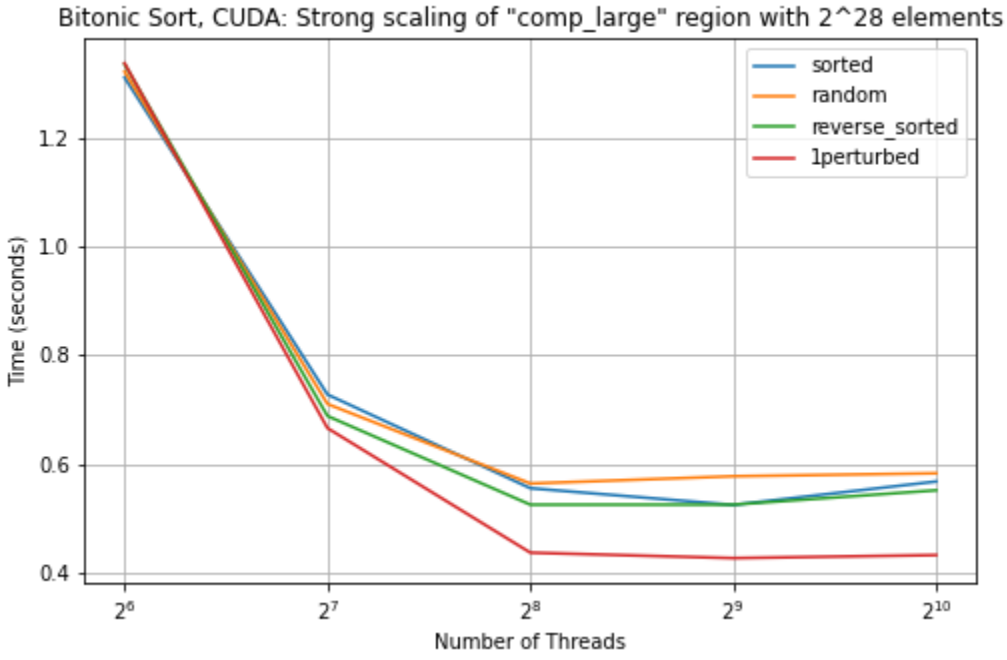

Bitonic Sort, CUDA: Strong scaling of "comp_large" region with 2^18 elements
Time (seconds)
0.0026
0.0024
0.0022
0.0020
0.0018
0.0016
0.0014
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
random
reverse_sorted
sorted
1perturbed

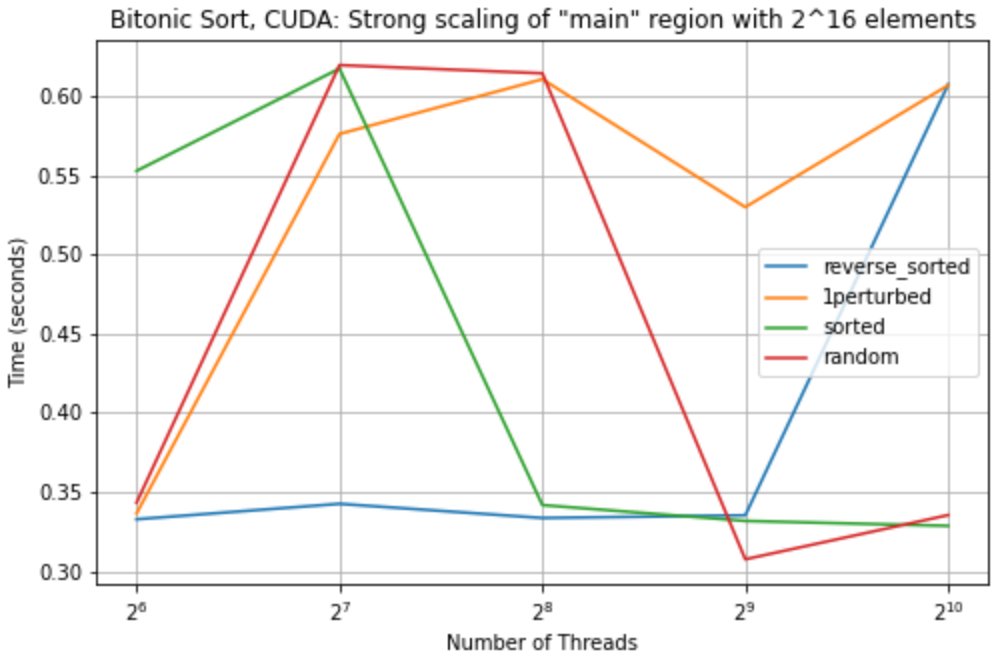

Bitonic Sort, CUDA: Strong scaling of "comp_large" region with 2^20 elements
Time (seconds)
0.006
0.005
0.004
0.003
0.002
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
reverse_sorted
sorted
1perturbed
random

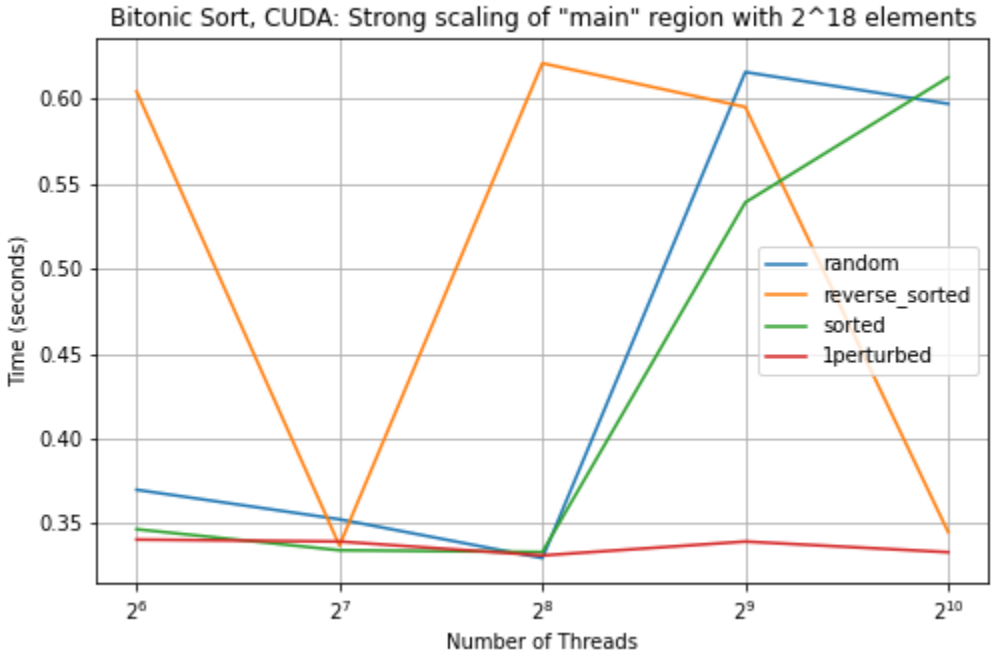

Bitonic Sort, CUDA: Strong scaling of "comp_large" region with 2^22 elements
Time (seconds)
0.0250
0.0225
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
reverse_sorted
random
1perturbed
sorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

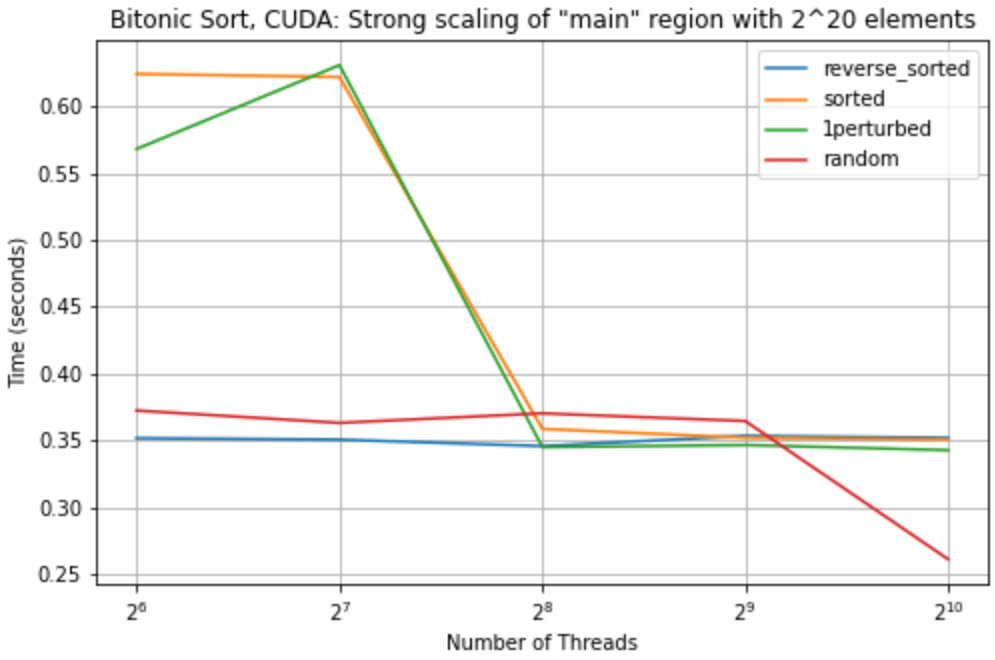

Bitonic Sort, CUDA: Strong scaling of "comp_large" region with 2^24 elements
Time (seconds)
0.10
0.08
0.06
0.04
reverse_sorted
sorted
1perturbed
random
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

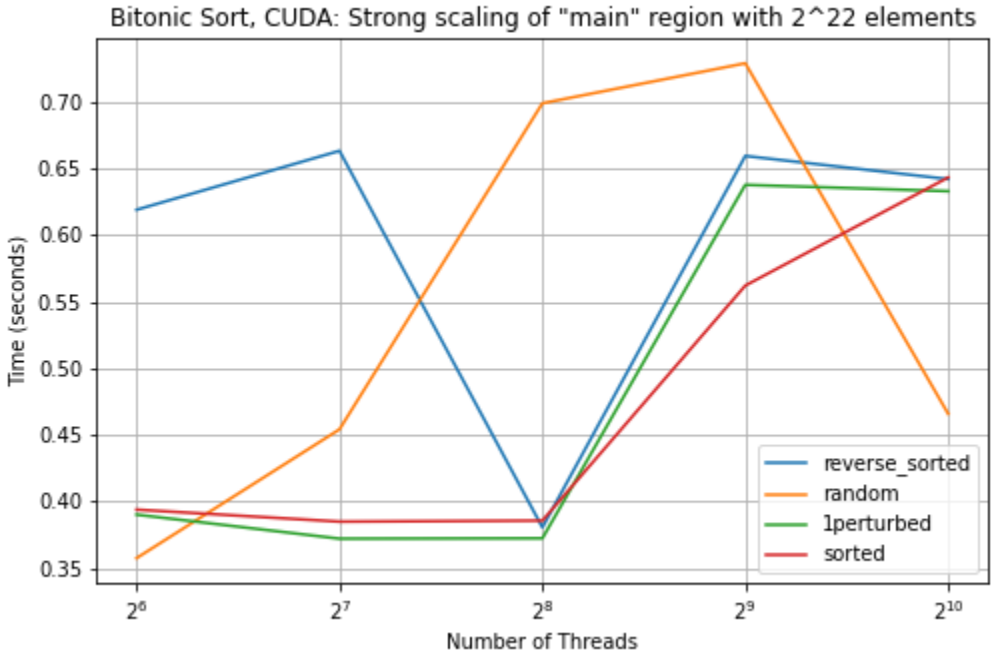

Bitonic Sort, CUDA: Strong scaling of "comp_large" region with 2^26 elements
random
1perturbed
sorted
reverse_sorted
Time (seconds)
0.30
0.25
0.20
0.15
0.10
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

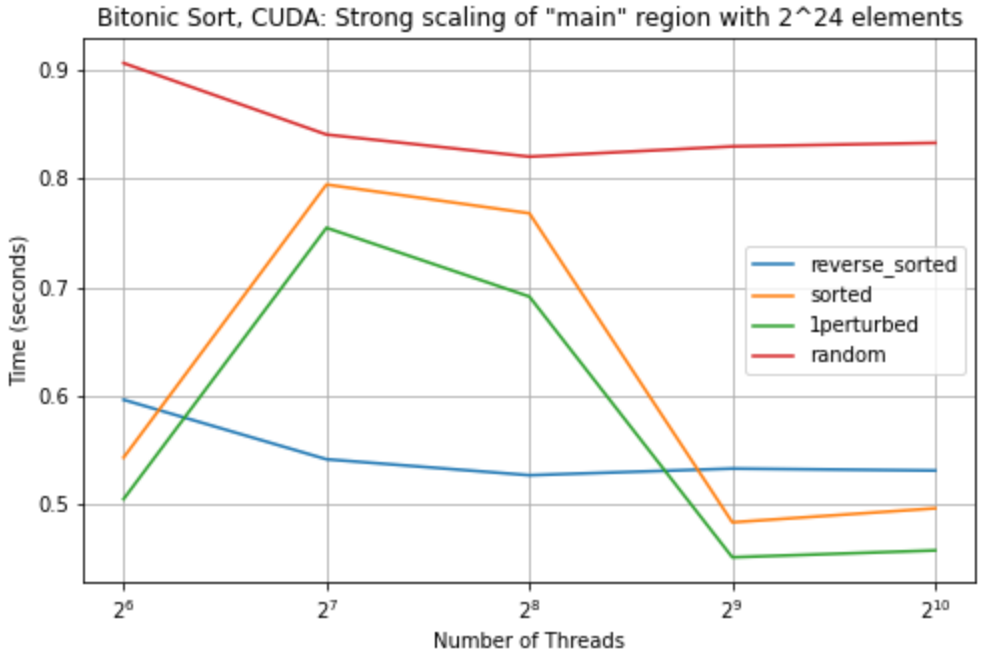

Bitonic Sort, CUDA: Strong scaling of "comp_large" region with 2^28 elements
sorted
random
reverse_sorted
1perturbed
Time (seconds)
1.2
1.0
0.8
0.6
0.4
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

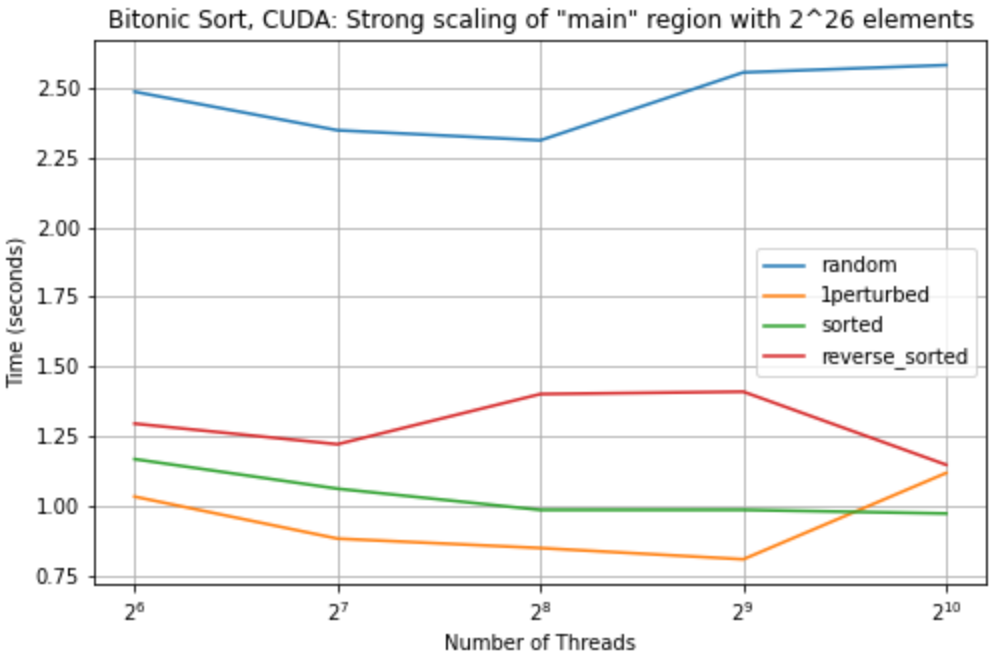

Bitonic Sort, CUDA: Strong scaling of "main" region with 2^16 elements
Time (seconds)
0.60
0.55
0.50
0.45
0.40
0.35
0.30
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
reverse_sorted
1perturbed
sorted
random

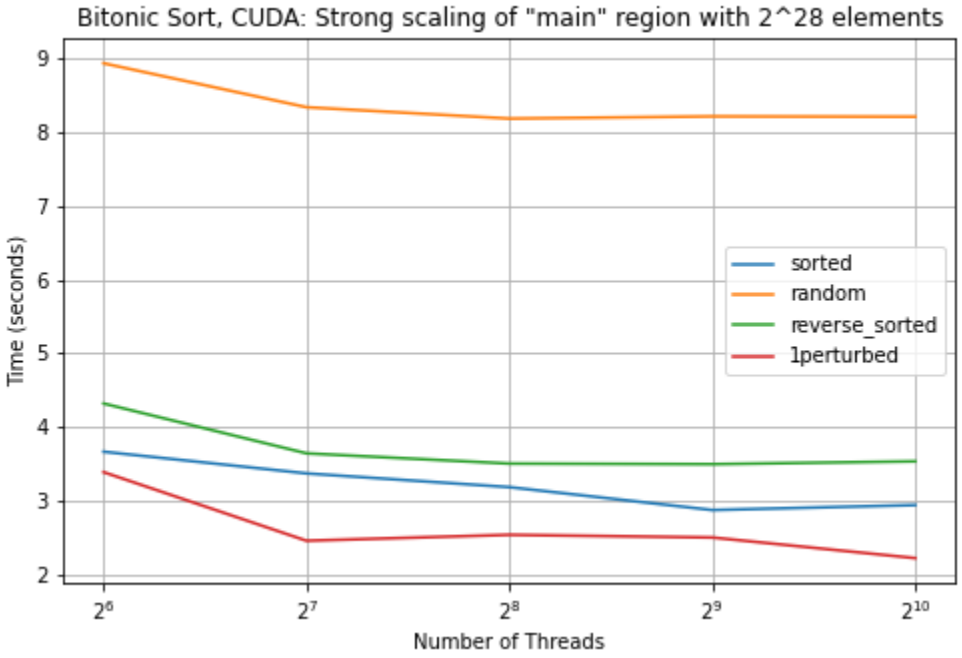

Bitonic Sort, CUDA: Strong scaling of "main" region with 2^18 elements
Time (seconds)
0.60
0.55
0.50
0.45
0.40
0.35
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
random
reverse_sorted
sorted
1perturbed

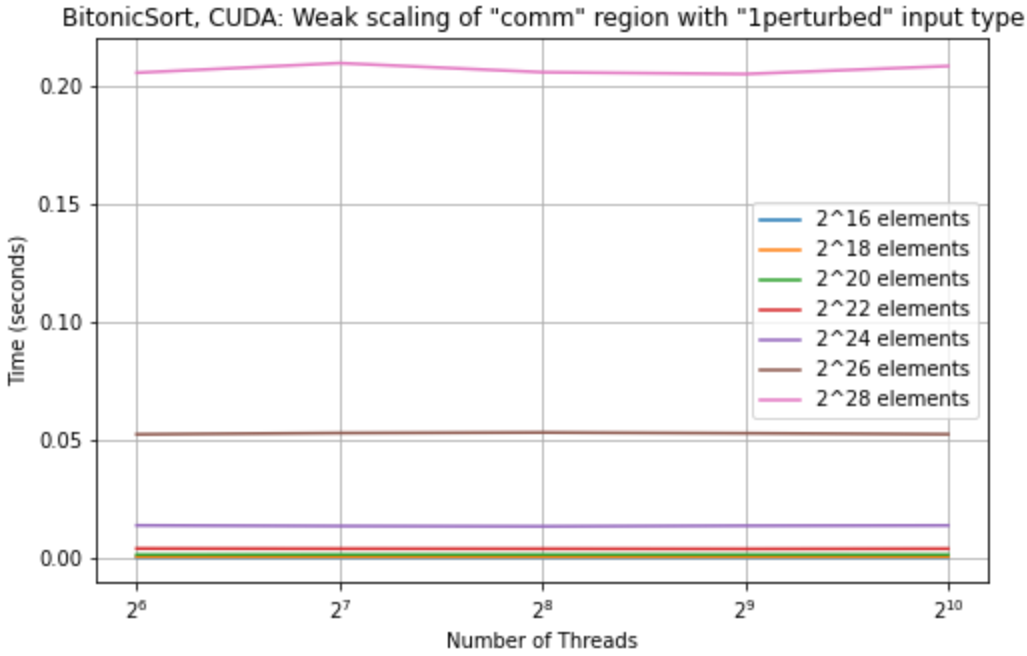

Bitonic Sort, CUDA: Strong scaling of "main" region with 2^20 elements
Time (seconds)
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
reverse_sorted
sorted
1perturbed
random

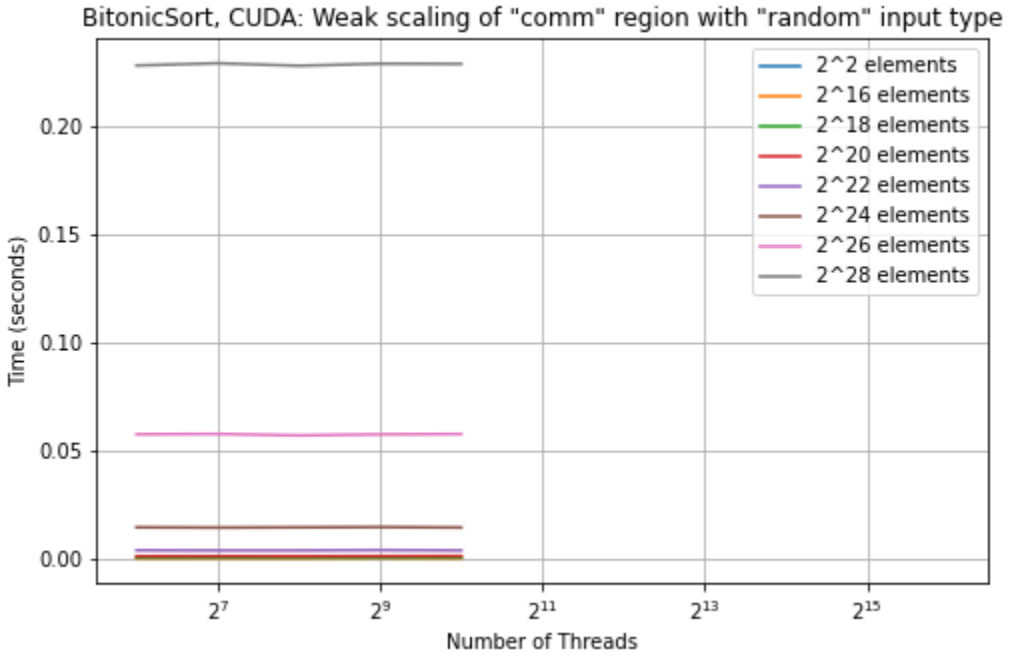

Bitonic Sort, CUDA: Strong scaling of "main" region with 2^22 elements
Time (seconds)
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
reverse_sorted
random
1perturbed
sorted

