CSCE 435 Group project

Group number: 15

Group members:

1. **Aurko Routh**
2. **Srikar Potlapalli**
3. **Andrew Chian**
4. **Soohwan Kim**

Project Topic: Parallel sorting algorithms

Performance comparison of sorting algorithms implemented with CUDA and MPI

Brief project description

In this project, we aim to evaluate and compare the performance of various parallel sorting algorithms on different implementations using different numbers of processes/ threads, input sizes, and input types. We will be implementing the following sorting algorithms:

1. Enumeration Sort(Srikar)
2. Odd-Even Transposition Sort(Andrew)
3. Merge Sort(Aurko)
4. Bitonic Sort(Soowhan)

Pseudocode for each parallel algorithm

Enumeration Sort (CUDA)

```
### Psuedo code for Enumeration Sort(CUDA)
```

```
begin
    initialize rank_array, sorted_array

   for each process do
        divide indexes in array to processes so processes work on indices which are their number + number of workers (loop).
            loop through array and compare current index to all other values
                Increment index of current value in the same index of rank array if (A[i] < A[j]) or A[i] = A[j] and i < j).
                else do not increment

    synchronize

   for each process do
        divide indexes in array to processes so processes work on indices which are their number + number of workers (loop).

            sorted_array[rank[working idx]] = array[working idx];


end ENUM_SORTING
```

Odd-Even Transposition Sort (CUDA)

```
// Define the array size
let N = size of the array

// CUDA kernel function for odd-even transposition sort
__global__ void oddEvenSortKernel(float *d_a, int n, int phase)
{
  // Calculate global thread index
  int index = threadIdx.x + blockDim.x * blockIdx.x;
  int idx1 = index;
  int idx2 = index + 1;

  // Check whether we are in an odd or even phase
  if ((phase % 2 == 0) && (idx2 < n) && (idx1 % 2 == 0))
  { // Even phase
    if (d_a[idx1] > d_a[idx2])
    {
      // Swap elements
      float temp = d_a[idx1];
      d_a[idx1] = d_a[idx2];
      d_a[idx2] = temp;
    }
  }
  else if ((phase % 2 == 1) && (idx2 < n) && (idx1 % 2 == 1))
  { // Odd phase
    if (d_a[idx1] > d_a[idx2])
    {
      // Swap elements
      float temp = d_a[idx1];
      d_a[idx1] = d_a[idx2];
      d_a[idx2] = temp;
    }
  }
}

// Host function to run the parallelized odd-even transposition sort
void cudaOddEvenSort(float *h_a, int n)
{
  float *d_a;

  // Allocate memory on the device
  cudaMalloc(&d_a, n * sizeof(float));

  // Copy data from host to device
  cudaMemcpy(d_a, h_a, n * sizeof(float), cudaMemcpyHostToDevice);

  // Setup block and grid dimensions
  dim3 blocks(BLOCKS, 1);   // Number of blocks
  dim3 threads(THREADS, 1); // Number of threads per block

  // Launch the kernel for each phase
  for (int i = 0; i < n; ++i)
  {
    oddEvenSortKernel<<<blocks, threads>>>(d_a, n, i);
    cudaDeviceSynchronize();
  }

  // Copy the sorted array back to the host
  cudaMemcpy(h_a, d_a, n * sizeof(float), cudaMemcpyDeviceToHost);
```

Merge Sort (CUDA)

```
global function merge_sort_caller(values):
    allocate device memory for dev_values and temp
    copy values from host to device (dev_values)

    blocks = number_of_blocks
    threads = number_of_threads

    for window from 2 to size of values (doubling each time):
        launch merge_sort CUDA kernel with arguments (dev_values, temp, size of values, window) using blocks and threads

    copy values from device to host (dev_values to values)
    free device memory (dev_values, temp)

global CUDA kernel function merge_sort(values, temp, num_vals, window):
    id = threadIdx.x + blockDim.x * blockIdx.x
    l = id * window
    r = l + window

    if r > num_vals:
        r = num_vals

    m = l + (r - l) / 2

    if l < num_vals:
        call merge function on the device (values, temp, l, m, r)
```