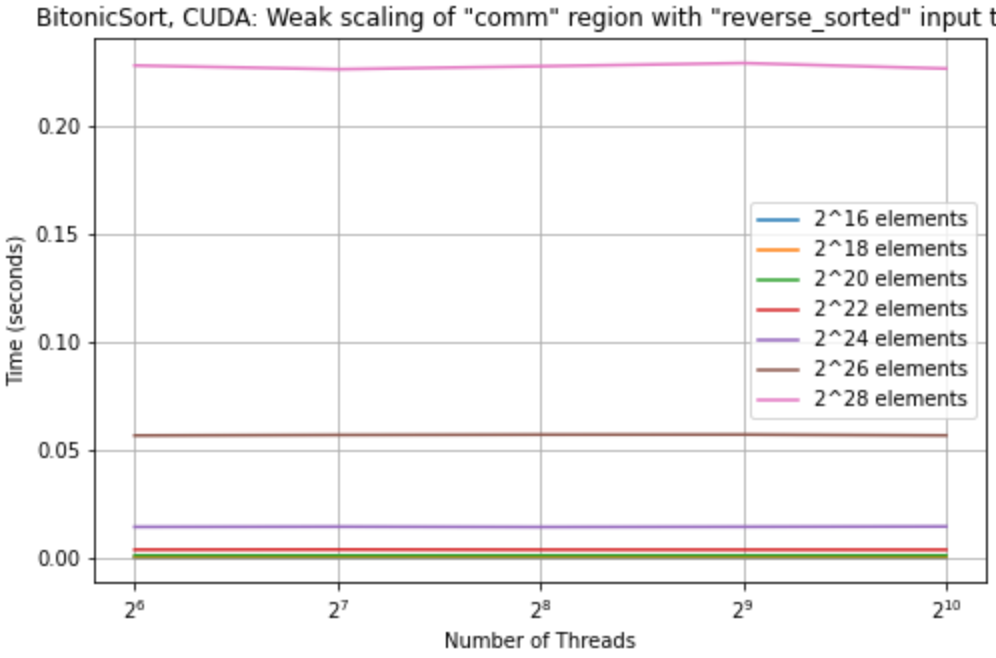

Bitonic Sort, CUDA: Strong scaling of "main" region with 2^24 elements
Time (seconds)
0.9
0.8
0.7
0.6
0.5
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
reverse_sorted
sorted
1perturbed
random

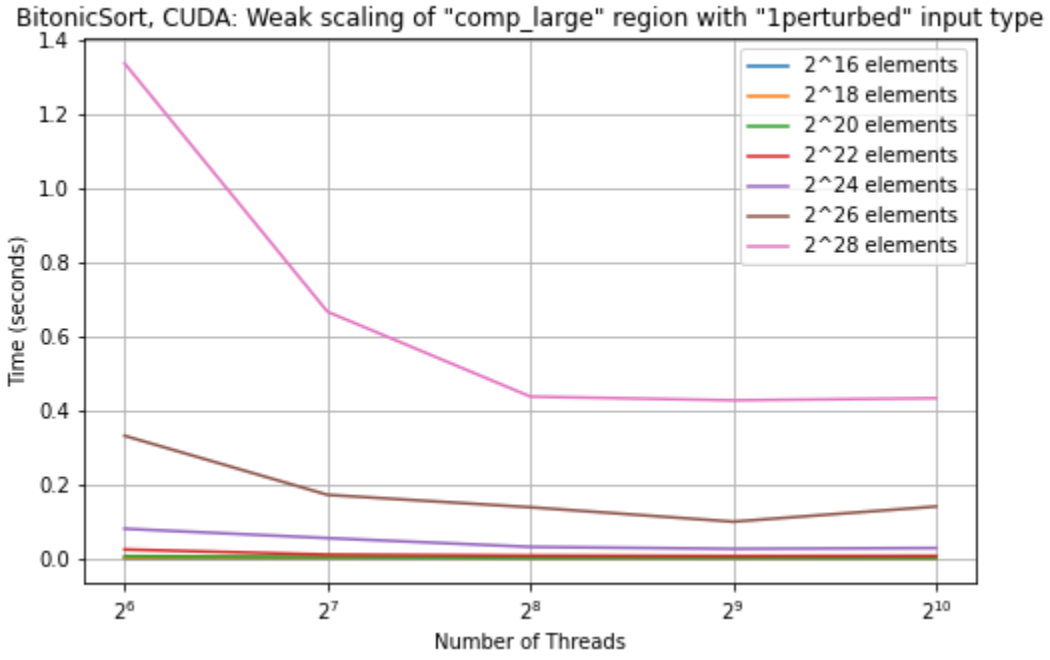

Bitonic Sort, CUDA: Strong scaling of "main" region with 2^26 elements
Time (seconds)
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
random
1perturbed
sorted
reverse_sorted
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

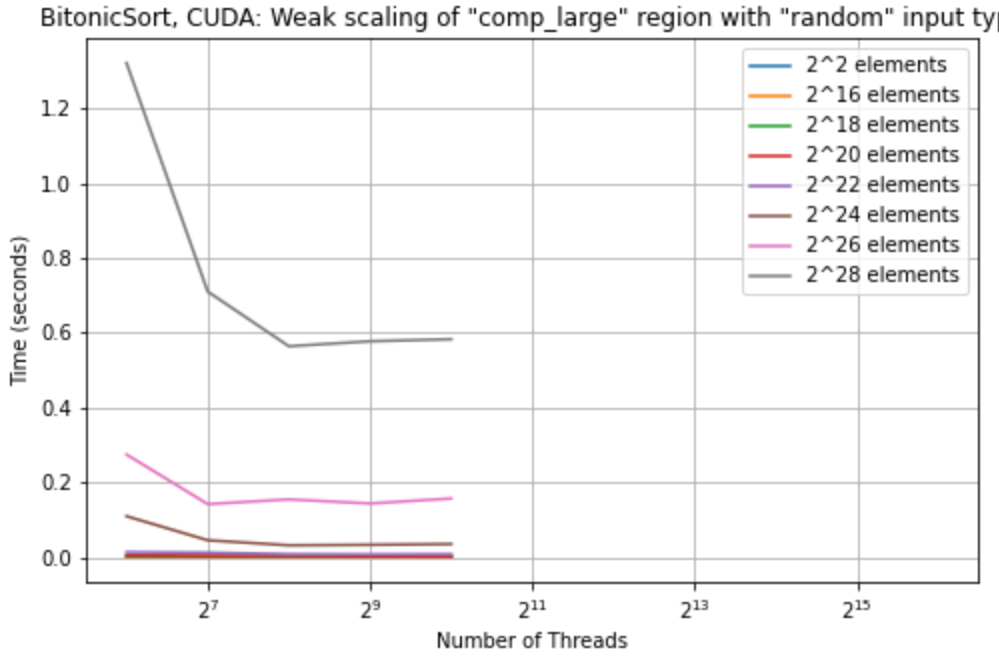

Bitonic Sort, CUDA: Strong scaling of "main" region with 2^28 elements
Time (seconds)
9
8
7
6
5
4
3
2
sorted
random
reverse_sorted
1perturbed
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

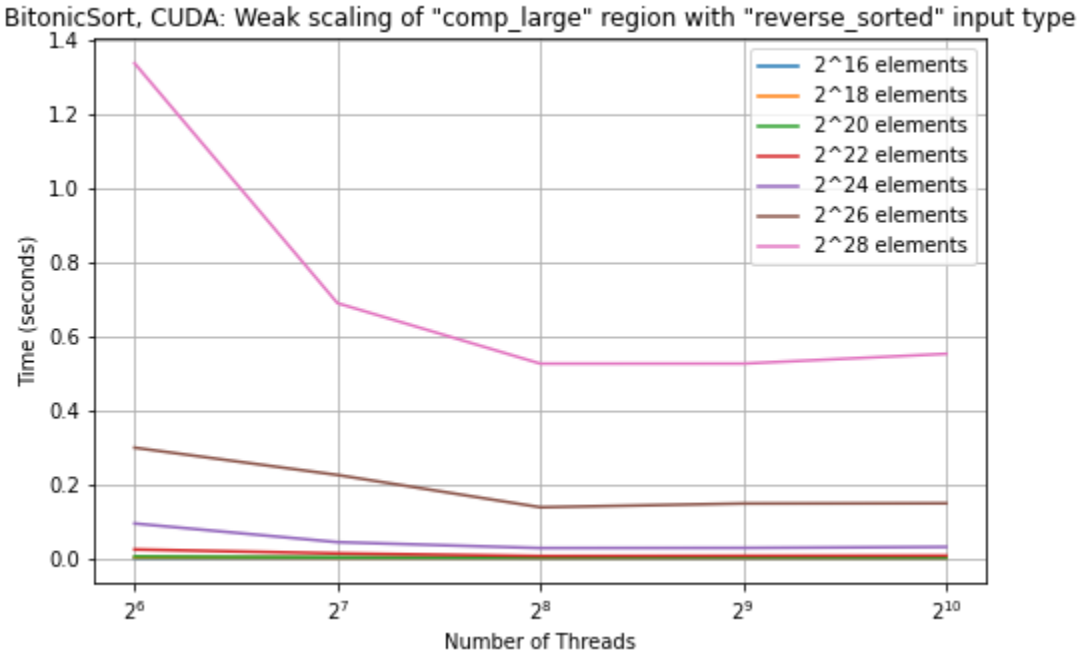

BitonicSort, CUDA: Weak scaling of "comm" region with "1perturbed" input type
Time (seconds)
0.20
0.15
0.10
0.05
0.00
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

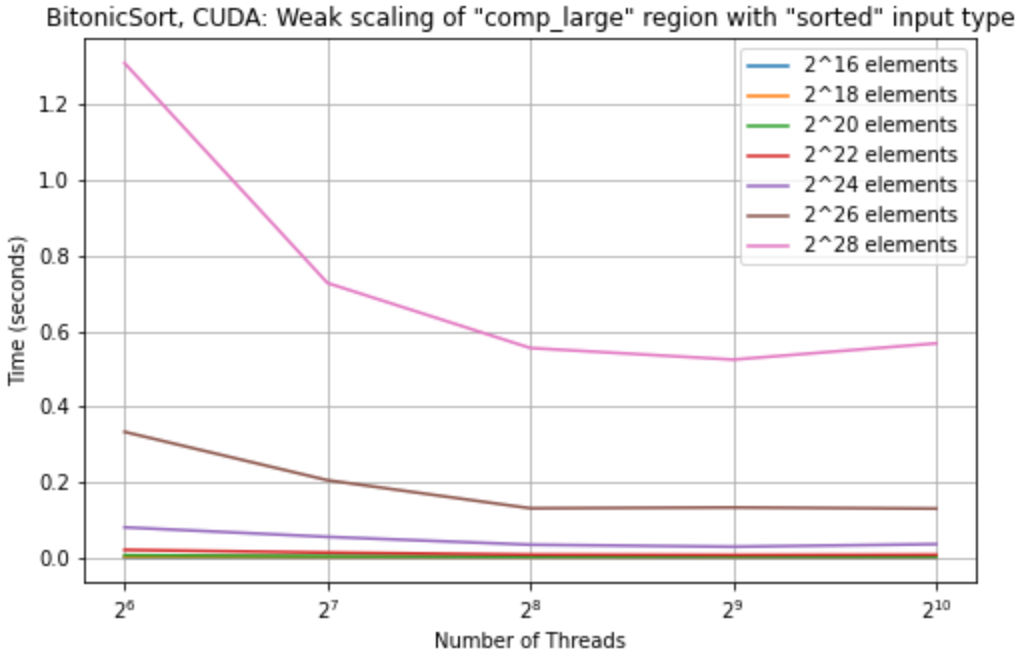

BitonicSort, CUDA: Weak scaling of "comm" region with "random" input type
Time (seconds)
0.20
0.15
0.10
0.05
0.00
$2^7$
$2^9$
$2^{11}$
$2^{13}$
$2^{15}$
Number of Threads
2^2 elements
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

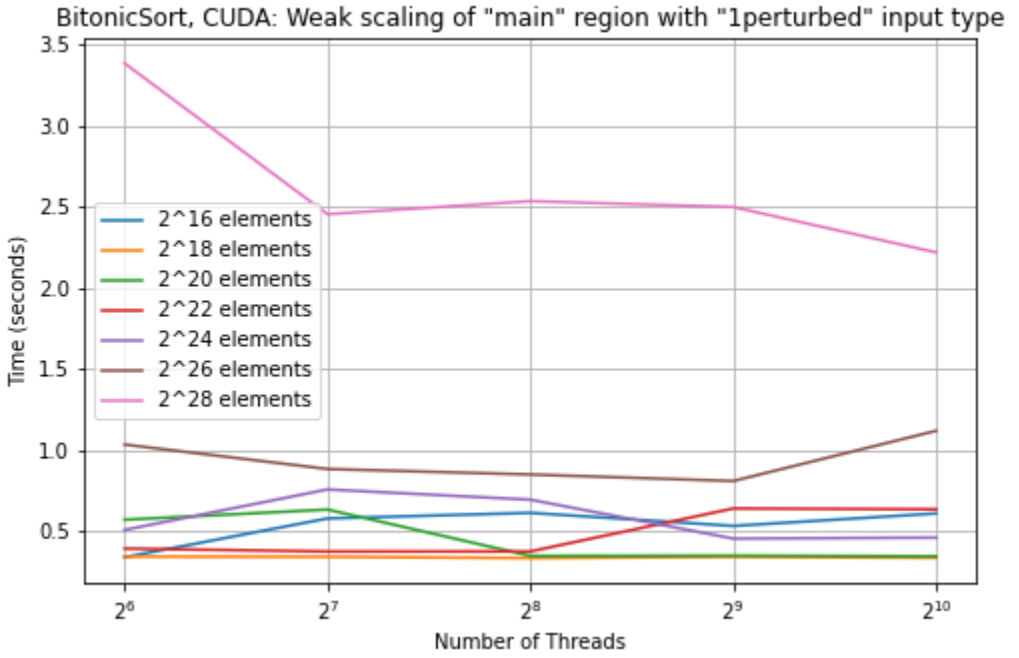

BitonicSort, CUDA: Weak scaling of "comm" region with "reverse_sorted" input type
Time (seconds)
0.20
0.15
0.10
0.05
0.00
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

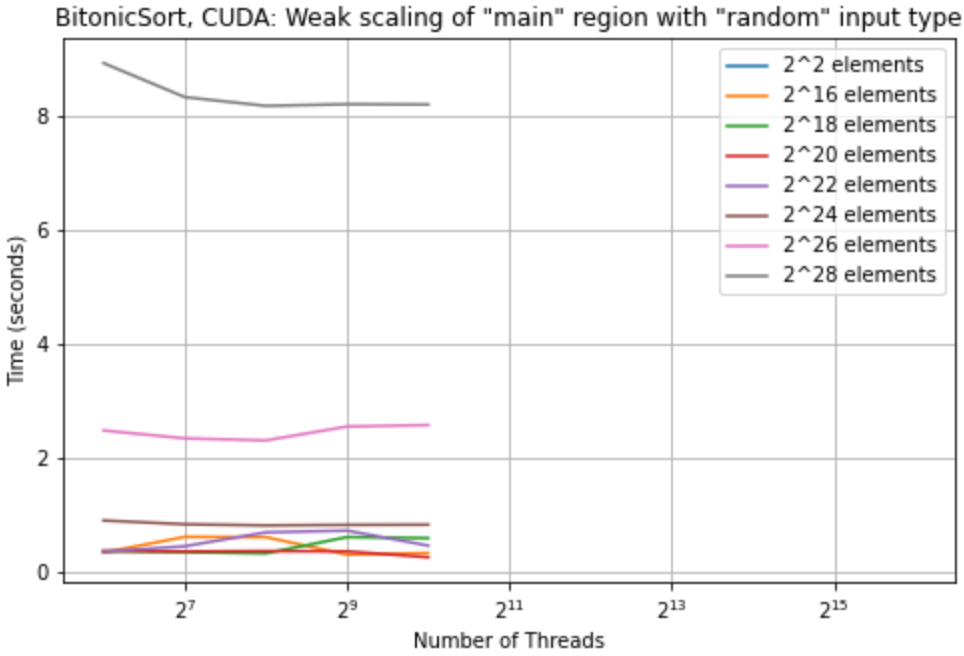

BitonicSort, CUDA: Weak scaling of "comp_large" region with "1perturbed" input type
Time (seconds)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

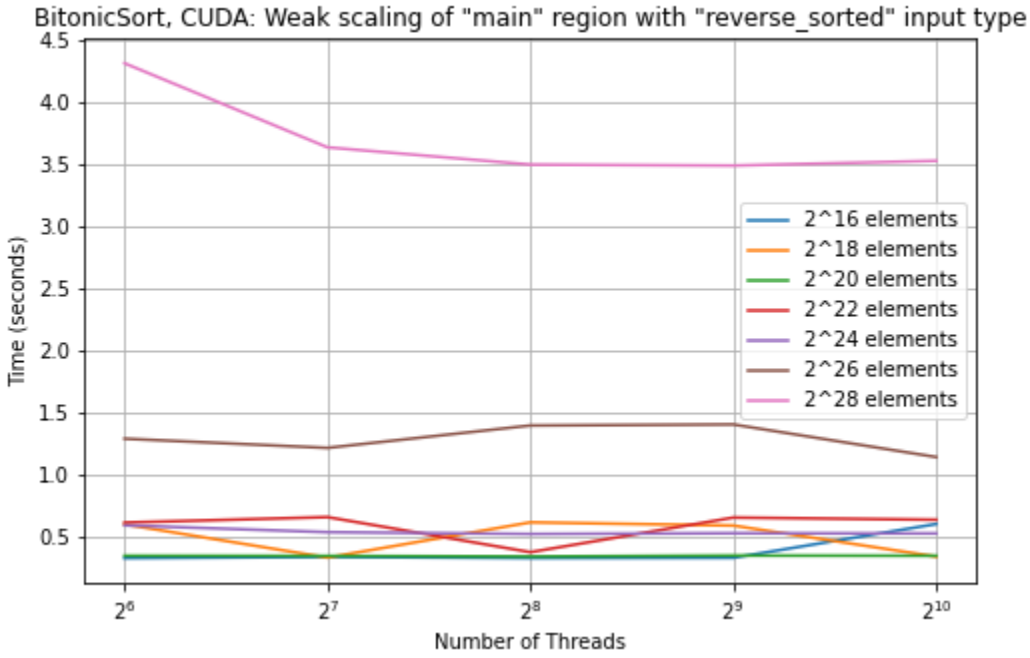

BitonicSort, CUDA: Weak scaling of "comp_large" region with "random" input type
2^2 elements
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
$2^7$
$2^9$
$2^{11}$
$2^{13}$
$2^{15}$
Number of Threads

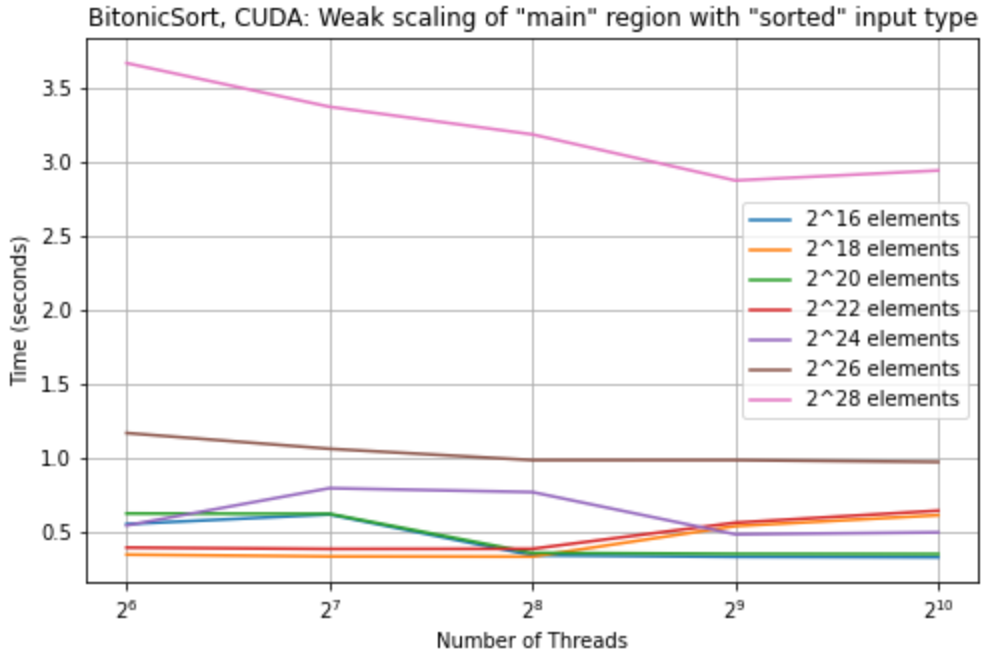

BitonicSort, CUDA: Weak scaling of "comp_large" region with "reverse_sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

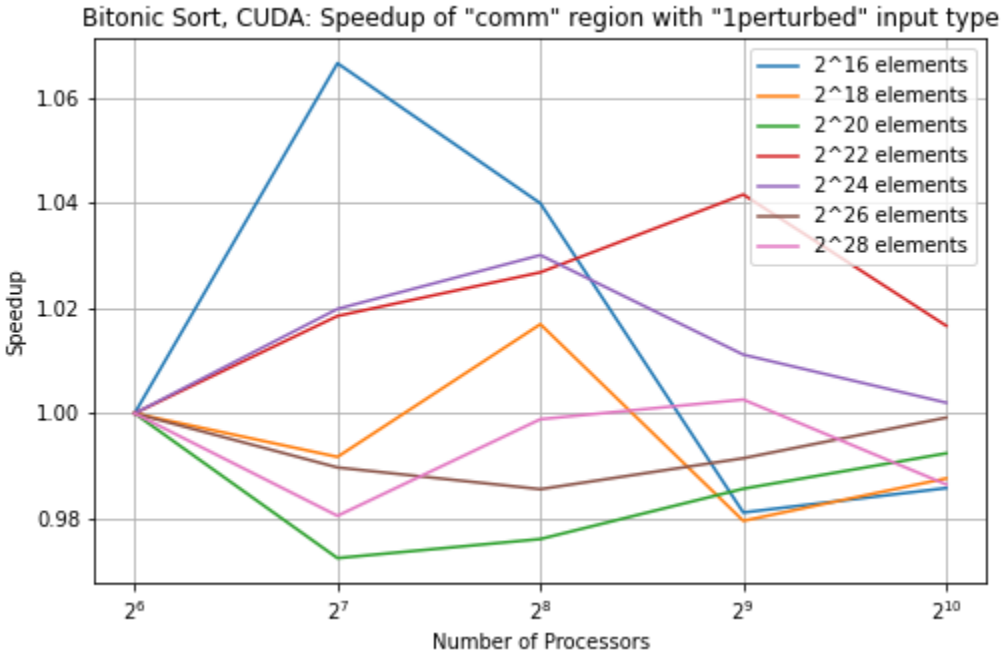

BitonicSort, CUDA: Weak scaling of "comp_large" region with "sorted" input type
Time (seconds)
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

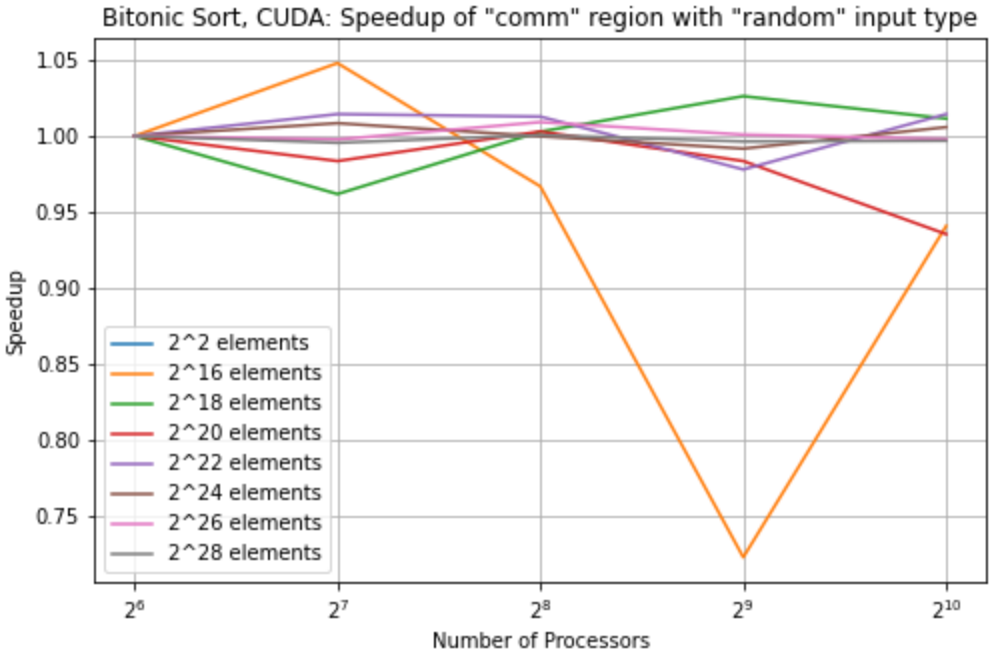

BitonicSort, CUDA: Weak scaling of "main" region with "1perturbed" input type
Time (seconds)
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

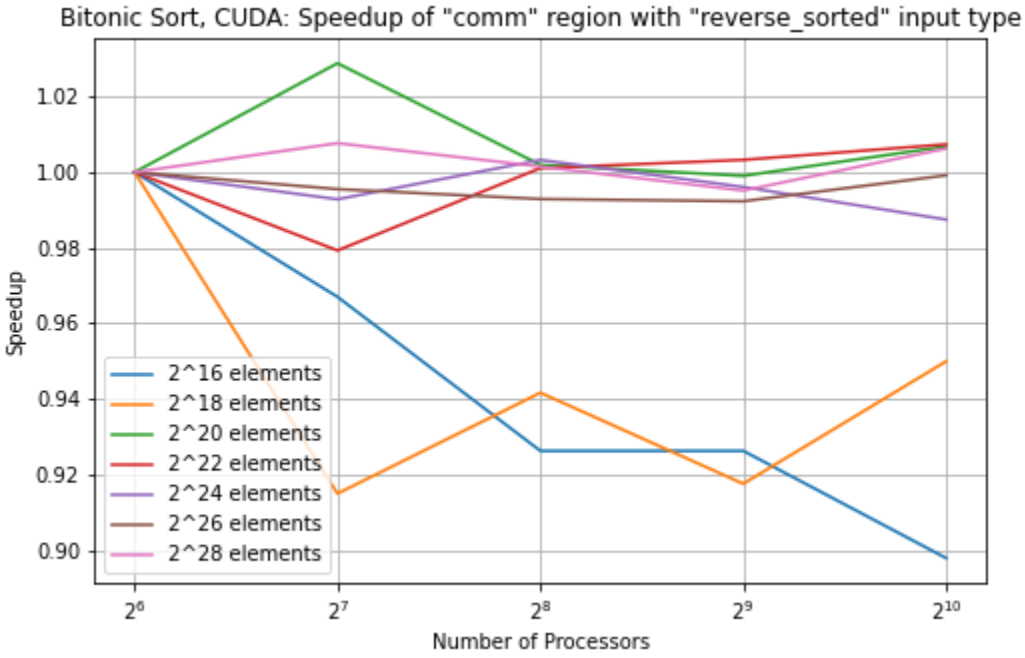

BitonicSort, CUDA: Weak scaling of "main" region with "random" input type
Time (seconds)
Number of Threads
2^2 elements
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

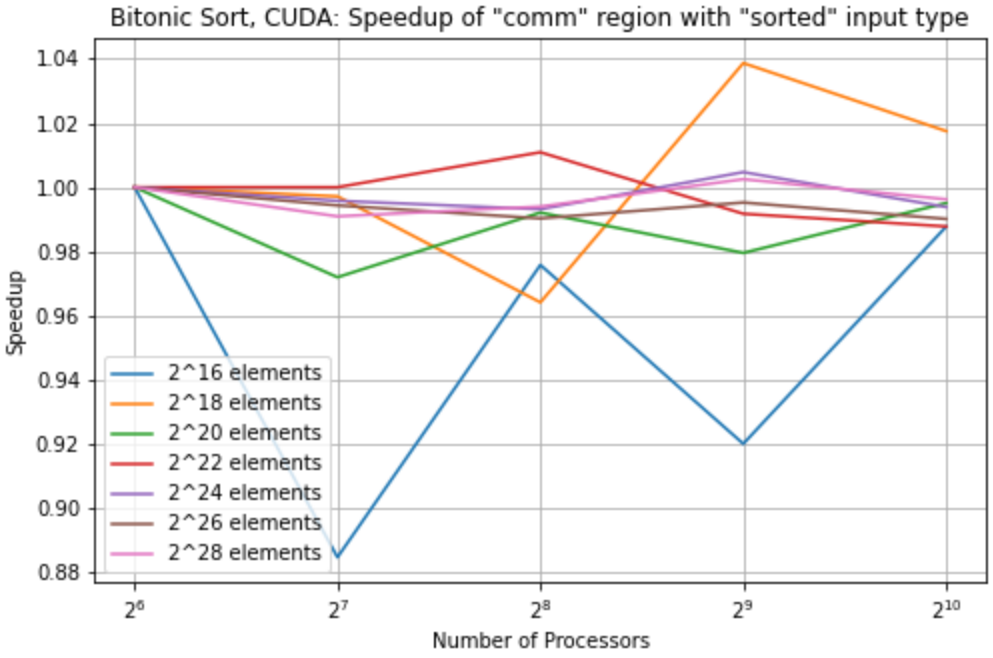

BitonicSort, CUDA: Weak scaling of "main" region with "reverse_sorted" input type
Time (seconds)
Number of Threads
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

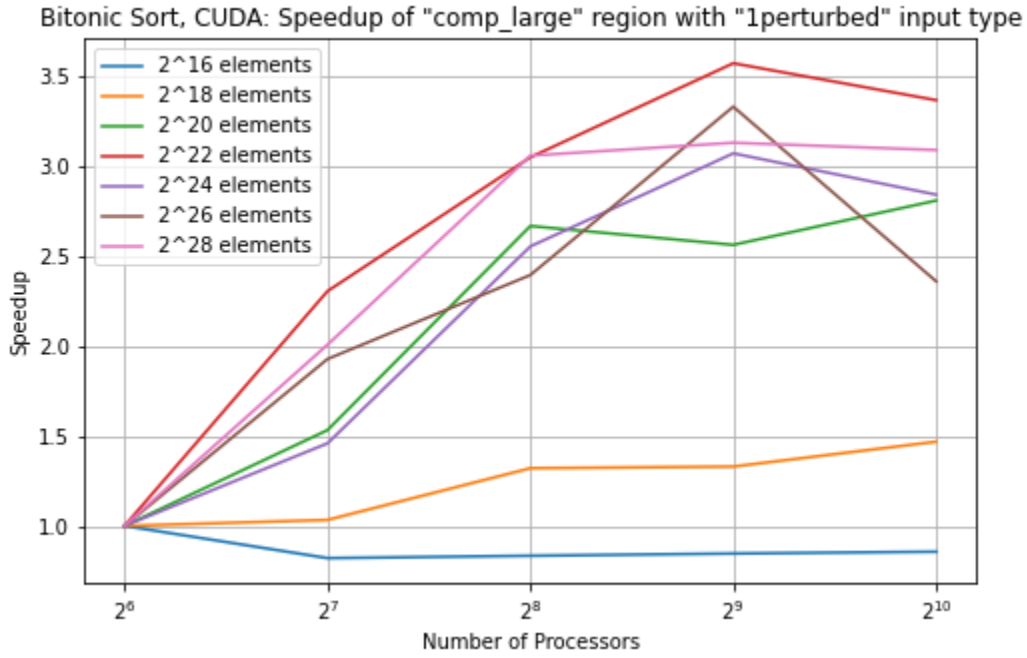

BitonicSort, CUDA: Weak scaling of "main" region with "sorted" input type
Time (seconds)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Threads

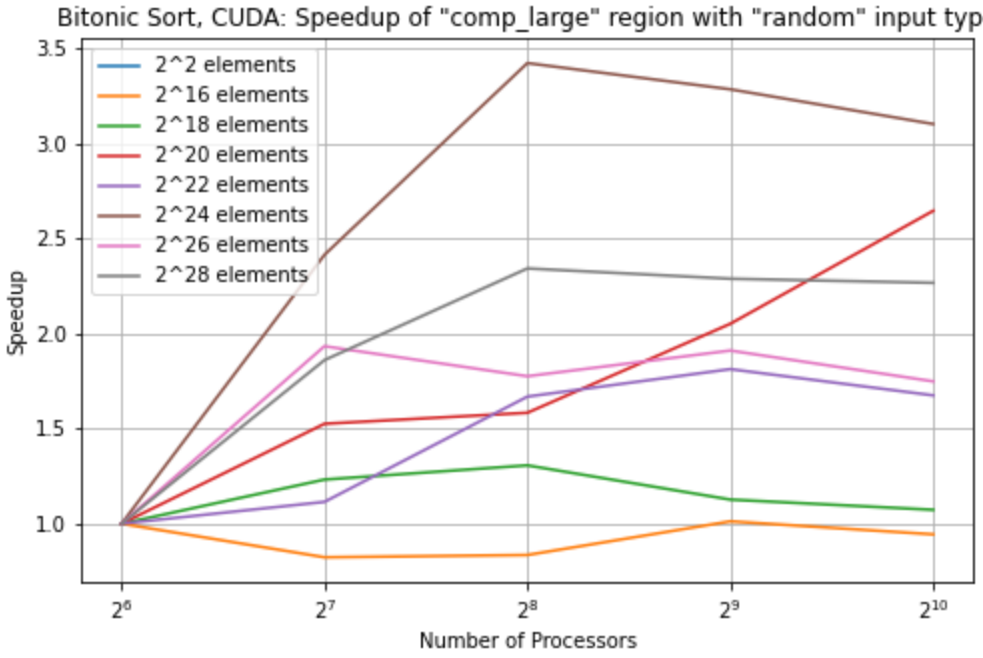

Bitonic Sort, CUDA: Speedup of "comm" region with "1perturbed" input type
Speedup
1.06
1.04
1.02
1.00
0.98
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Processors

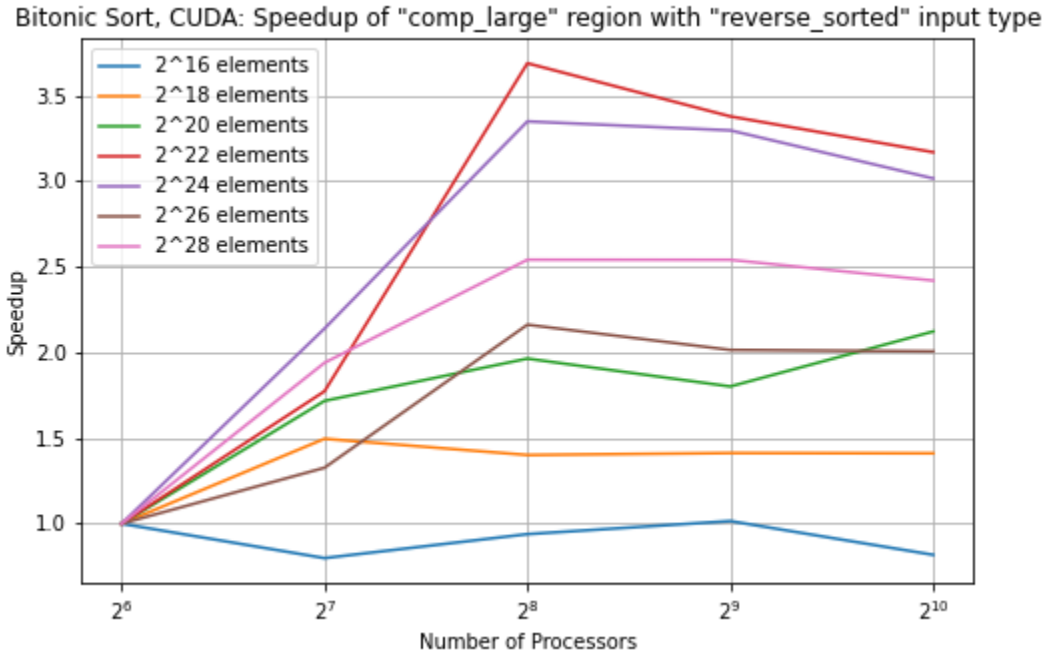

Bitonic Sort, CUDA: Speedup of "comm" region with "random" input type
Speedup
Number of Processors
1.05
1.00
0.95
0.90
0.85
0.80
0.75
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
2^2 elements
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

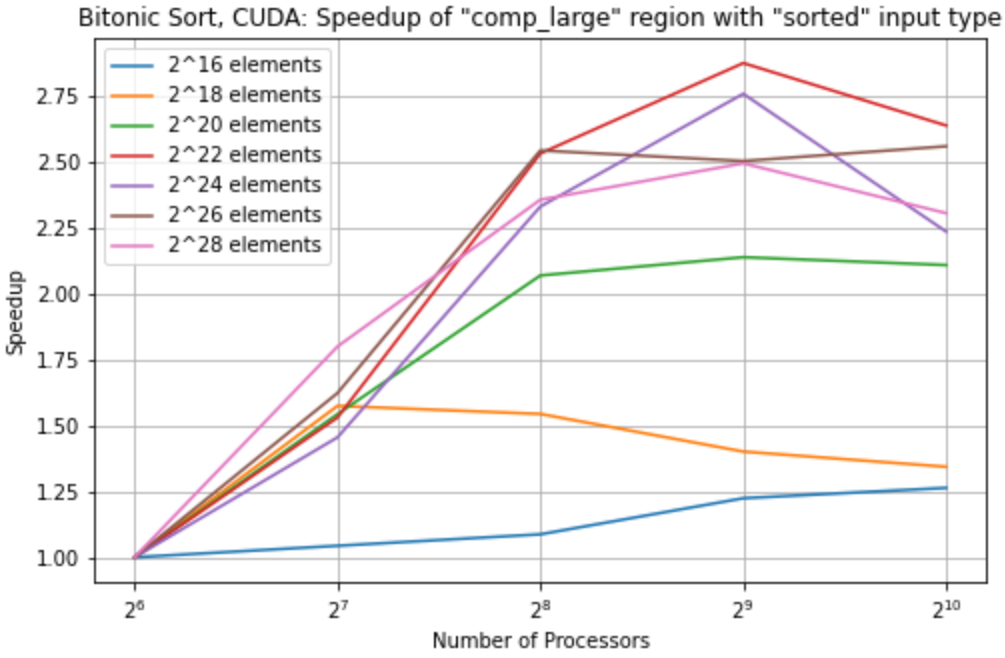

Bitonic Sort, CUDA: Speedup of "comm" region with "reverse_sorted" input type
Speedup
Number of Processors
1.02
1.00
0.98
0.96
0.94
0.92
0.90
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

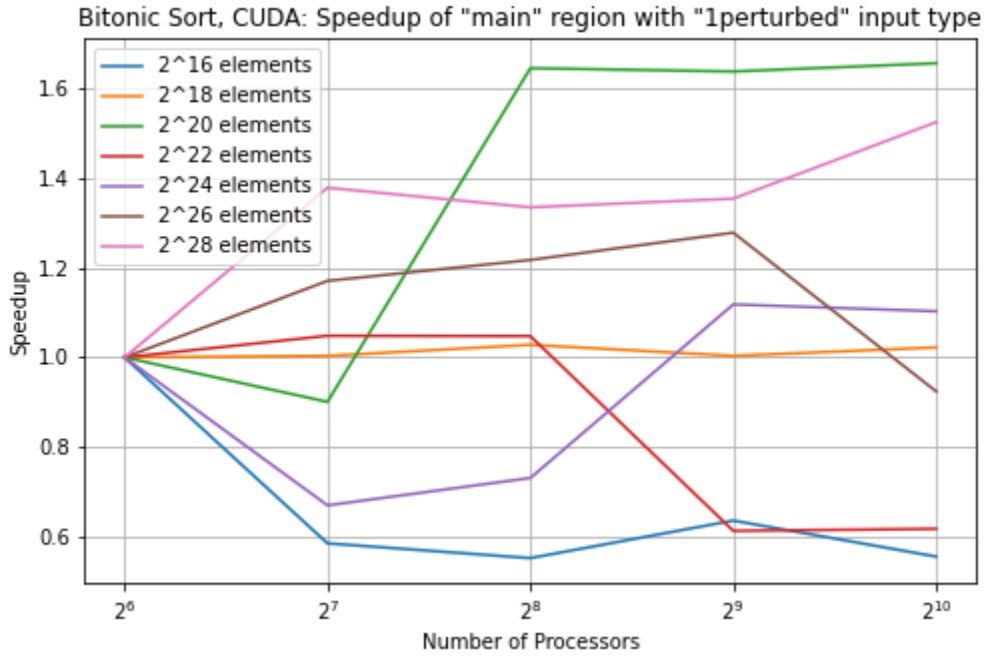

Bitonic Sort, CUDA: Speedup of "comm" region with "sorted" input type
Speedup
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

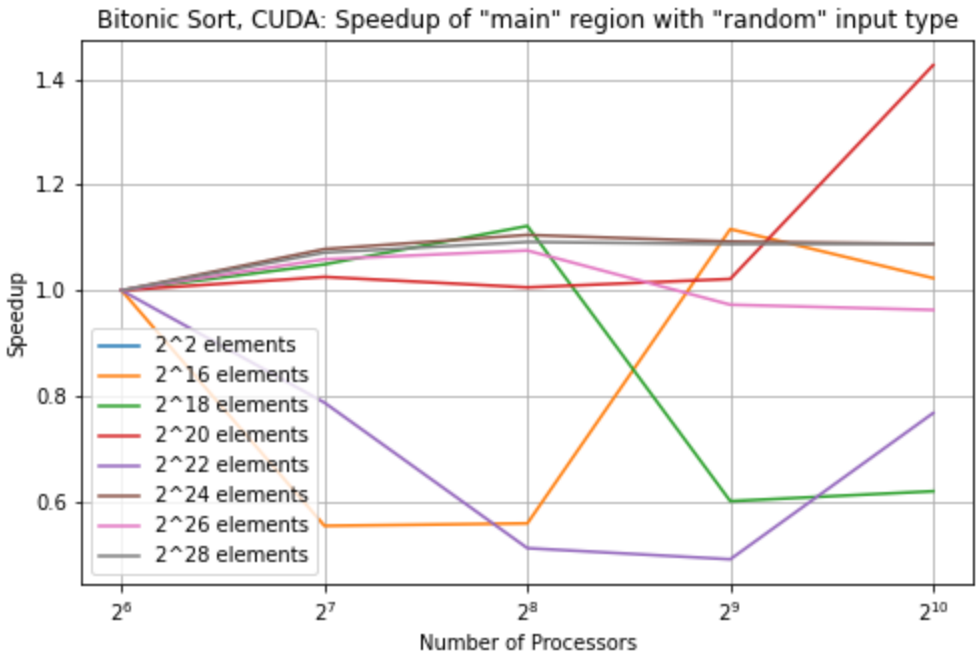

Bitonic Sort, CUDA: Speedup of "comp_large" region with "1perturbed" input type
Speedup
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