Bitonic Sort (CUDA)

```
global function bitonic_sort(values):
    allocate device memory for dev_values
    size = NUM_VALS * size_of(float)

    cudaMalloc((void**) &dev_values, size)

    // Memory copy from host to device
    cudaMemcpy(dev_values, values, size, cudaMemcpyHostToDevice)

    dim3 blocks(BLOCKS, 1)     // Number of blocks
    dim3 threads(THREADS, 1)   // Number of threads

    create CUDA events for timing

    /* Major step */
    for k from 2 to NUM_VALS (doubling each time):
        /* Minor step */
        for j from k / 2 to 1 (halving each time):
            launch bitonic_sort_step CUDA kernel with arguments (dev_values, j, k) using blocks and threads
            // Timing events for this iteration

    // Memory copy from device to host
    cudaMemcpy(values, dev_values, size, cudaMemcpyDeviceToHost)

    // Free device memory
    cudaFree(dev_values)

    global CUDA kernel function bitonic_sort_step(dev_values, j, k):
        i = threadIdx.x + blockDim.x * blockIdx.x
        ixj = i ^ j // Sorting partners

        if ixj > i:
            if (ixj) > i and (i & k) == 0:
                // Sort ascending
                if dev_values[i] > dev_values[ixj]:
                    swap dev_values[i] with dev_values[ixj]

            if (ixj) > i and (i & k) != 0:
                // Sort descending
                if dev_values[i] < dev_values[ixj]:
                    swap dev_values[i] with dev_values[ixj]
```

Enumeration Sort (MPI)

```
begin
    set calculations_per_thread = input_size / processes
    initialize rank, rank_idx arrays to the same size as the calculations_per_thread
    initialize recieved_array, sorted_array to the same size as input_size

    MPI broadcast the original array to all processes

    All processes do the following:
        start

            set count variable to 0
            loop through original array skipping num_processes each time to get working index
                set rank array at count to 0
                set rank_idx array at count to working index

                increment rank array at count by one for every other element it is greater than. If it is equal
                to another element, increase only if working index is greater than comparing index

                increment count by 1
        stop

    recieve results from all processes sequentially and set sorted_array[rank[i]] = original_array[rank_idx[i]];
end
```

Merge Sort(MPI)

```
// Initialize variables
subarr_size = n / size
i = 1
while subarr_size <= n
    declare custom_comm
    if rank % i == 0
        custom_comm = new MPI_Communicator
    else
        custom_comm = MPI_UNDEFINED
    // If part of the new communicator
    if rank % i == 0
        // Get the rank within the custom communicator
        custom_rank = get_comm_rank(custom_comm)
        // Allocate memory for local array and temporary array
        local_arr = allocate_array(float, subarr_size)
        temp = allocate_array(float, subarr_size)
        // Scatter operation
        MPI_Scatter(data: arr, count: subarr_size, datatype: float, recvbuf: local_arr, root: 0, comm: custom_comm)
        // Perform merge sort on local array
        mergeSort(array: local_arr, helper: temp, low: 0, high: subarr_size - 1)
        // Gather operation
        MPI_Gather(sendbuf: local_arr, count: subarr_size, datatype: float, recvbuf: arr, root: 0, comm: custom_comm)

    subarr_size <<= 1
    i <<= 1
```

Odd-even Transposition (MPI)

```
Variables:
    world_rank, world_size

Procedure Merge_low(local_A, temp_B, temp_C, local_n):
    Initialize ai, bi, ci to 0
    While ci < local_n:
        If ai < local_n and (bi >= local_n or local_A[ai] <= temp_B[bi]):
            Assign local_A[ai] to temp_C[ci]
            Increment ai
        Else:
            Assign temp_B[bi] to temp_C[ci]
            Increment bi
        Increment ci
    Copy temp_C back into local_A

Procedure Merge_high(local_A, temp_B, temp_C, local_n):
    Initialize ai, bi, ci to local_n - 1
    While ci >= 0:
        If ai >= 0 and (bi < 0 or local_A[ai] >= temp_B[bi]):
            Assign local_A[ai] to temp_C[ci]
            Decrement ai
        Else:
            Assign temp_B[bi] to temp_C[ci]
            Decrement bi
        Decrement ci
    Copy temp_C back into local_A

Function Compare_floats(a, b):
    Convert a and b to floats fa and fb
    Return (fa > fb) - (fa < fb)

Procedure Odd_even_iter(local_A, temp_B, temp_C, local_n, phase, even_partner, odd_partner, my_rank, p, comm):
    If phase % 2 == 0:
        If even_partner >= 0:
            Perform MPI_Sendrecv with even_partner
            If my_rank % 2 != 0:
                Call Merge_high
            Else:
                Call Merge_low
    Else:
        If odd_partner >= 0:
            Perform MPI_Sendrecv with odd_partner
            If my_rank % 2 != 0:
                Call Merge_low
```

```
            Call Merge_low
    Else:
        If odd_partner >= 0:
            Perform MPI_Sendrecv with odd_partner
            If my_rank % 2 != 0:
                Call Merge_low
            Else:
                Call Merge_high

Procedure Sort(local_A, local_n, my_rank, p, comm):
    Allocate memory for temp_B and temp_C
    Determine even_partner and odd_partner based on my_rank
    Sort local_A using quick sort
    For phase from 0 to p-1:
        Call Odd_even_iter
    Free memory allocated for temp_B and temp_C
```