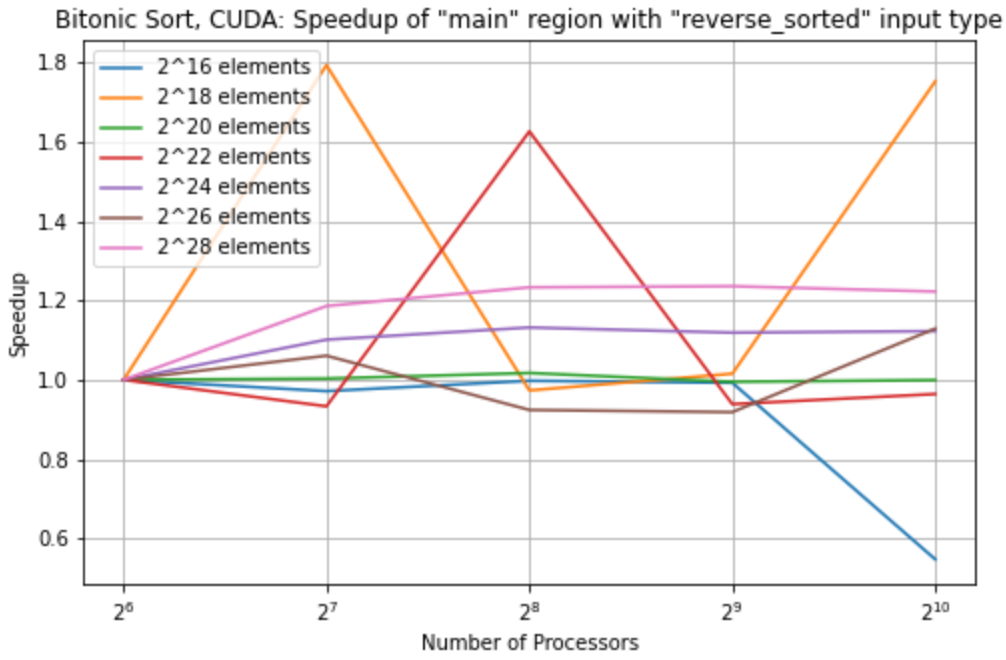

Bitonic Sort, CUDA: Speedup of "comp_large" region with "random" input type
2^2 elements
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
3.5
3.0
2.5
2.0
1.5
1.0
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Processors

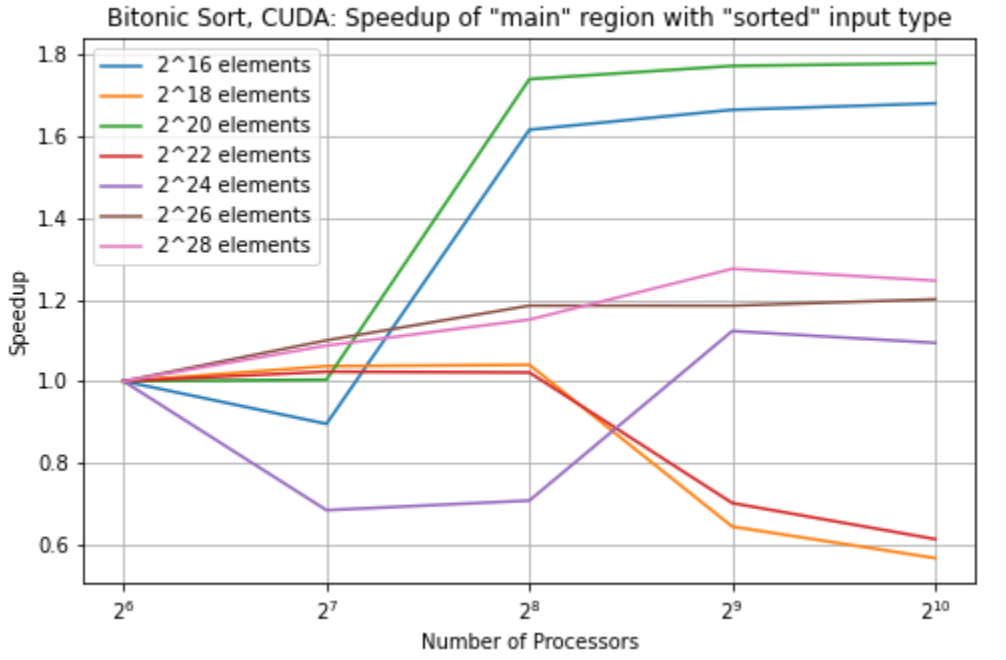

Bitonic Sort, CUDA: Speedup of "comp_large" region with "reverse_sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
3.5
3.0
2.5
2.0
1.5
1.0
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Processors


Bitonic Sort, CUDA: Speedup of "comp_large" region with "sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Processors


Bitonic Sort, CUDA: Speedup of "main" region with "1perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
1.6
1.4
1.2
1.0
0.8
0.6
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Processors


Bitonic Sort, CUDA: Speedup of "main" region with "random" input type
Speedup
1.4
1.2
1.0
0.8
0.6
2^2 elements
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
$2^6$
$2^7$
$2^8$
$2^9$
$2^{10}$
Number of Processors

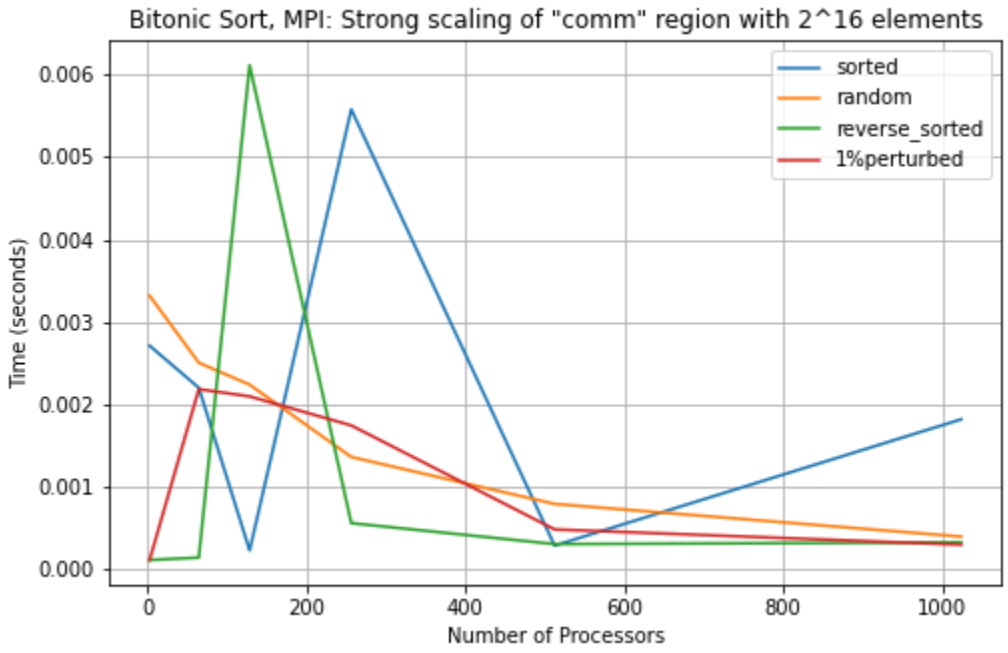


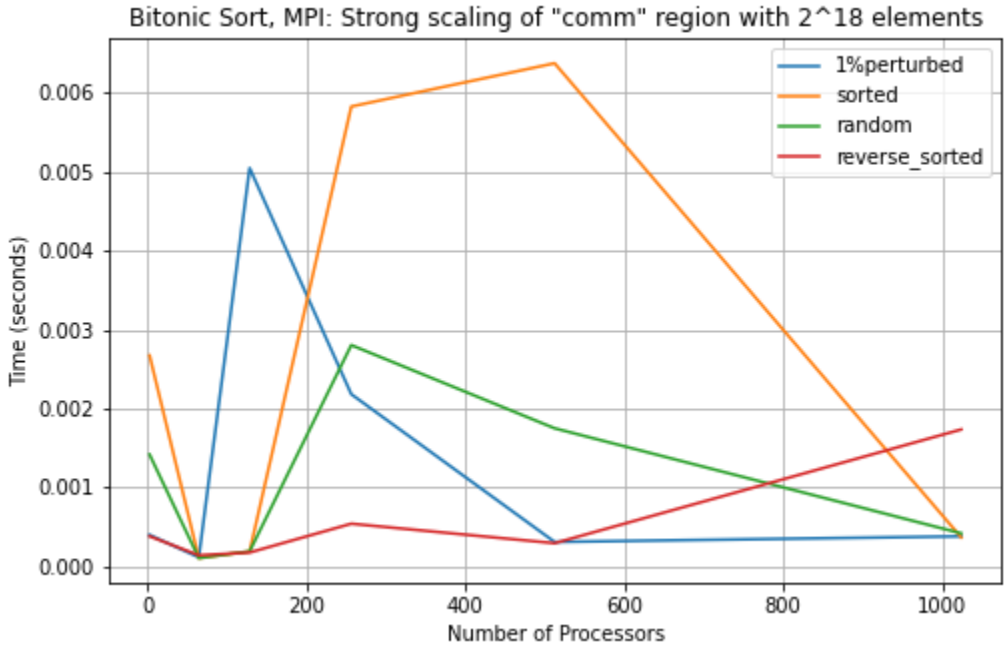



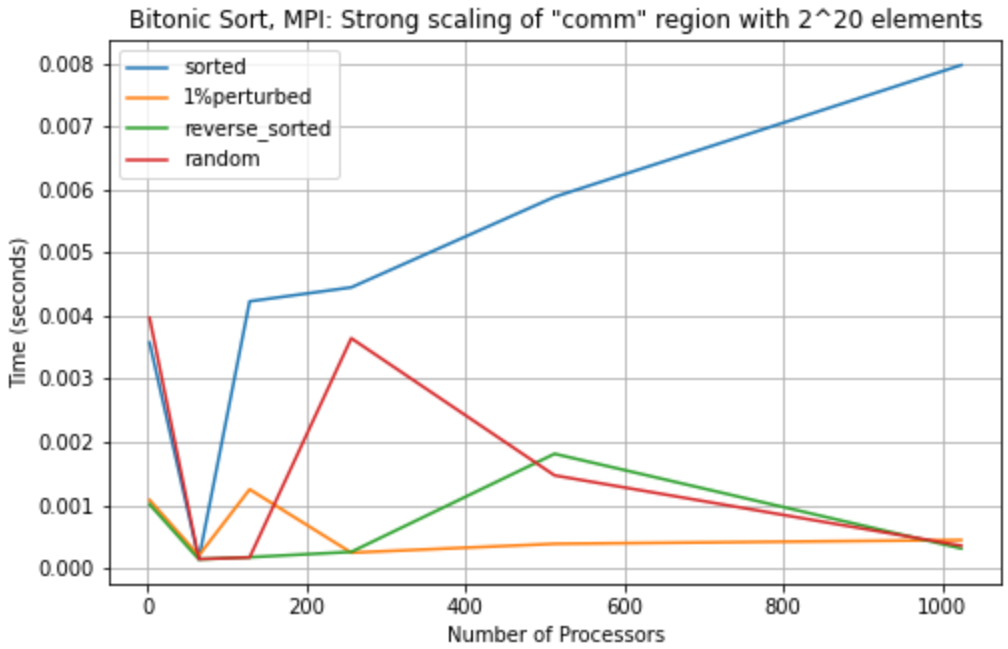

Throughout all of the implementations, we can expect an increase in the computational and communication time as the size of the array increase. We can also expect bitonic sort parallelism to reach a point of diminishing returns as we increase the number of threads. Additionally, as we change the different array populations such as random, sorted, reverse sorted, and 1% perturped, we can expect slight differences in our

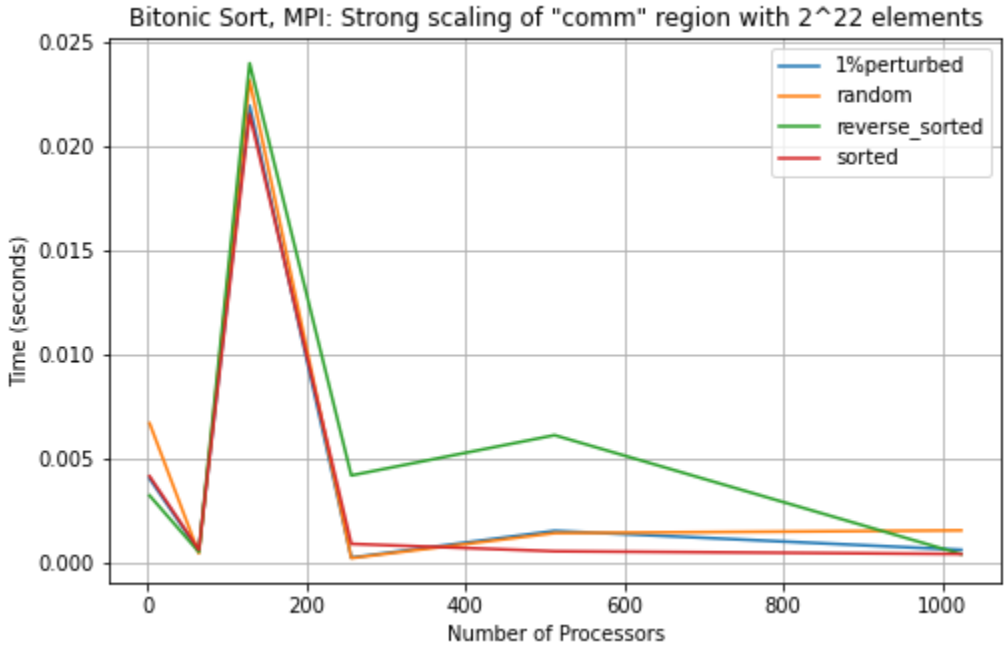
runtimes. For instance, a “sorted” array should have significantly less runtime than an unsorted array.

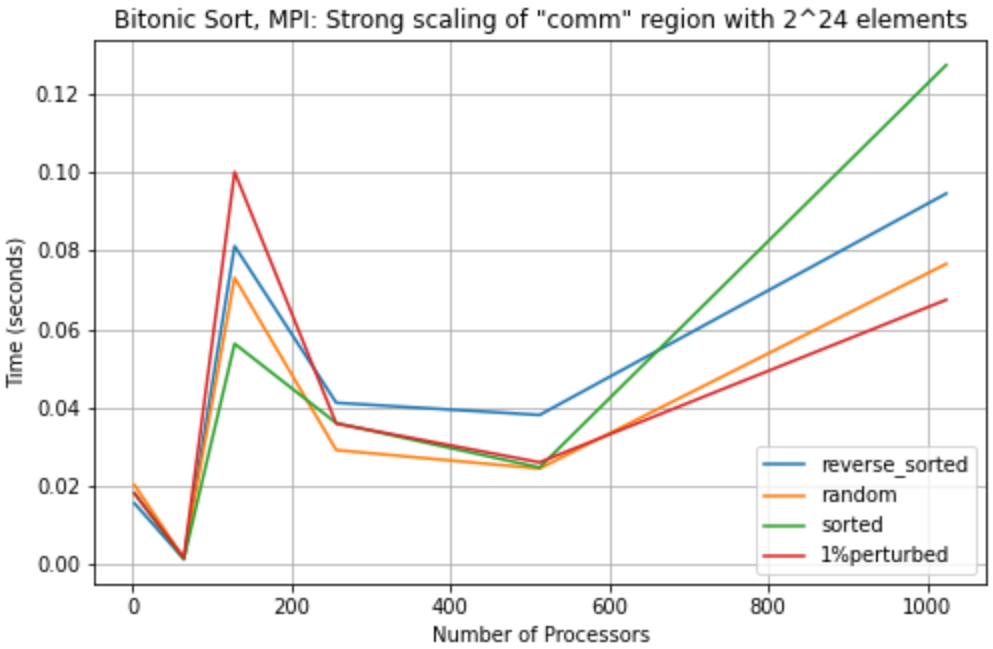
## Bitonic Sort MPI

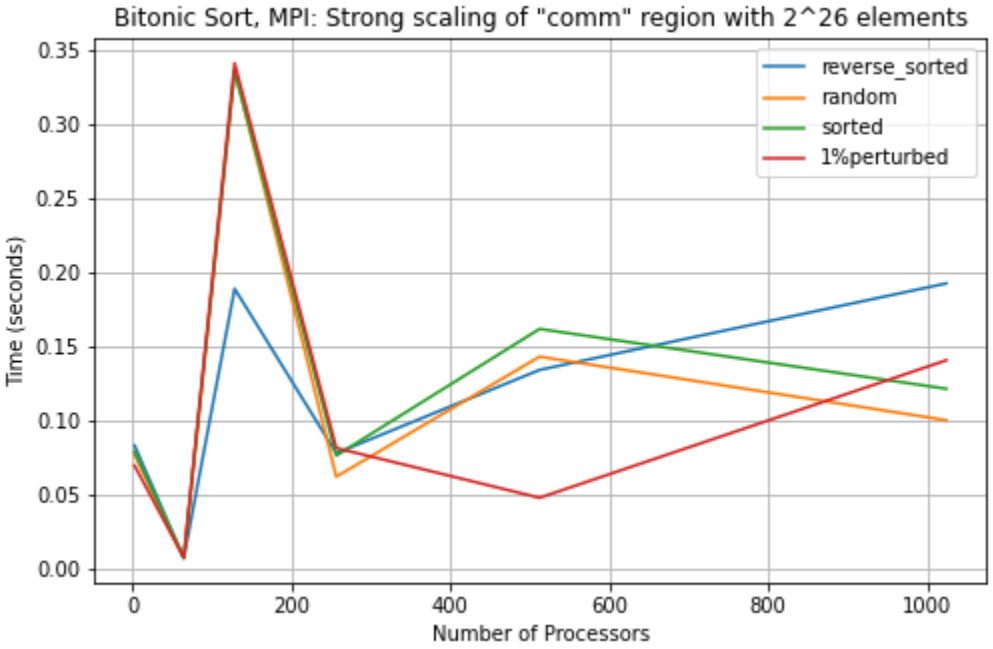


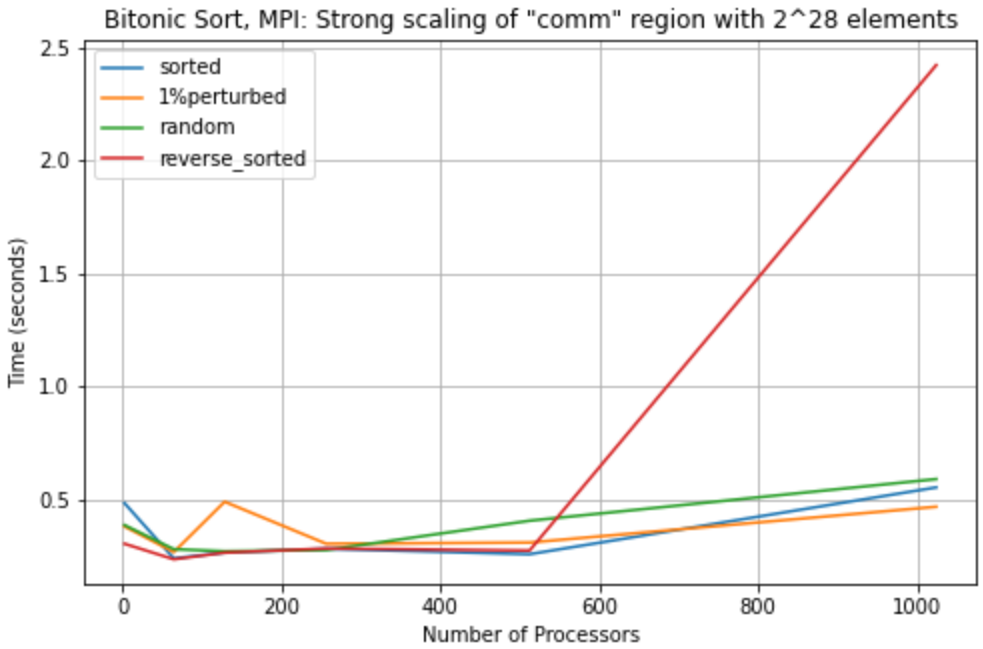



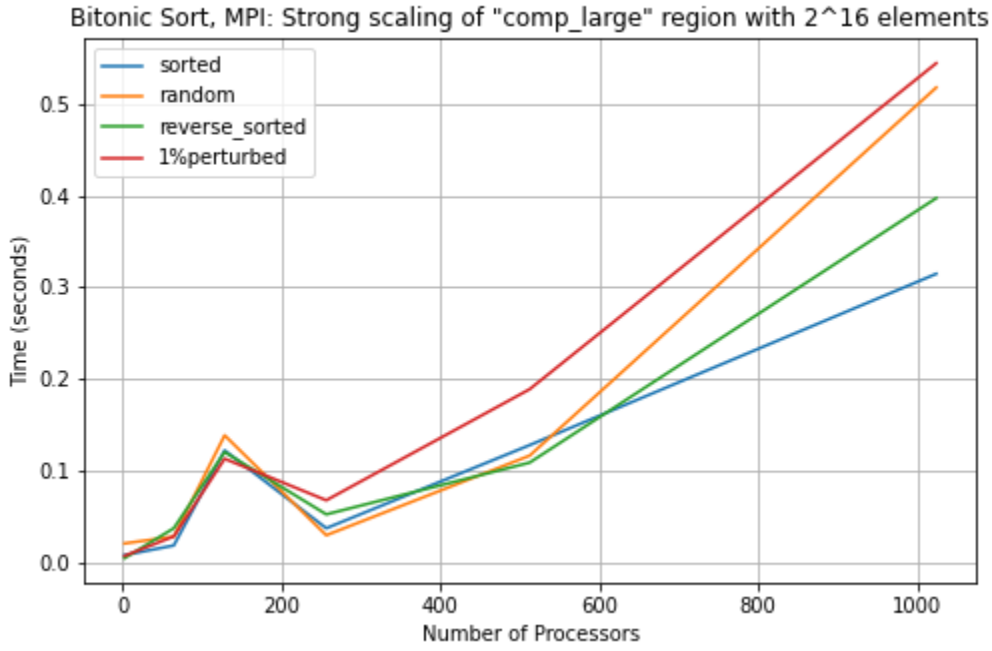

Bitonic Sort, MPI: Strong scaling of "comm" region with 2^20 elements
sorted
1%perturbed
reverse_sorted
random
Time (seconds)
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0
200
400
600
800
1000
Number of Processors

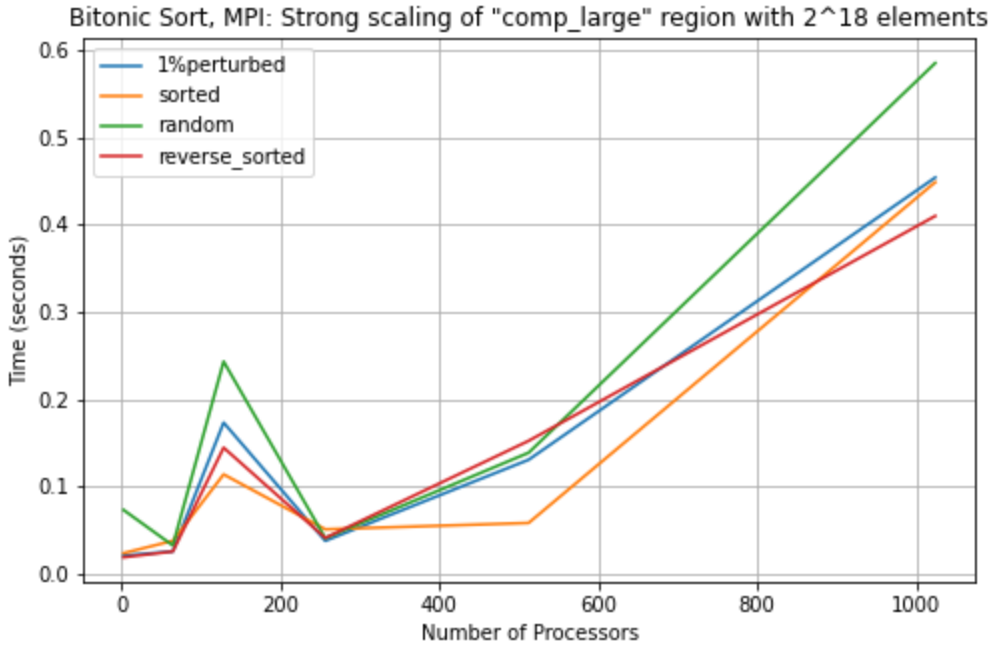

Bitonic Sort, MPI: Strong scaling of "comm" region with 2^22 elements
1%perturbed
random
reverse_sorted
sorted
Time (seconds)
0.000
0.005
0.010
0.015
0.020
0.025
0
200
400
600
800
1000
Number of Processors

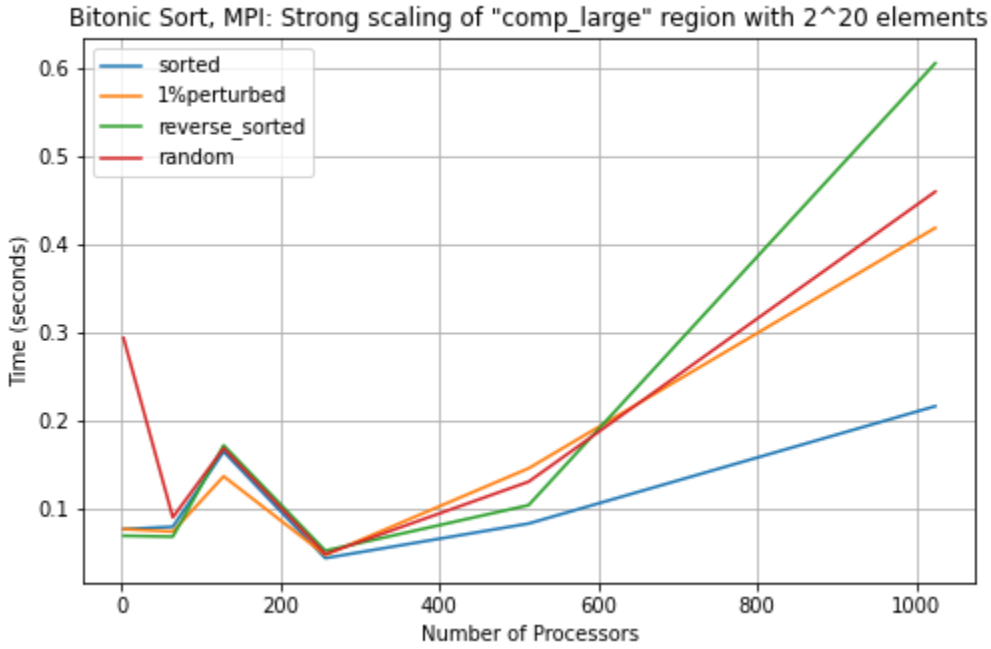

Bitonic Sort, MPI: Strong scaling of "comm" region with 2^24 elements
Time (seconds)
Number of Processors
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0
200
400
600
800
1000
reverse_sorted
random
sorted
1%perturbed

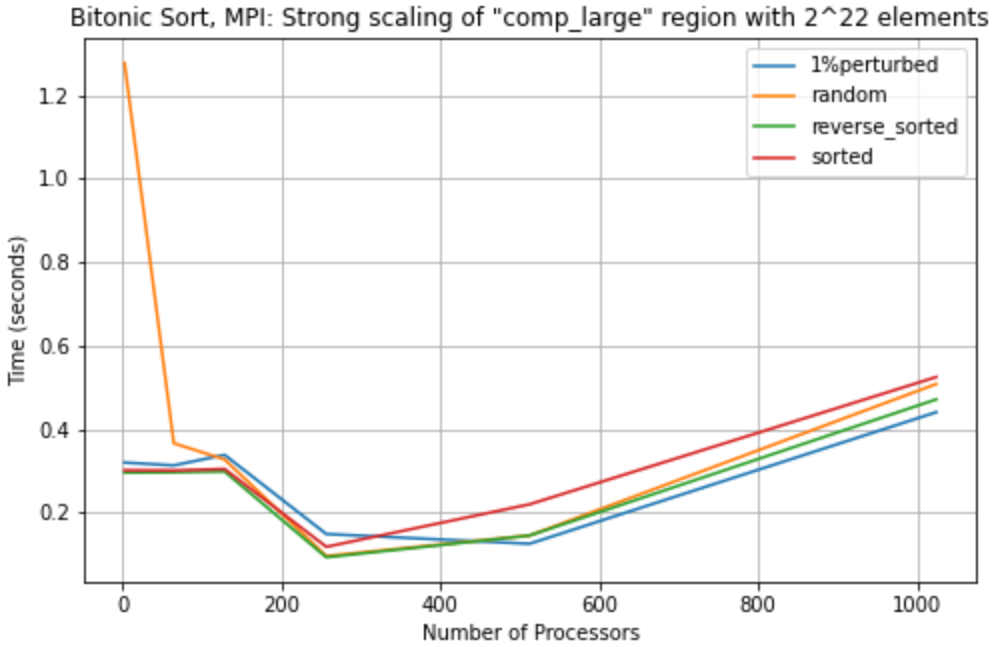

Bitonic Sort, MPI: Strong scaling of "comm" region with 2^26 elements
Time (seconds)
Number of Processors
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0
200
400
600
800
1000
reverse_sorted
random
sorted
1%perturbed

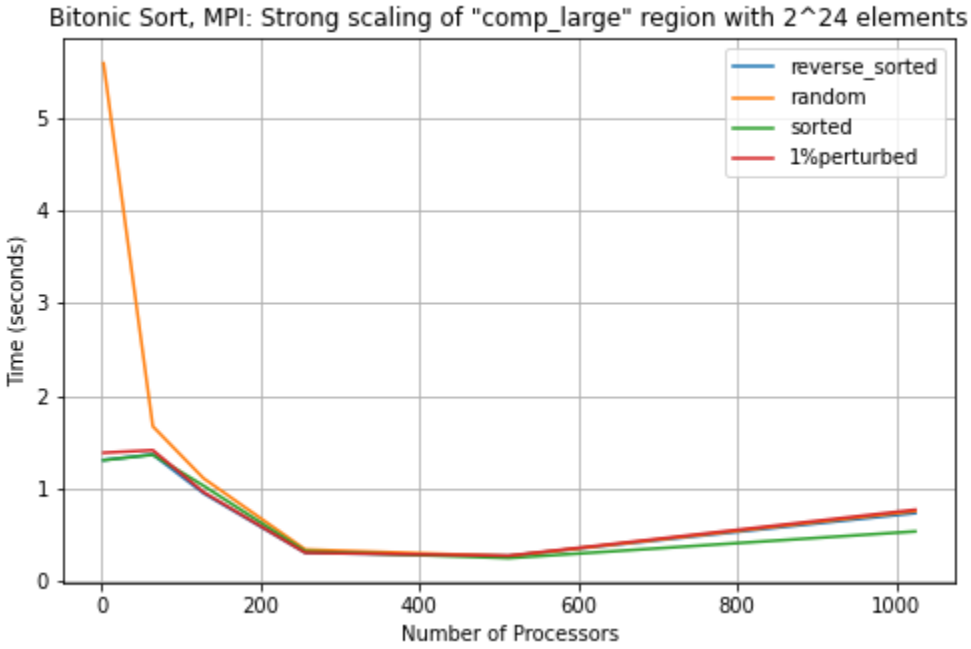

Bitonic Sort, MPI: Strong scaling of "comm" region with 2^28 elements
sorted
1%perturbed
random
reverse_sorted
Time (seconds)
Number of Processors
0
200
400
600
800
1000
0.5
1.0
1.5
2.0
2.5

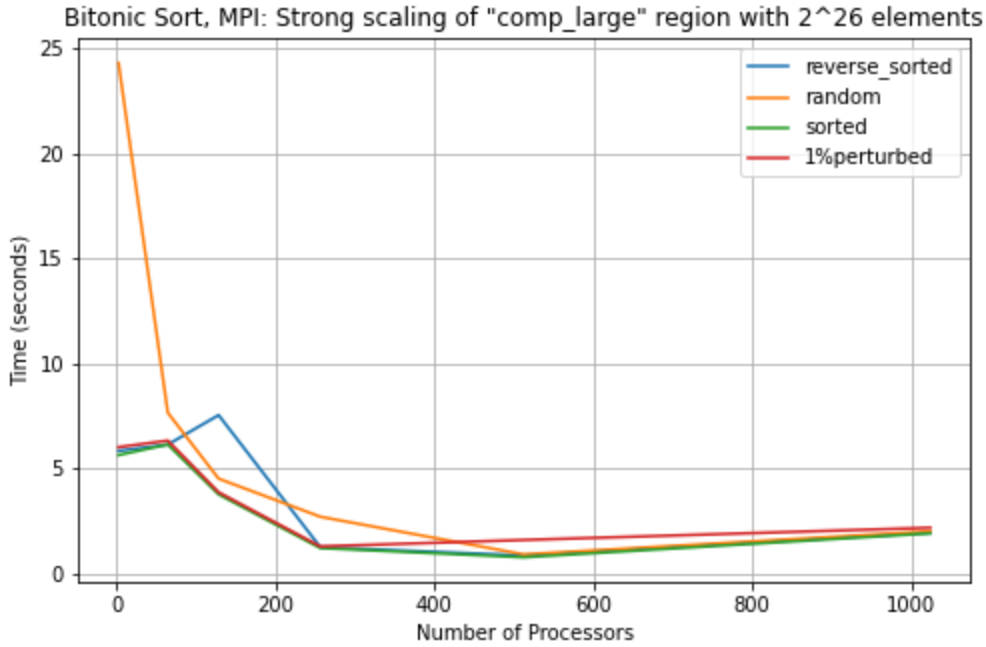

Bitonic Sort, MPI: Strong scaling of "comp_large" region with 2^16 elements
sorted
random
reverse_sorted
1%perturbed
Time (seconds)
Number of Processors
0
200
400
600
800
1000
0.0
0.1
0.2
0.3
0.4
0.5

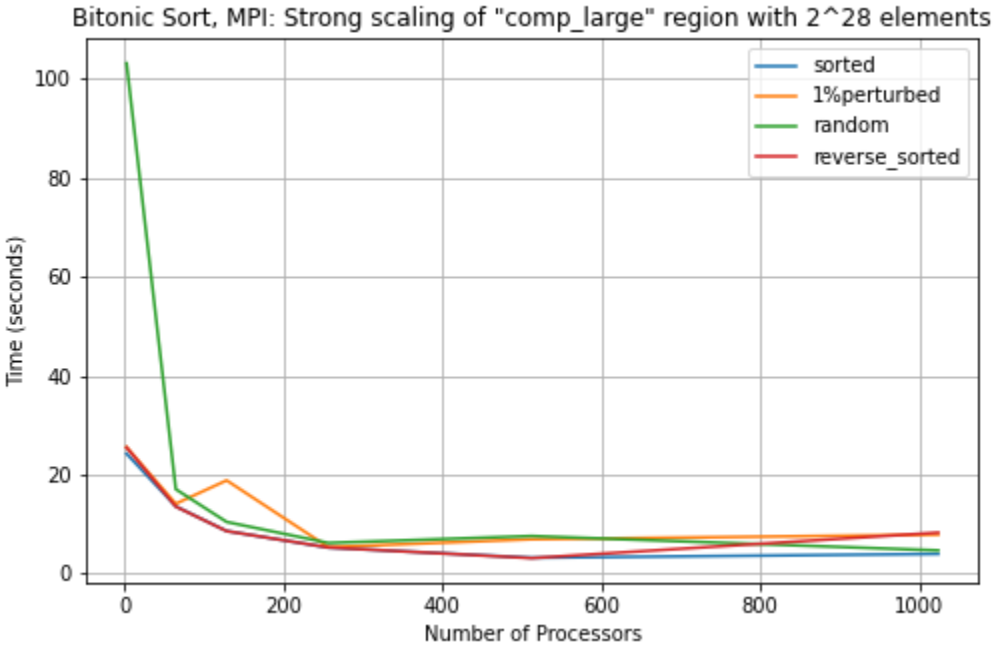

Bitonic Sort, MPI: Strong scaling of "comp_large" region with 2^18 elements
1%perturbed
sorted
random
reverse_sorted
Time (seconds)
Number of Processors
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
200
400
600
800
1000

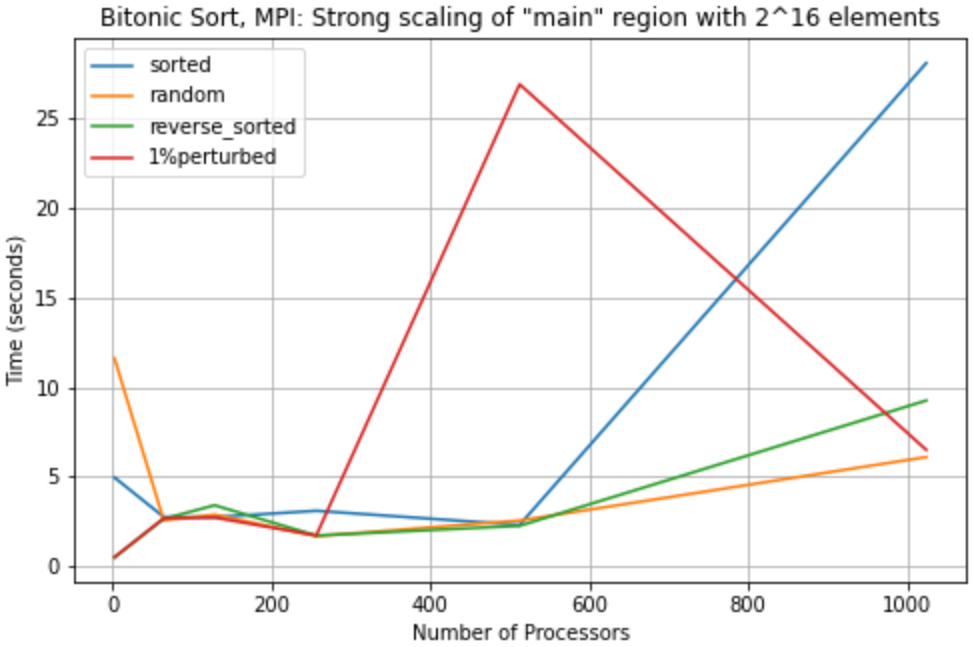

Bitonic Sort, MPI: Strong scaling of "comp_large" region with 2^20 elements
sorted
1%perturbed
reverse_sorted
random
Time (seconds)
Number of Processors
0.1
0.2
0.3
0.4
0.5
0.6
0
200
400
600
800
1000

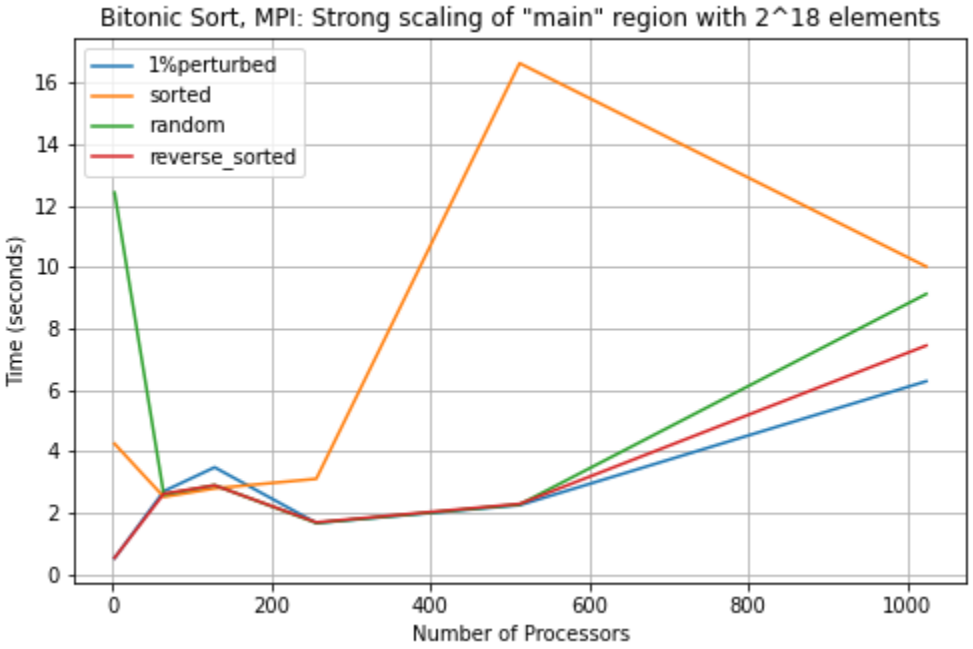

Bitonic Sort, MPI: Strong scaling of "comp_large" region with 2^22 elements
1%perturbed
random
reverse_sorted
sorted
Time (seconds)
Number of Processors
0
200
400
600
800
1000
0.2
0.4
0.6
0.8
1.0
1.2

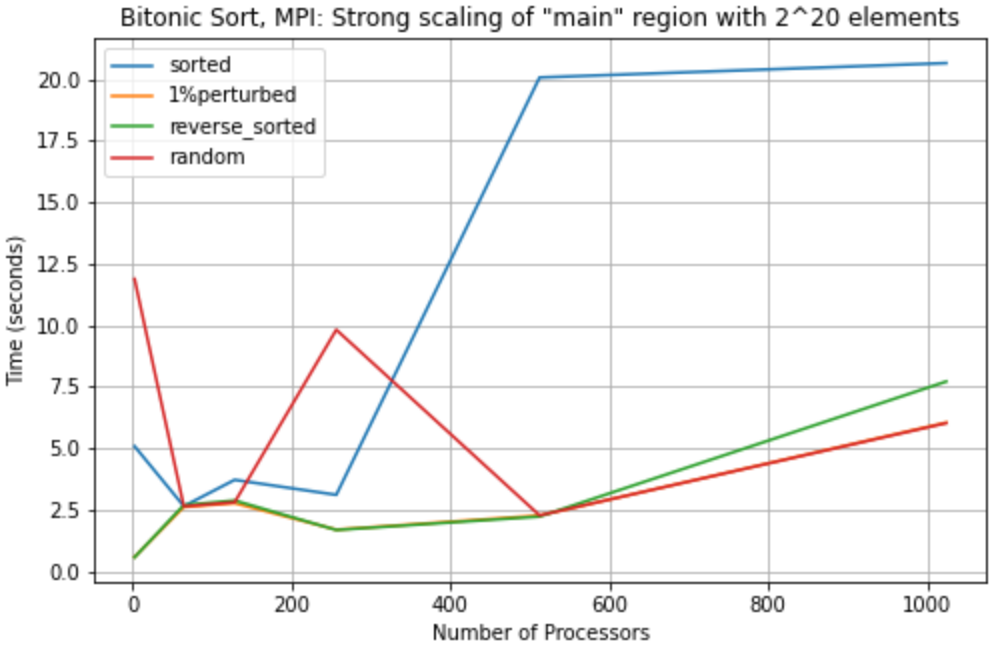

Bitonic Sort, MPI: Strong scaling of "comp_large" region with 2^24 elements
reverse_sorted
random
sorted
1%perturbed
Time (seconds)
Number of Processors
0
200
400
600
800
1000
0
1
2
3
4
5

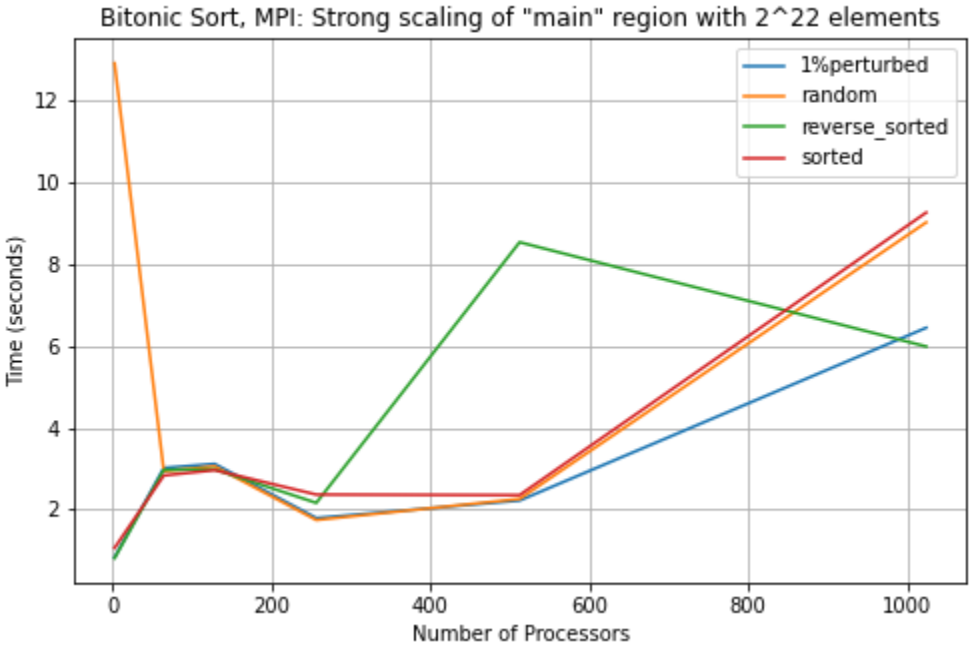

Bitonic Sort, MPI: Strong scaling of "comp_large" region with 2^26 elements
Time (seconds)
Number of Processors
reverse_sorted
random
sorted
1%perturbed
0
5
10
15
20
25
0
200
400
600
800
1000