Bitonic Sort (MPI)

```
1. Initialize MPI
2. Get the total number of processes (size) and the rank of the current process (rank

function bitonicSort(arr, direction, myRank, numProcesses):
    localSize = length(arr) / numProcesses
    localArr = scatter(arr, localSize)  // Distribute the data among processes

    for step in 1 to log2(numProcesses):  // Logarithmic phases
        for subStep in step-1 to 0 step -1:  // Each phase has sub-steps
            mask = 2^(subStep + 1) - 1
            partner = myRank xor mask
            localArr = compareAndSwap(localArr, partner, direction)

    gather(arr, localArr, localSize)  // Gather the sorted data back to the root proc

end function

function compareAndSwap(localArr, partner, direction):
    // Communication between two processes
    sendArray = localArr
    recvArray = receive(partner)

    if (myRank < partner) XOR (direction == ASCENDING):
        if localArr > recvArray:
            localArr = recvArray
    else:
        if localArr < recvArray:
            localArr = recvArray

    send(localArr, partner)

    return localArr

end function


// Main program
if rank == 0:
    arr = generate_random_array()
else:
    arr = empty_array()

direction = ASCENDING  // or DESCENDING
bitonicSort(arr, direction, rank, size)

if rank == 0:
    print("Sorted Array:", arr)

Finalize MPI
```

Evaluation plan - what and how will you measure and compare

We will measure and compare the performance of each algorithm based on the following criteria:

- Input sizes
    - 2^16, 2^18, 2^20, 2^22, 2^24, 2^26, 2^28
- Input types
    - Sorted, Random, Reverse sorted, 1%perturbed
- Number of processes / threads
    - MPI: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024
    - CUDA: 64, 128, 256, 512, 1024
- Strong scaling (same problem size, varying number of processors/threads)
- Weak scaling (increasing problem size with the number of processes)
- Speedup plot

The algorithms will be tested with every possible combination of input sizes, input types, and number of processes/threads. Different input sizes will provide insights to how the algorithms perform on small or large datasets, different input types will help assess the robustness of the algorithms under different data distributions, and different number of processes / threads will show the efficiency of parallelism in the algorithm.

We will generate a strong scaling plot for each ‘main’, ‘comp’, and ‘comm’ section in the algorithm, comparing the number of processes/threads vs average runtime for each input type in the same plot with a fixed input size of 2^28.

We will also generate a weak scaling plot for each ‘main’, ‘comp’, and ‘comm’ section in the algorithm and for each input type. In the plot, we are comparing the number of processes / threads vs average runtime for different input sizes in the same plot.

Lastly, we generate the speedup plot, which is similar to strong scaling except we are now considering the ratio of the algorithm with a single process/thread to the algorithm with parallel processes/threads.

Performance evaluation

Considerations

For all CUDA algorithms, we did not test the thread counts of 2048 or 4096 because the GRACE computer limited the max thread count to 1024. Also there was a mistake on the speedup plot generation where in the title it says an input size. Disregard that since the speedup plots plot multiple input sizes.

Enumeration Sort:

Considerations:

Enumeration sort is an algorithm that is really only good for smaller input sizes and performs poorly at larger input sizes. Each thread must perform at minimum (input_size^2)/(num_threads) comparisons/operations, so at larger input sizes the algorithm takes a lot longer to complete.

We did not have all of the cali files for enumeration sort in the submission because the algorithm took longer than 30 min to run and was forced to exit because the time limit was set to 30 min. Some of the cali files show the algorithm taking 30-60 min because the time limit was set to 60 min for those jobs submissions. However, this could not be done for all of the jobs because it was very costly and we ran out of account credits by doing this. The missing cali files (data points) were job submissions that took greater than 30 min to complete.

For the CUDA implementation of enumeration sort, we changed the input sizes to allow for more data to be present in the final graph. Only the first 3 input sizes yielded results that took less than 30 min to complete. Because of this, we measured the algorithm's time at the same process numbers for input sizes of 2^15, 2^17, and 2^19.

The completion time of the algorithm for 1 process was also measured in MPI and CUDA for all input types and sizes, however the algorithm was often (unsurprisingly) taking longer than 30 min to complete. So that is an explanation for the missing single process cali files and any missing speedup plots.

Performance Analysis:

The weak scaling graph for enumeration in MPI shows a comparison of how input size affects the algorithm's performance as the number of processes increases. The bottom most line starts higher than zero but looks that way relative to the other lines. Because the minimum number of comparisons/operations that this algorithm needs to do is (input_size)^2/(num_procs), increasing the input size drastically increases the average completion time per process. Given the equation in the previous sentence, if the algorithm takes t time to complete and you increase the input size by a factor of 4(2^2) and increase the number of processes by a factor of 16(2^4), the algorithm should also take t time to complete. The graph reflects this trend.

The Strong scaling graphs for enumeration comp large in MPI show a comparison of average times/rank between different input types. The graphs show that Reverse Sorted is the fastest input type when compared to the other input types by a large margin. The average time for a reverse sorted input is consistently 2x faster than the random input type. One possible explanation for this is that the way the algorithm is implemented, If the algorithm is reverse sorted, the earlier processes will enter the if statement in one comparison and perform an operation while the later processes will perform three comparisons and then maybe perform an operation or not (referenced code shown below). This means that the reverse sorted array will have half as many operations to do as the sorted array and will therefore be twice as fast. If this does not explain the speed up, another cause could be that the earlier processes will finish faster than the later processes. Additionally, because the consolidation of the data to one array is done sequentially (no other choice because there is no shared memory), the algorithm will complete the sequential portion faster. Additionally, the average time per rank for each input type cuts in half everytime the number of processes is doubled, which is also consistent with the equation.

```
//CALCULATION PART FOR WORKER PROCESS STARTS HERE

  int count = 0;
  CALI_MARK_BEGIN("comp");
  CALI_MARK_BEGIN("comp_small");
  for(int i = taskid; i < NUM_VALS; i += numworkers_inc_master){

      if (i < NUM_VALS) {
          rank[count] = 0;
          rank_idx[count] = i;
          for (int j = 0; j < NUM_VALS; j++) {
              if (received_data[j] < received_data[i] || (received_data[j] == received_data[i] && j < i)) {
                  rank[count]++;
              }
          }
      }
      count++;
  }
  CALI_MARK_END("comp_small");
  CALI_MARK_END("comp");

//CALCULATION PART FOR WORKER PROCESS ENDS HERE
```