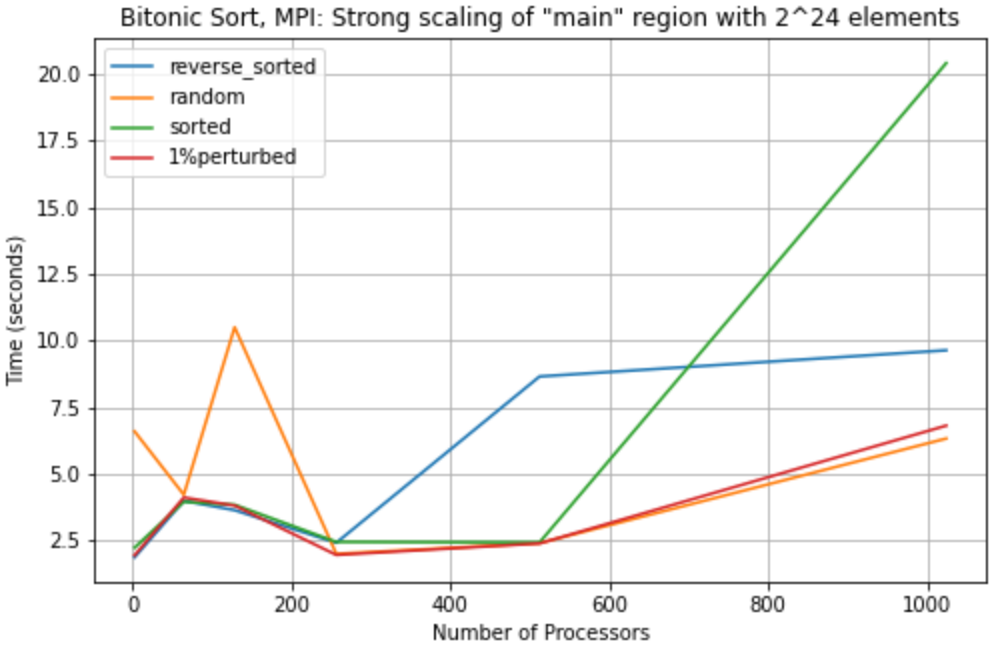

Bitonic Sort, MPI: Strong scaling of "comp_large" region with 2^28 elements
Time (seconds)
Number of Processors
sorted
1%perturbed
random
reverse_sorted
0
20
40
60
80
100
0
200
400
600
800
1000

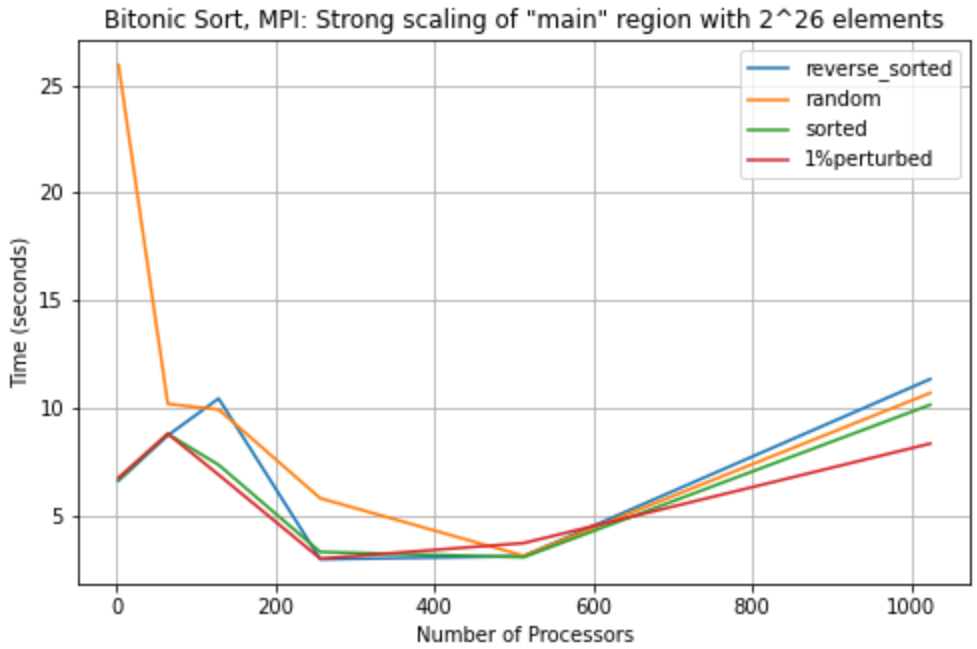

Bitonic Sort, MPI: Strong scaling of "main" region with 2^16 elements
sorted
random
reverse_sorted
1%perturbed
Time (seconds)
Number of Processors
0
5
10
15
20
25
0
200
400
600
800
1000

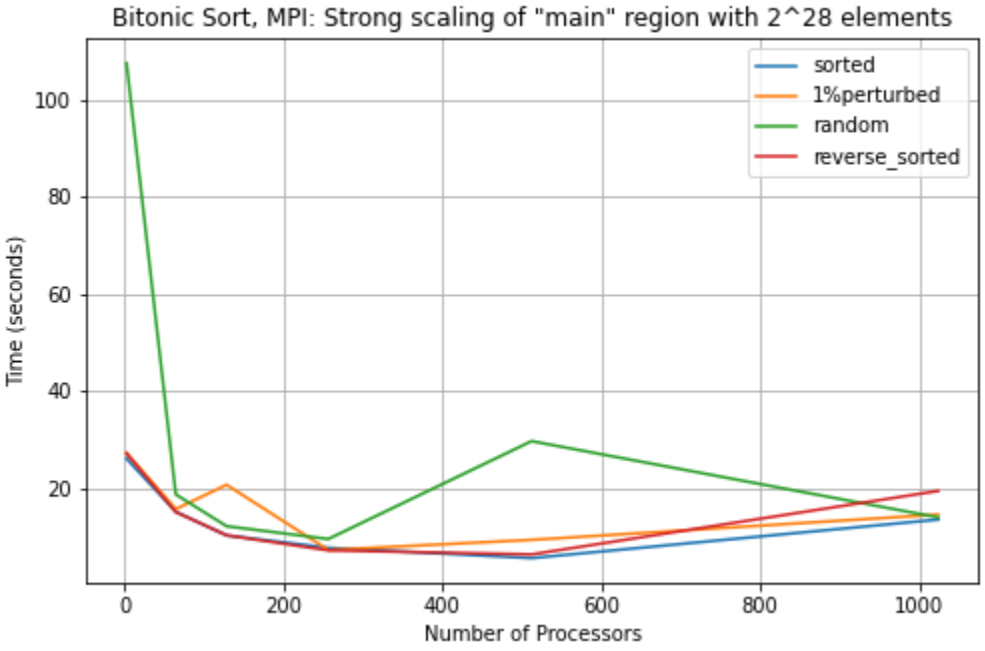

Bitonic Sort, MPI: Strong scaling of "main" region with 2^18 elements
1%perturbed
sorted
random
reverse_sorted
Time (seconds)
Number of Processors
0
2
4
6
8
10
12
14
16
0
200
400
600
800
1000

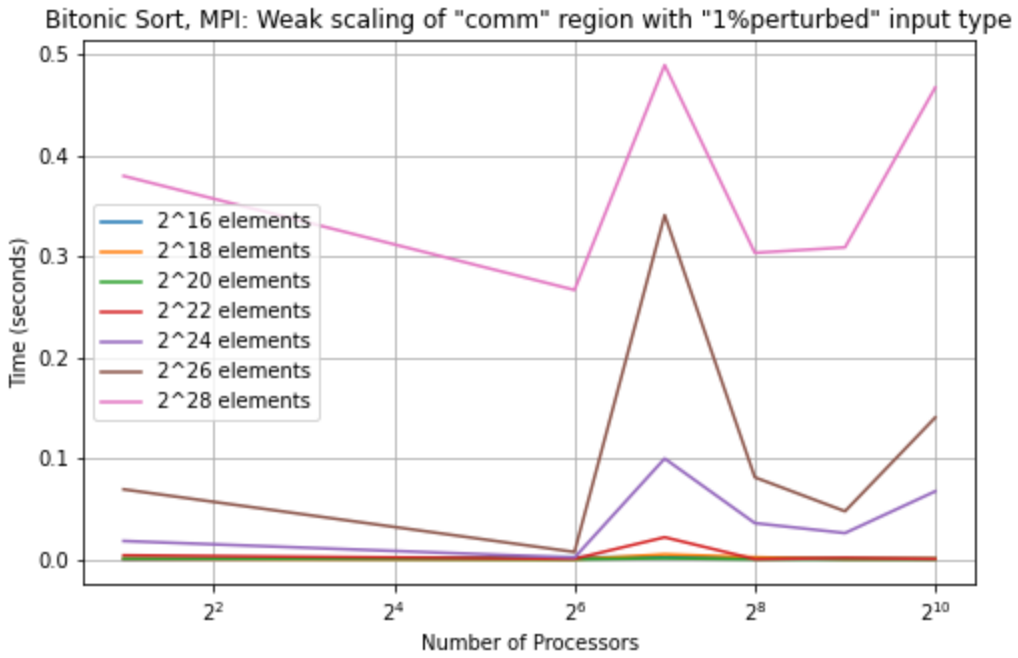

Bitonic Sort, MPI: Strong scaling of "main" region with 2^20 elements
sorted
1%perturbed
reverse_sorted
random
Time (seconds)
Number of Processors
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
200
400
600
800
1000

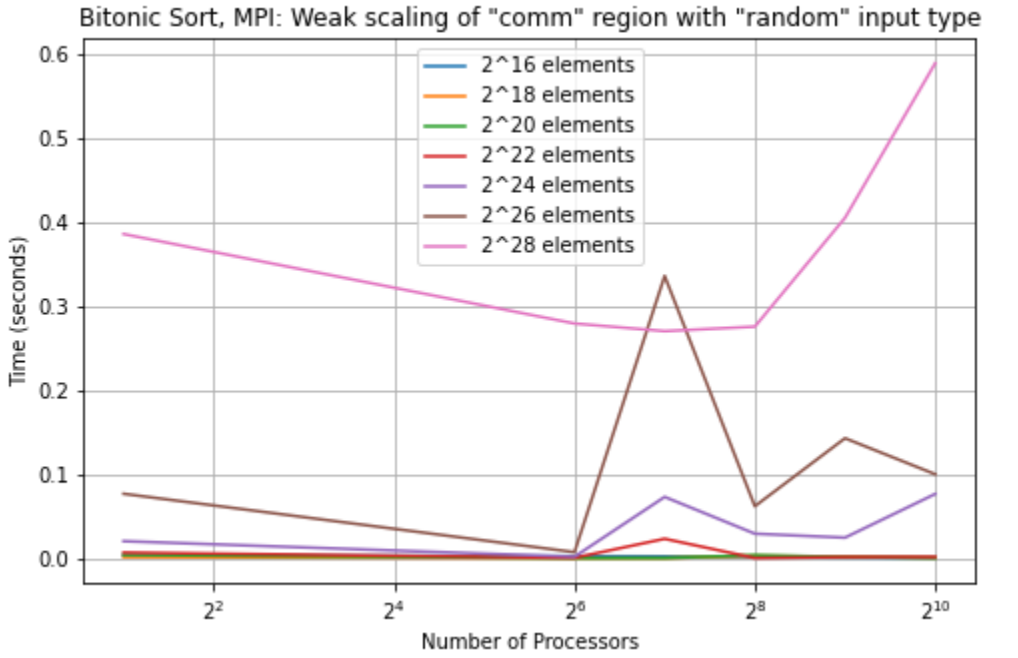

Bitonic Sort, MPI: Strong scaling of "main" region with 2^22 elements
1%perturbed
random
reverse_sorted
sorted
Time (seconds)
Number of Processors
2
4
6
8
10
12
0
200
400
600
800
1000

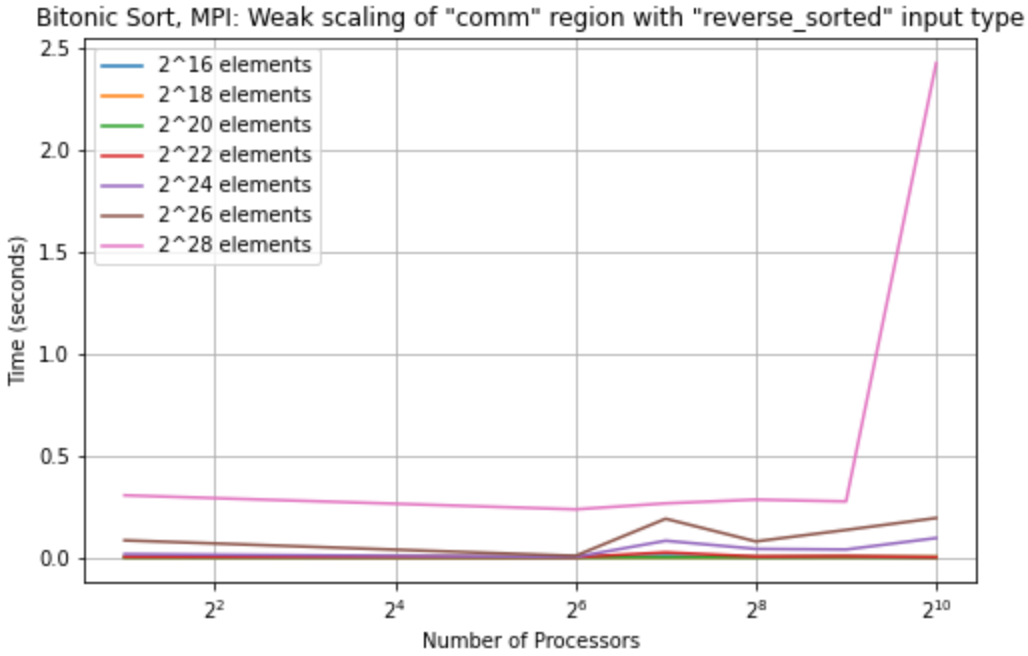

Bitonic Sort, MPI: Strong scaling of "main" region with 2^24 elements
reverse_sorted
random
sorted
1%perturbed
Time (seconds)
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0
200
400
600
800
1000
Number of Processors

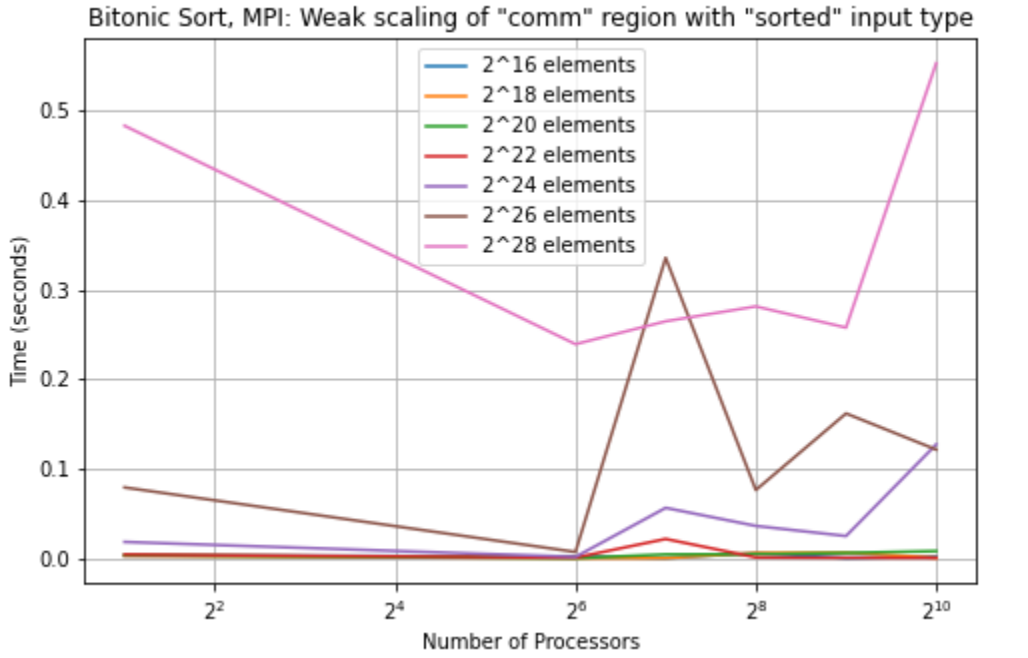

Bitonic Sort, MPI: Strong scaling of "main" region with 2^26 elements
reverse_sorted
random
sorted
1%perturbed
Time (seconds)
25
20
15
10
5
0
200
400
600
800
1000
Number of Processors

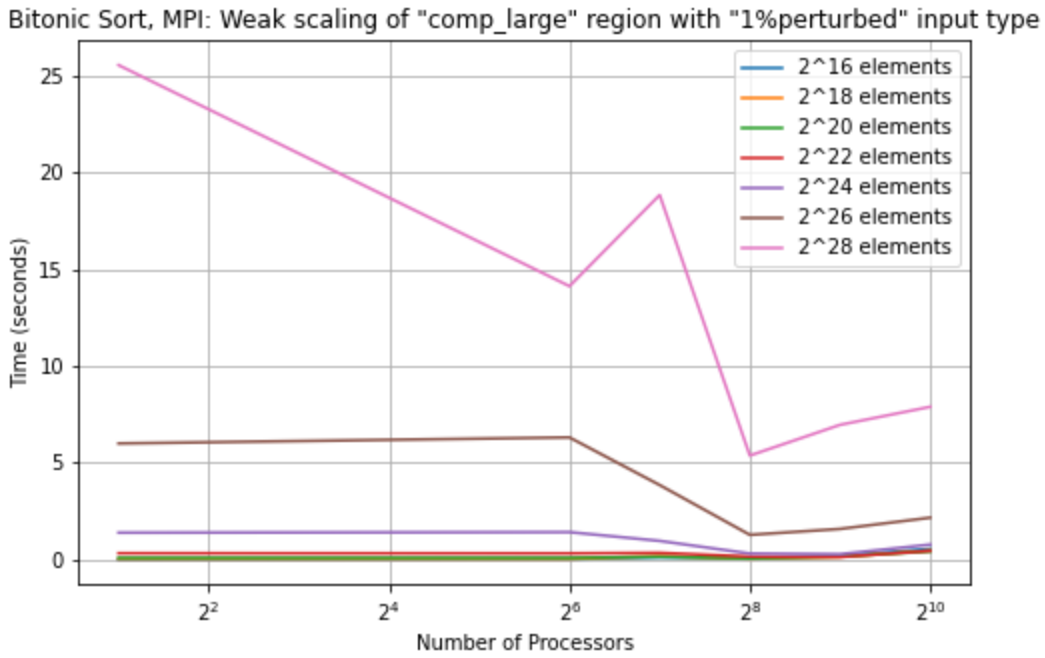

Bitonic Sort, MPI: Strong scaling of "main" region with 2^28 elements
sorted
1%perturbed
random
reverse_sorted
Time (seconds)
100
80
60
40
20
0
200
400
600
800
1000
Number of Processors

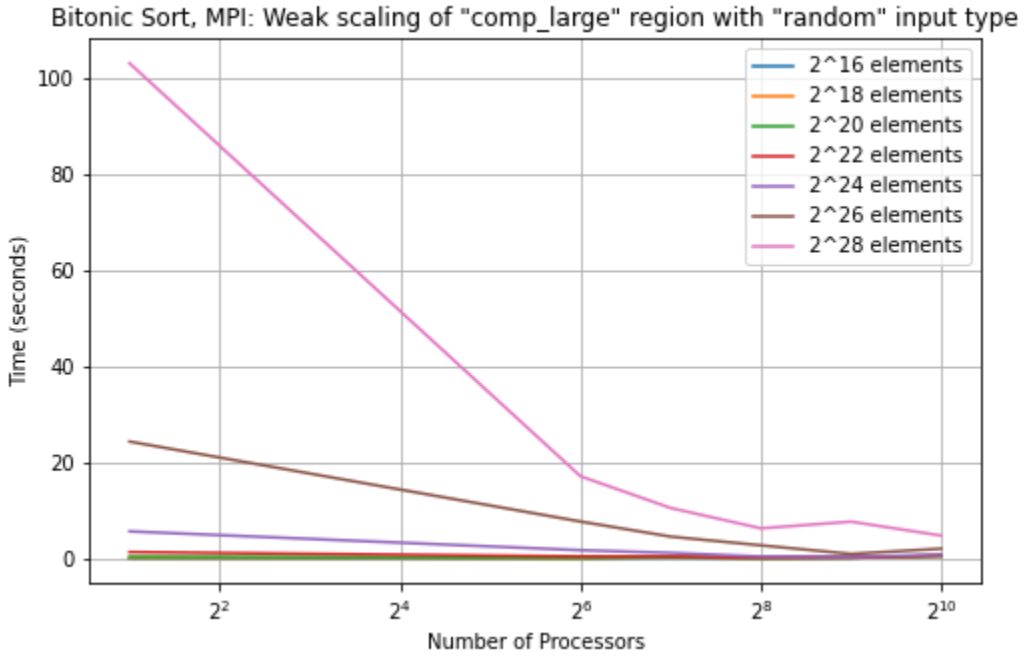

Bitonic Sort, MPI: Weak scaling of "comm" region with "1%perturbed" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