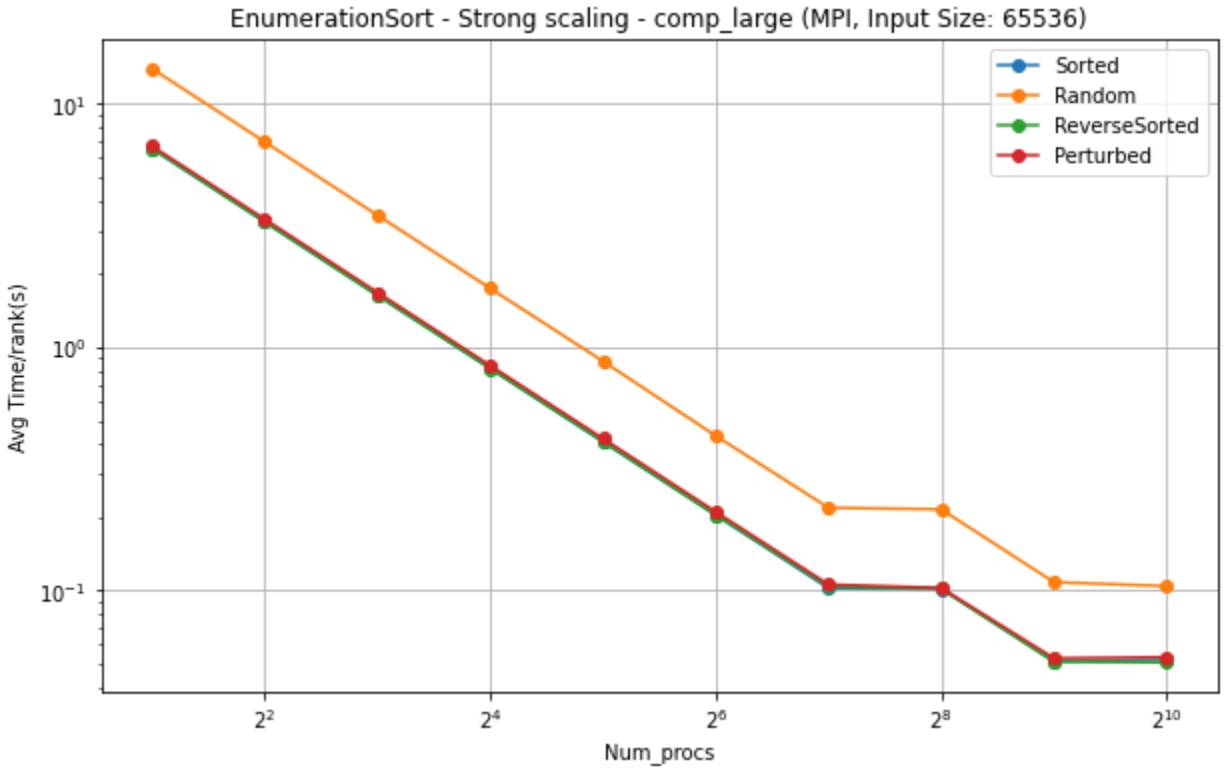
The weak scaling graph for enumeration in CUDA also shows a comparison of how input size affects the algorithm's performance as the number of processes increases. The largest input size has missing data points because with a lower number of processes, the algorithm takes longer than 30 min and those data points are not included.

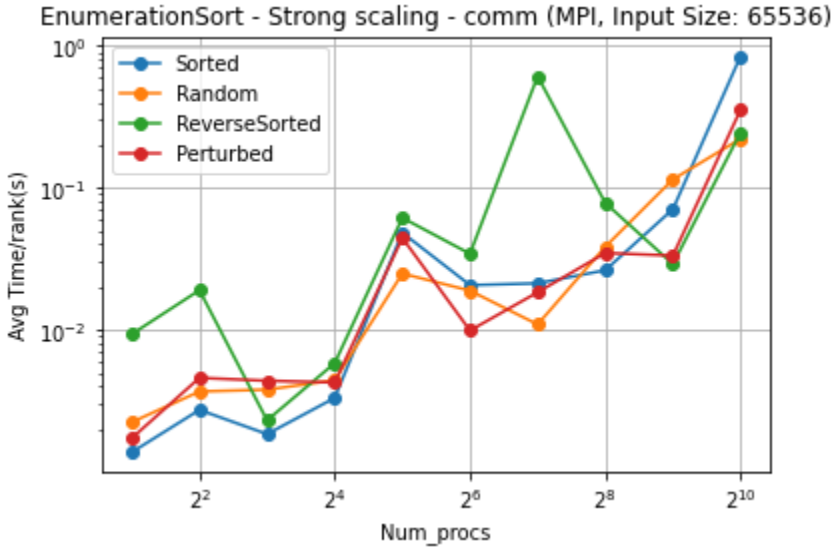
The strong scaling graphs for enumeration CUDA show similar trends as the strong scaling for enumeration MPI. However, because the implementation of the algorithm was slightly different, the completion time stays relatively consistent regardless of the input type. The main reason we think the average time stays consistent is that the consolidation of the sorted array is done in parallel in this implementation because CUDA has shared memory and all of the processes can write into the array at different indexes at the same time.