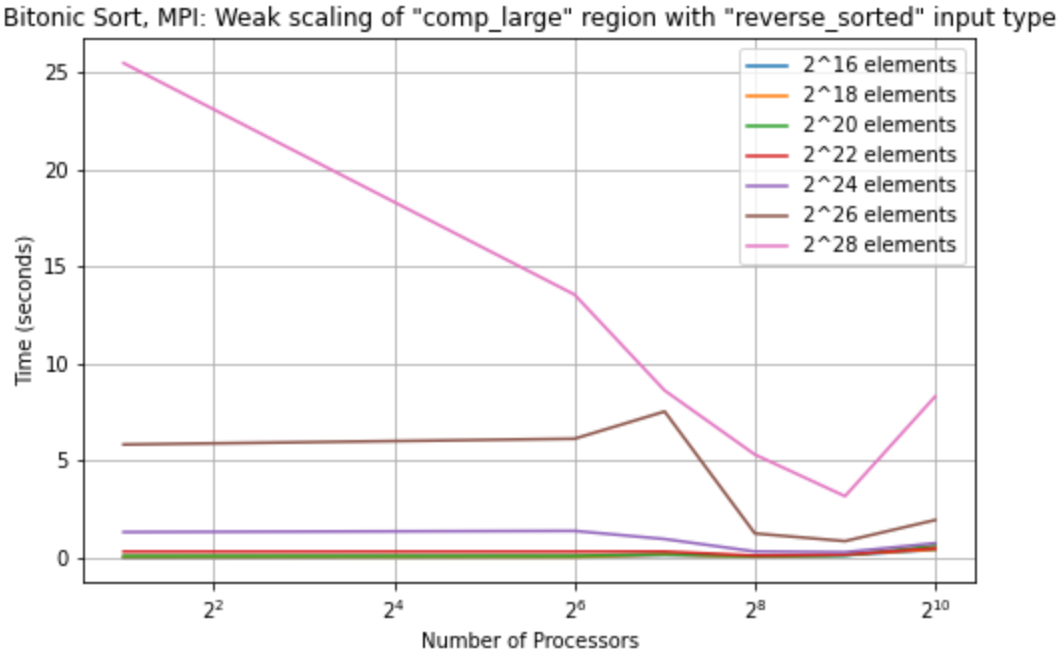

Bitonic Sort, MPI: Weak scaling of "comm" region with "random" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

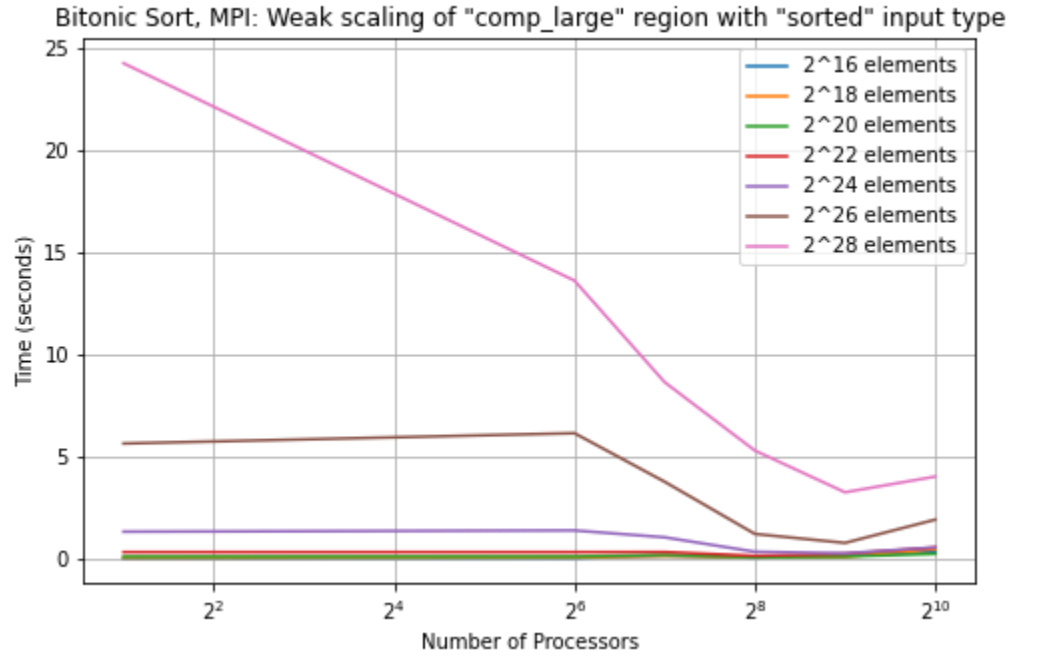

Bitonic Sort, MPI: Weak scaling of "comm" region with "reverse_sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
Number of Processors

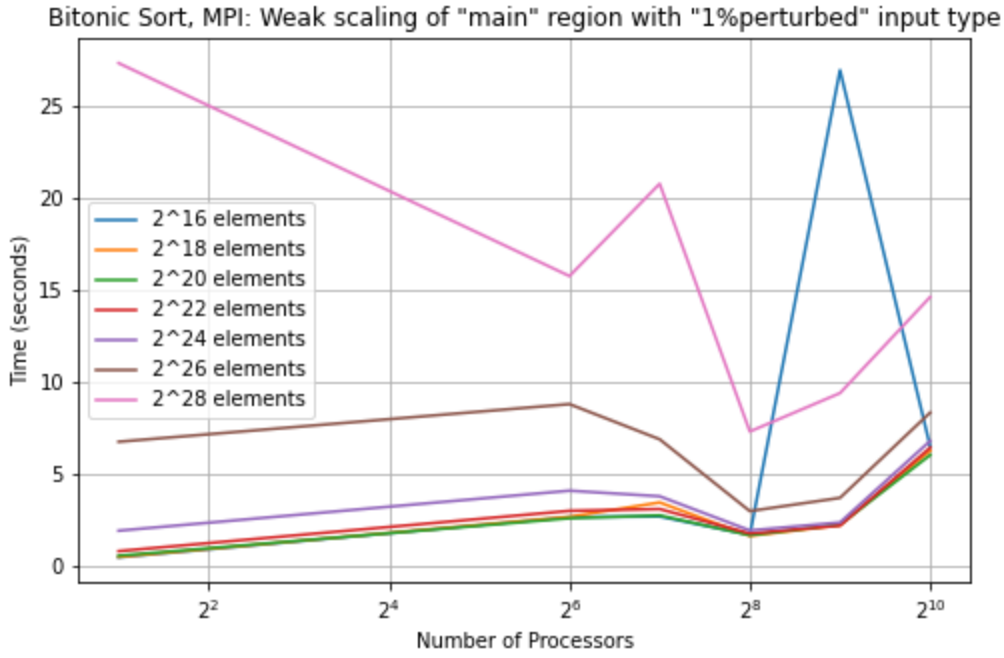

Bitonic Sort, MPI: Weak scaling of "comm" region with "sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Time (seconds)
Number of Processors

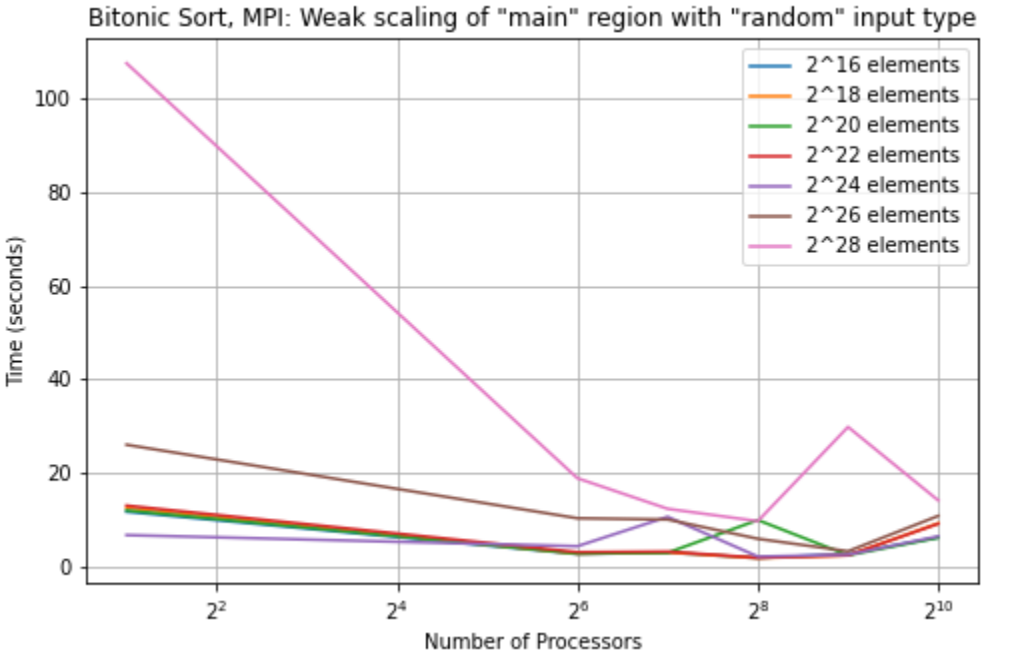

Bitonic Sort, MPI: Weak scaling of "comp_large" region with "1%perturbed" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

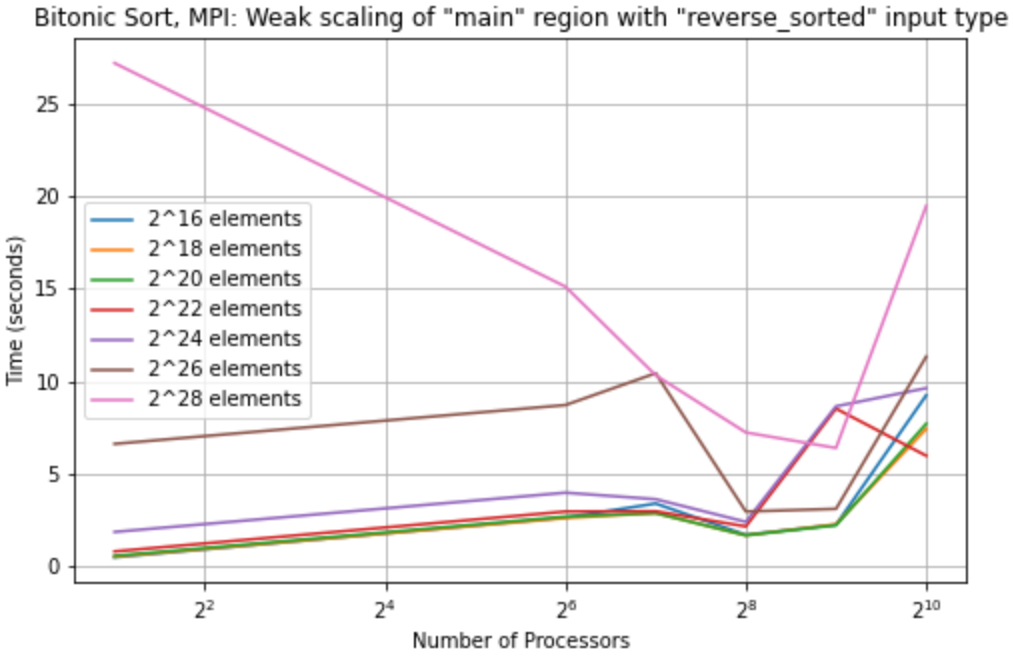

Bitonic Sort, MPI: Weak scaling of "comp_large" region with "random" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

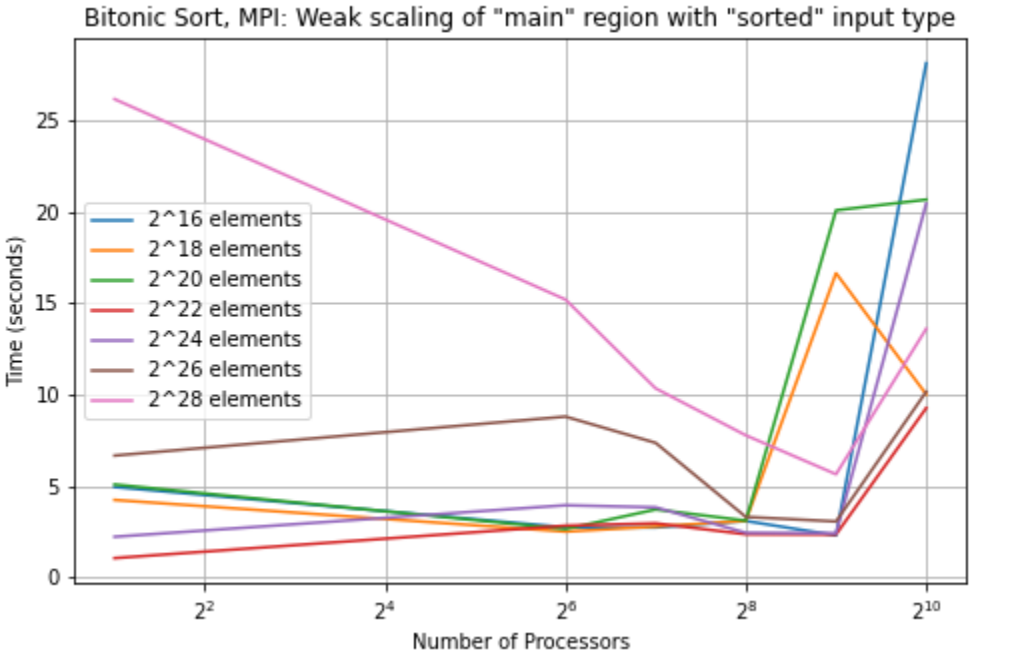

Bitonic Sort, MPI: Weak scaling of "comp_large" region with "reverse_sorted" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

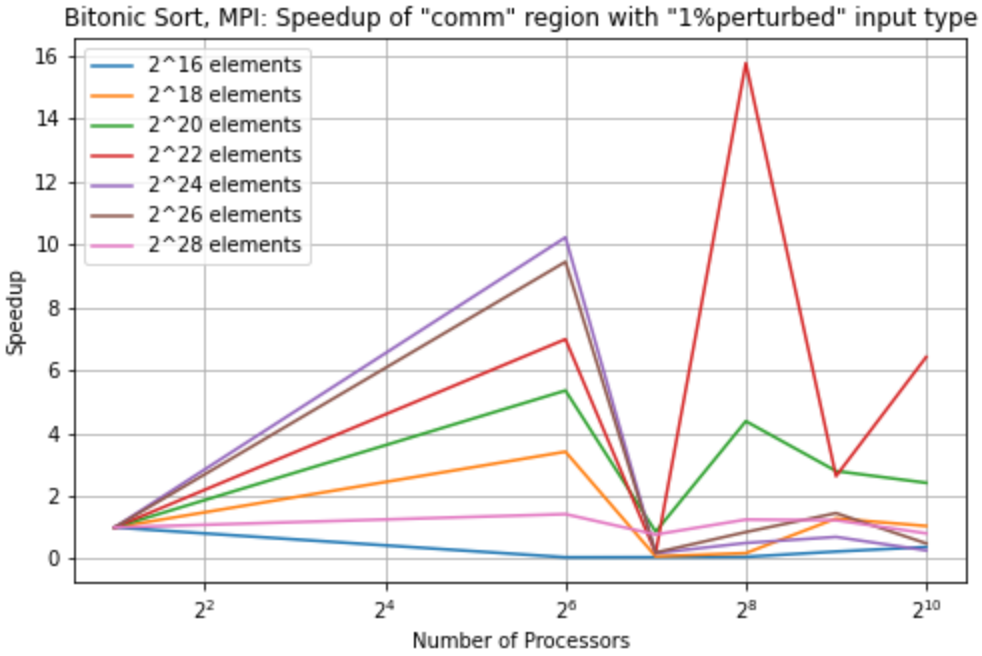

Bitonic Sort, MPI: Weak scaling of "comp_large" region with "sorted" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

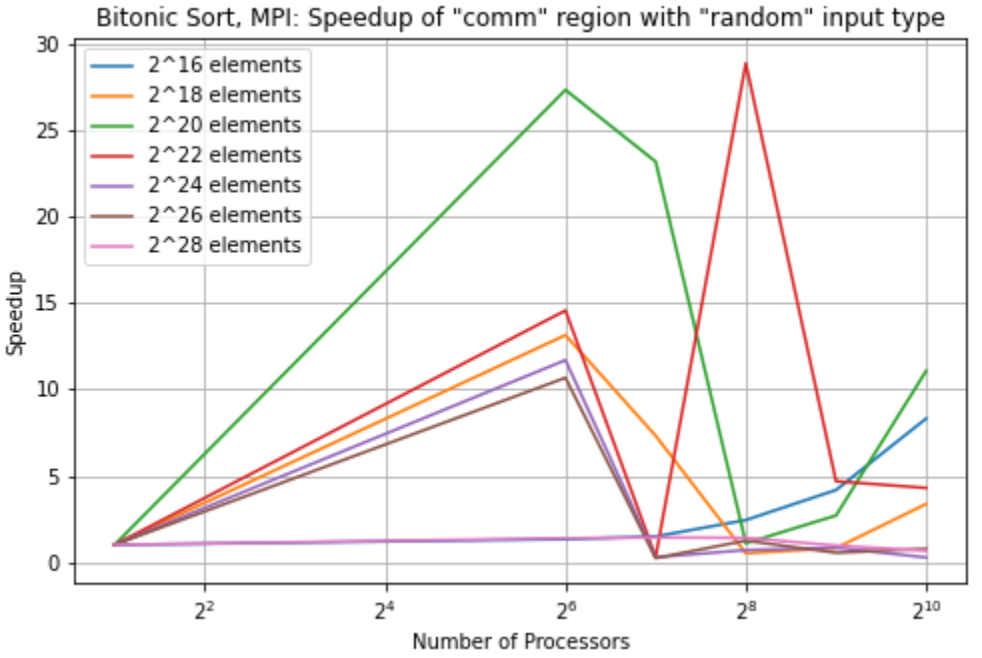

Bitonic Sort, MPI: Weak scaling of "main" region with "1%perturbed" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

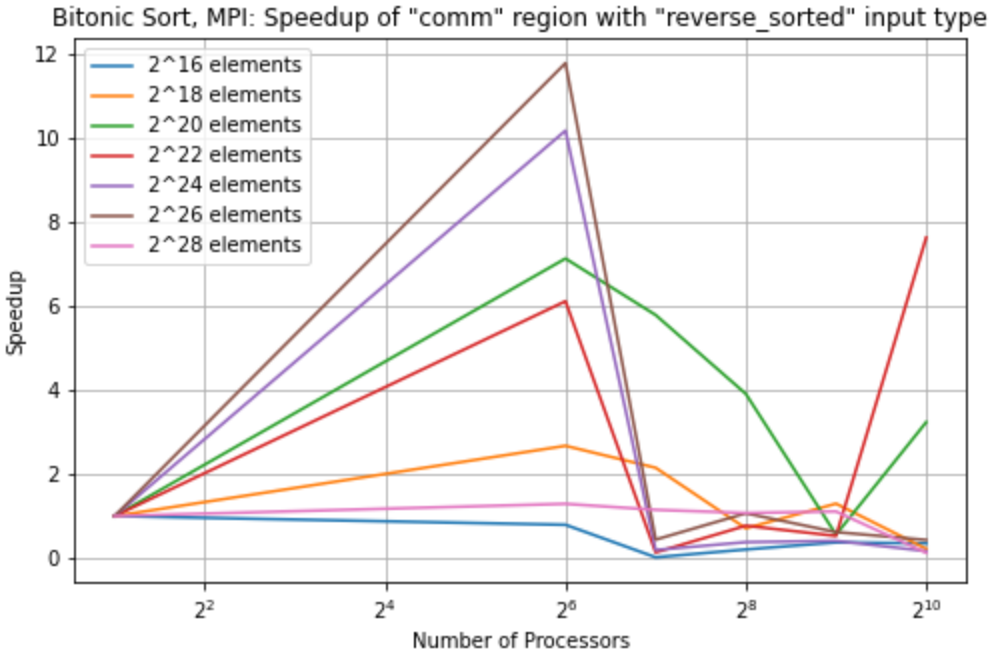

Bitonic Sort, MPI: Weak scaling of "main" region with "random" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

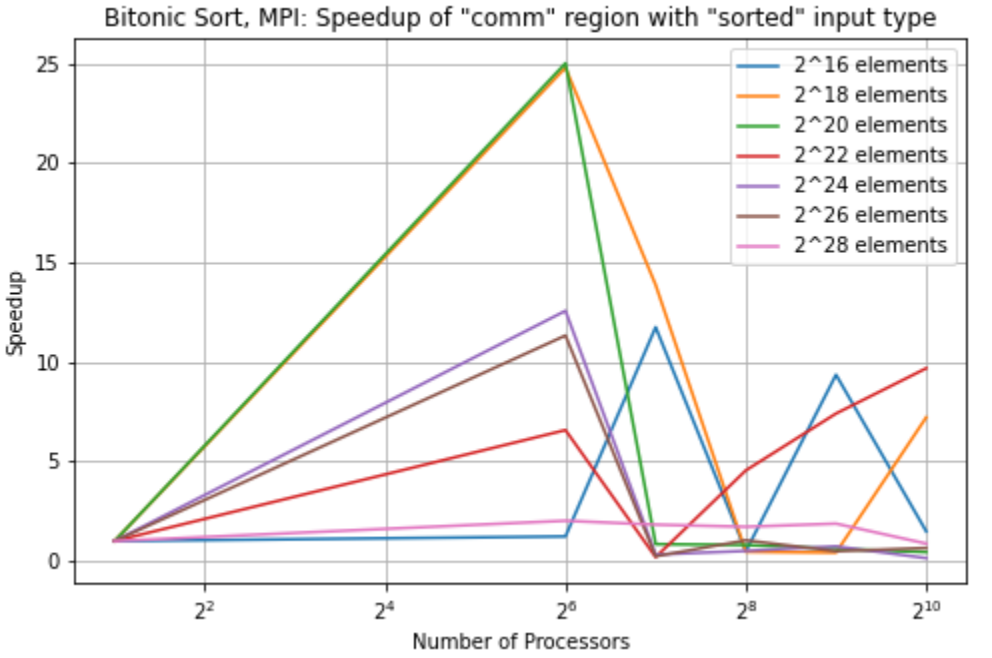

Bitonic Sort, MPI: Weak scaling of "main" region with "reverse_sorted" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

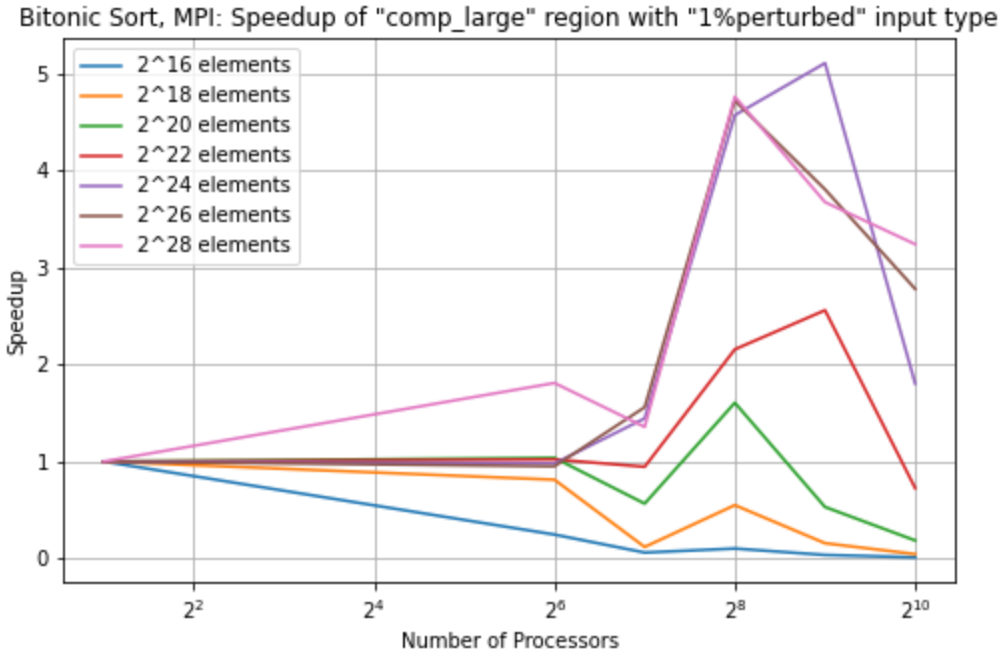

Bitonic Sort, MPI: Weak scaling of "main" region with "sorted" input type
Time (seconds)
Number of Processors
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements

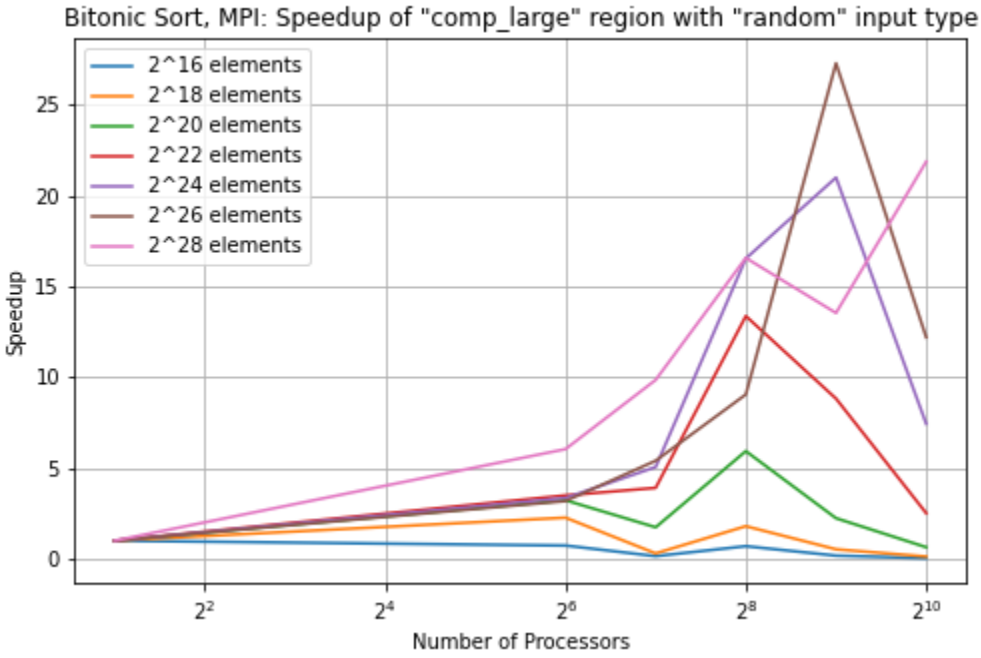

Bitonic Sort, MPI: Speedup of "comm" region with "1%perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors

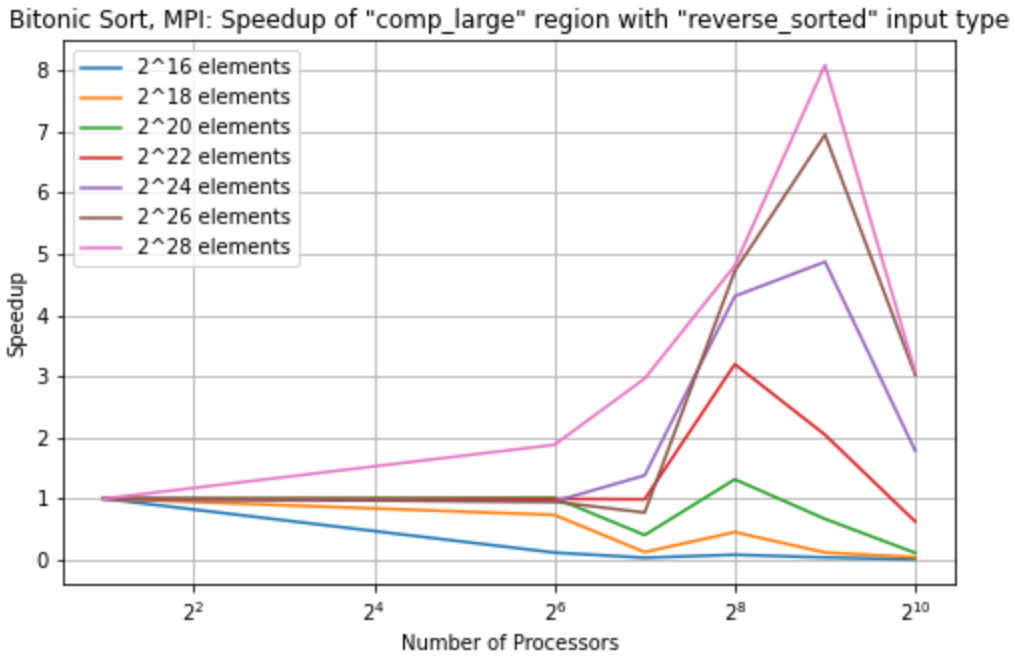

Bitonic Sort, MPI: Speedup of "comm" region with "random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors

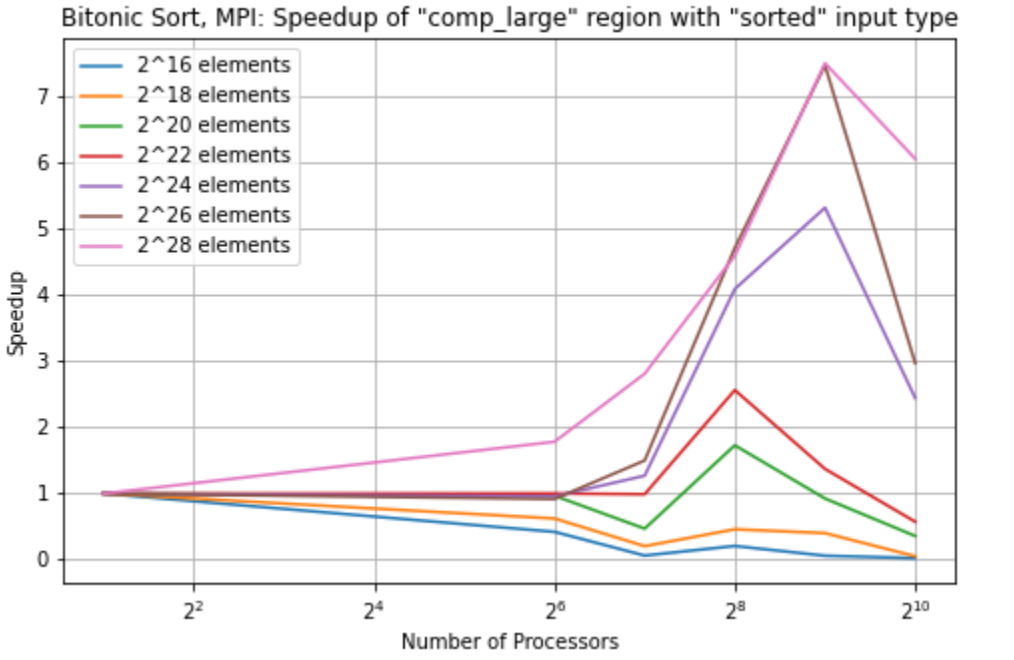

Bitonic Sort, MPI: Speedup of "comm" region with "reverse_sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors

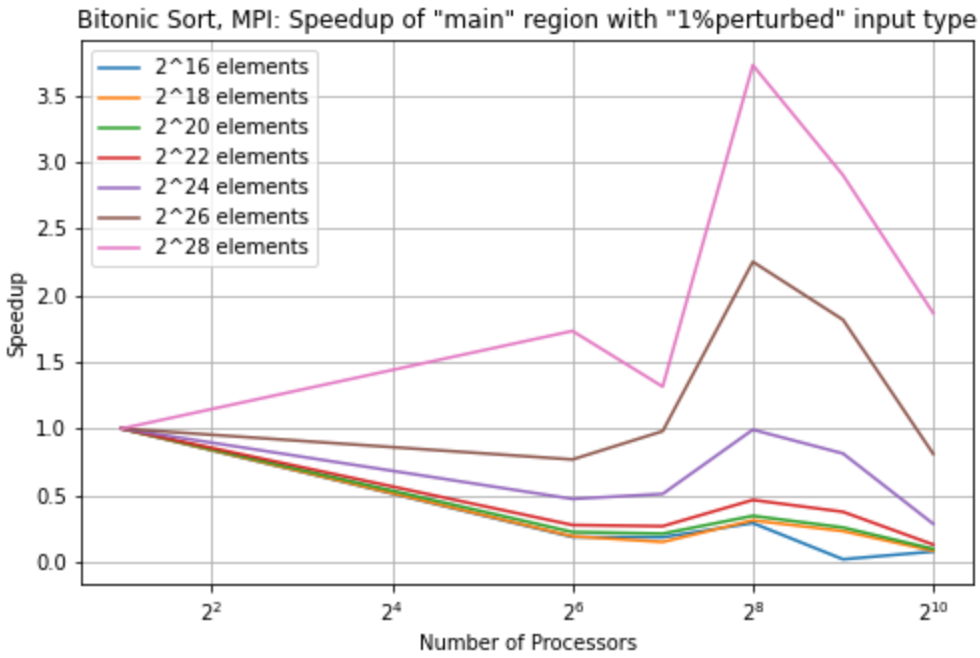

Bitonic Sort, MPI: Speedup of "comm" region with "sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors

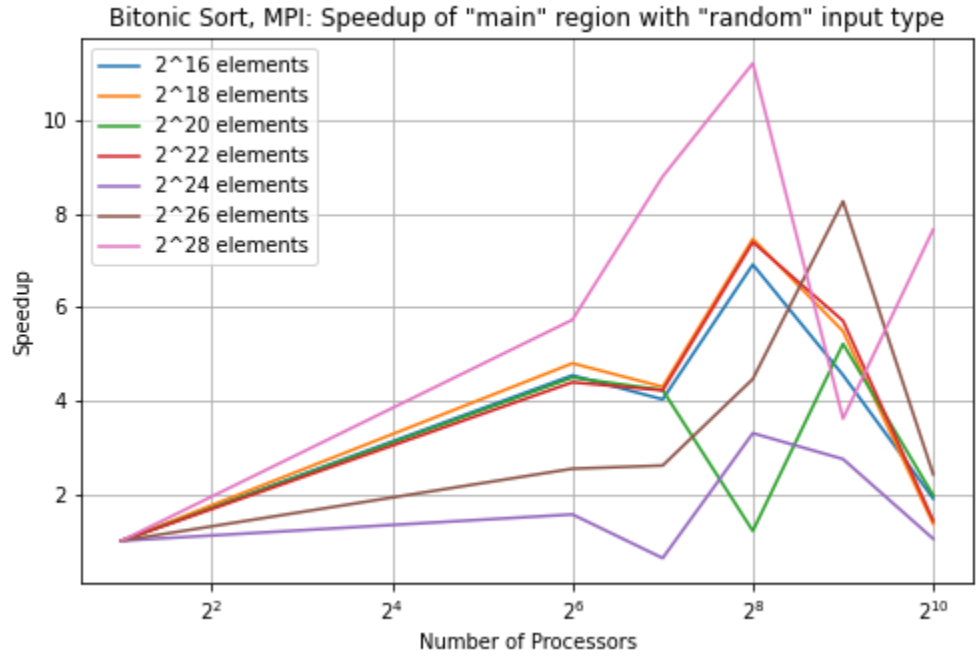

Bitonic Sort, MPI: Speedup of "comp_large" region with "1%perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors

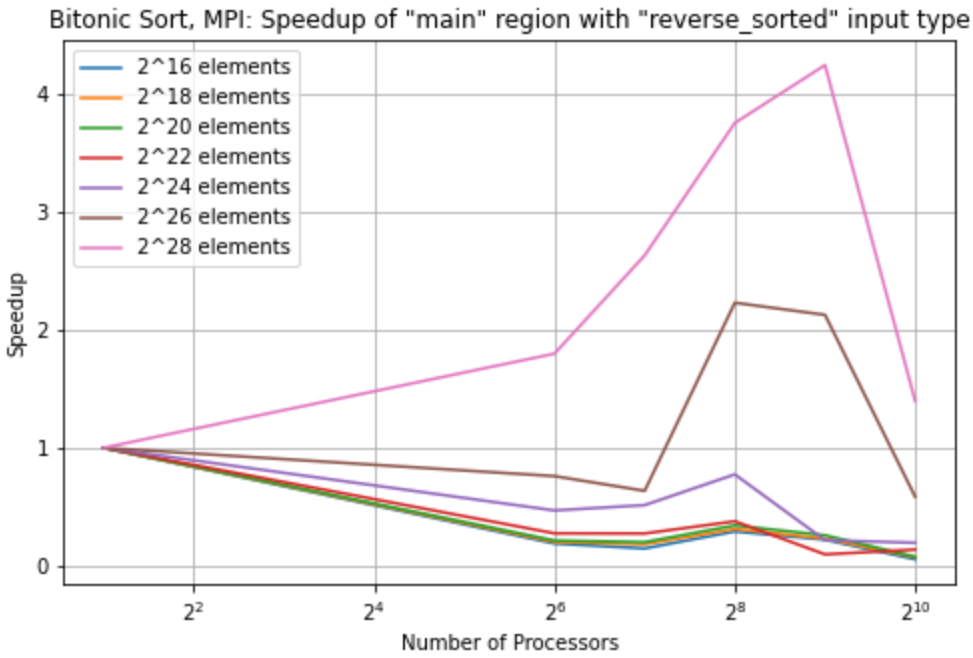

Bitonic Sort, MPI: Speedup of "comp_large" region with "random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors

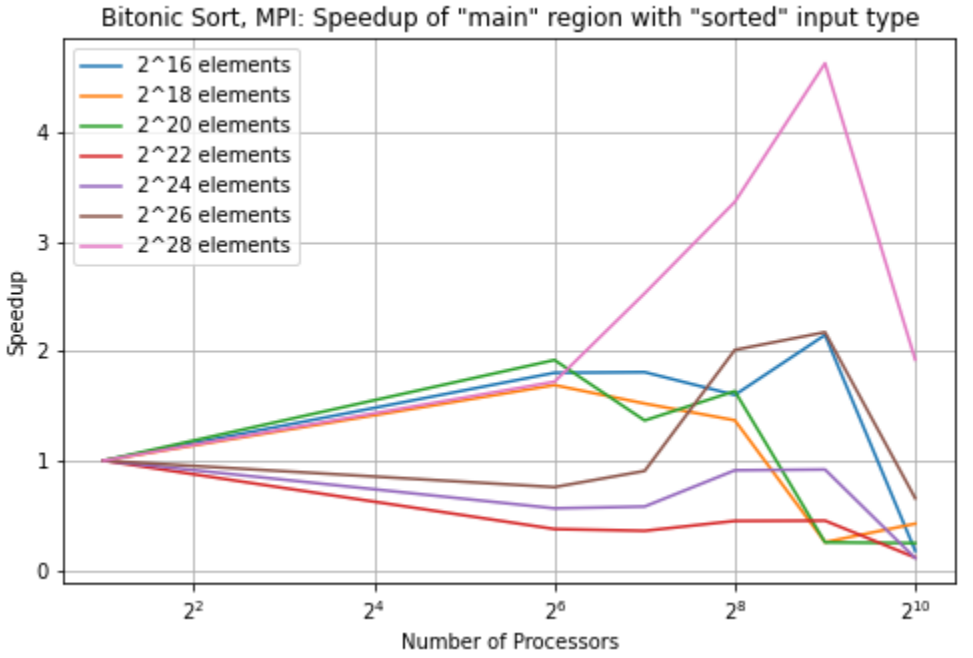

Bitonic Sort, MPI: Speedup of "comp_large" region with "reverse_sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors


Bitonic Sort, MPI: Speedup of "comp_large" region with "sorted" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
Number of Processors


Bitonic Sort, MPI: Speedup of "main" region with "1%perturbed" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors


Bitonic Sort, MPI: Speedup of "main" region with "random" input type
2^16 elements
2^18 elements
2^20 elements
2^22 elements
2^24 elements
2^26 elements
2^28 elements
Speedup
2
4
6
8
10
$2^2$
$2^4$
$2^6$
$2^8$
$2^{10}$
Number of Processors

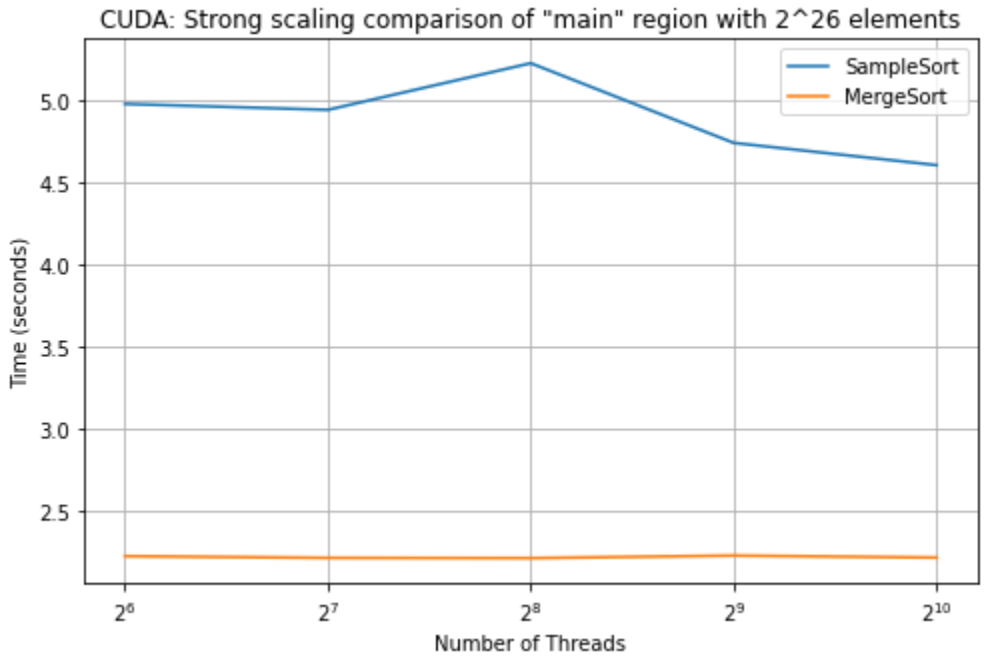


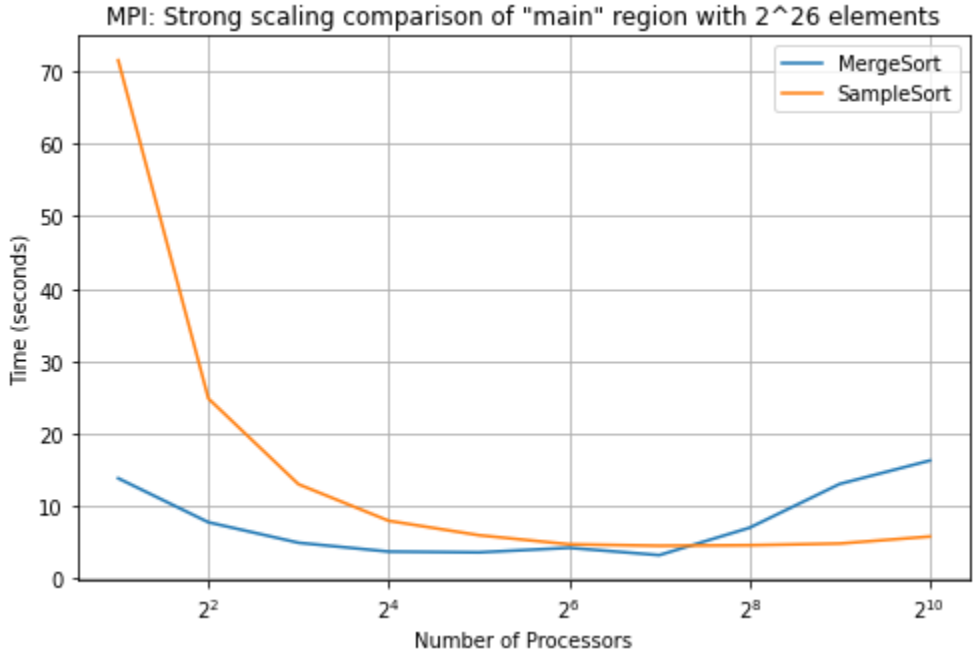




Throughout all of the implementations, we can expect an increase in the computational and communication time as the size of the array increases. We can also expect bitonic sort parallelism to reach a point of diminishing returns as we increase the number of threads. Additionally, as we change the different array populations such as random, sorted, reverse sorted, and 1% perturbed, we can expect slight differences in our

runtimes. For instance, a “sorted” array should have significantly less runtime than an unsorted array.

## Comparison



When comparing Sample Sort and Merge Sort, you can see that Merge Sort was consistently faster with CUDA and MPI. The only time that Sample Sort was faster was with MPI for 2^7, 2^8, 2^9, and 2^10 processors.