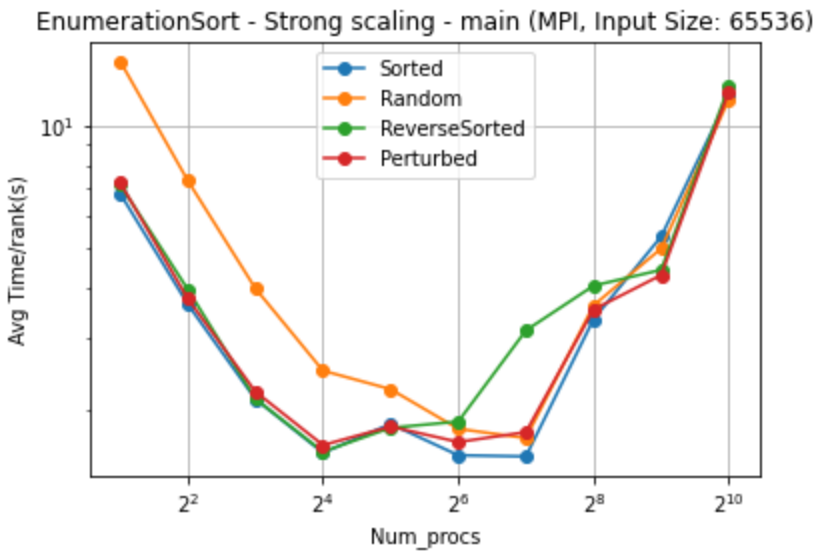
Enumeration Sort Plots:

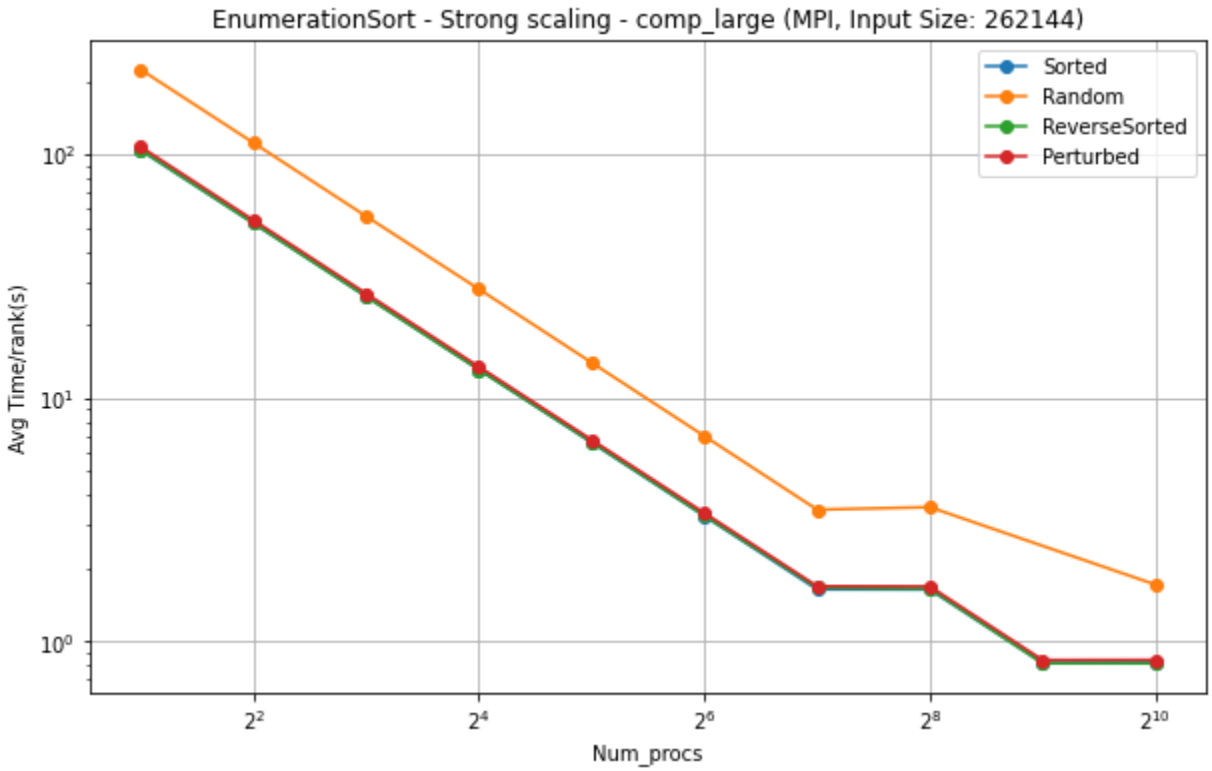
MPI:

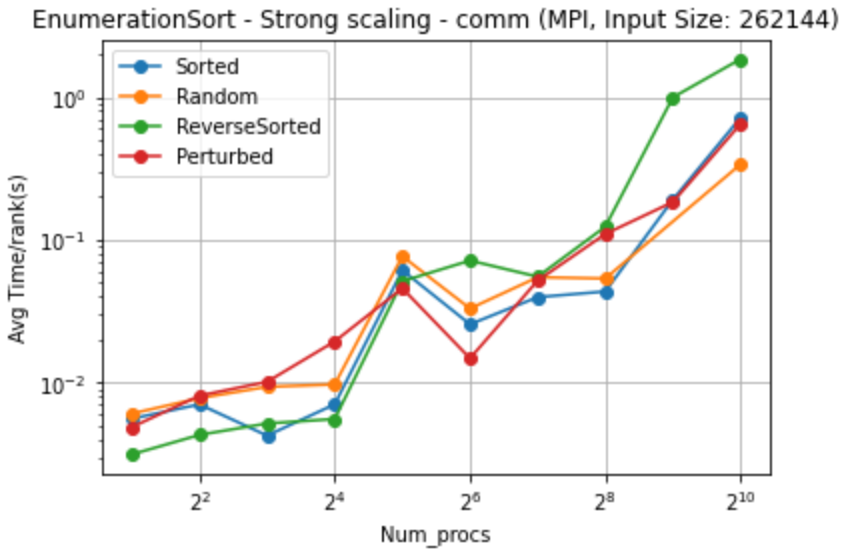
Strong Scaling:



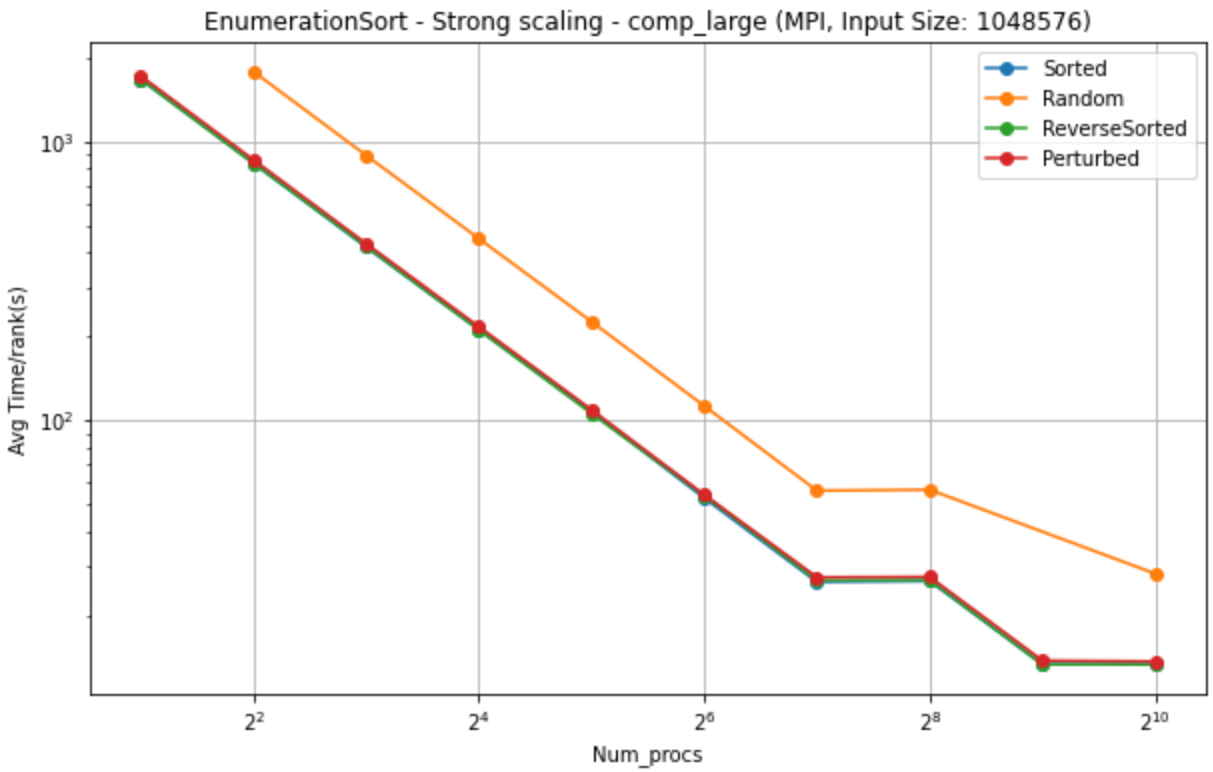


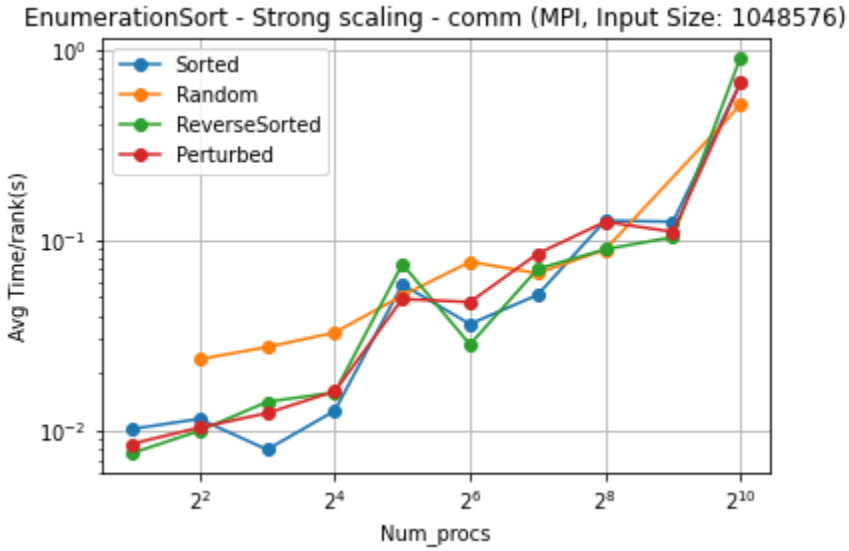

EnumerationSort - Strong scaling - main (MPI, Input Size: 65536)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_procs

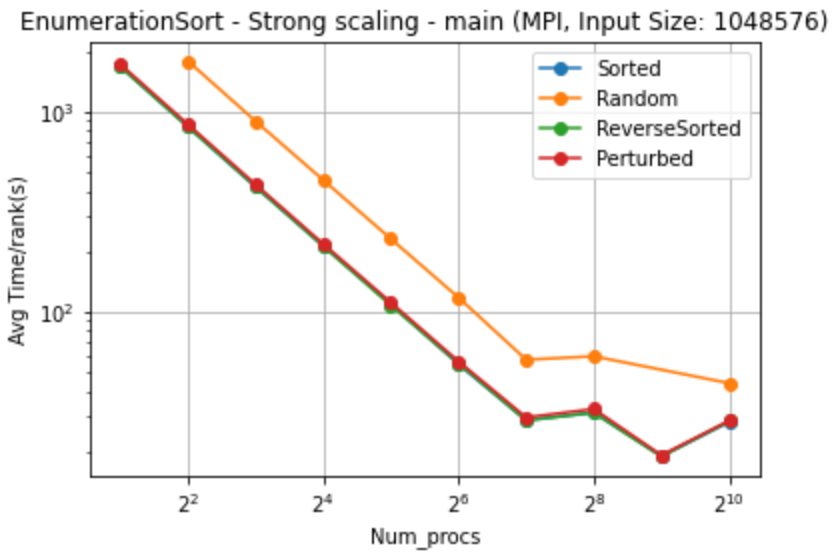

EnumerationSort - Strong scaling - comp_large (MPI, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_procs

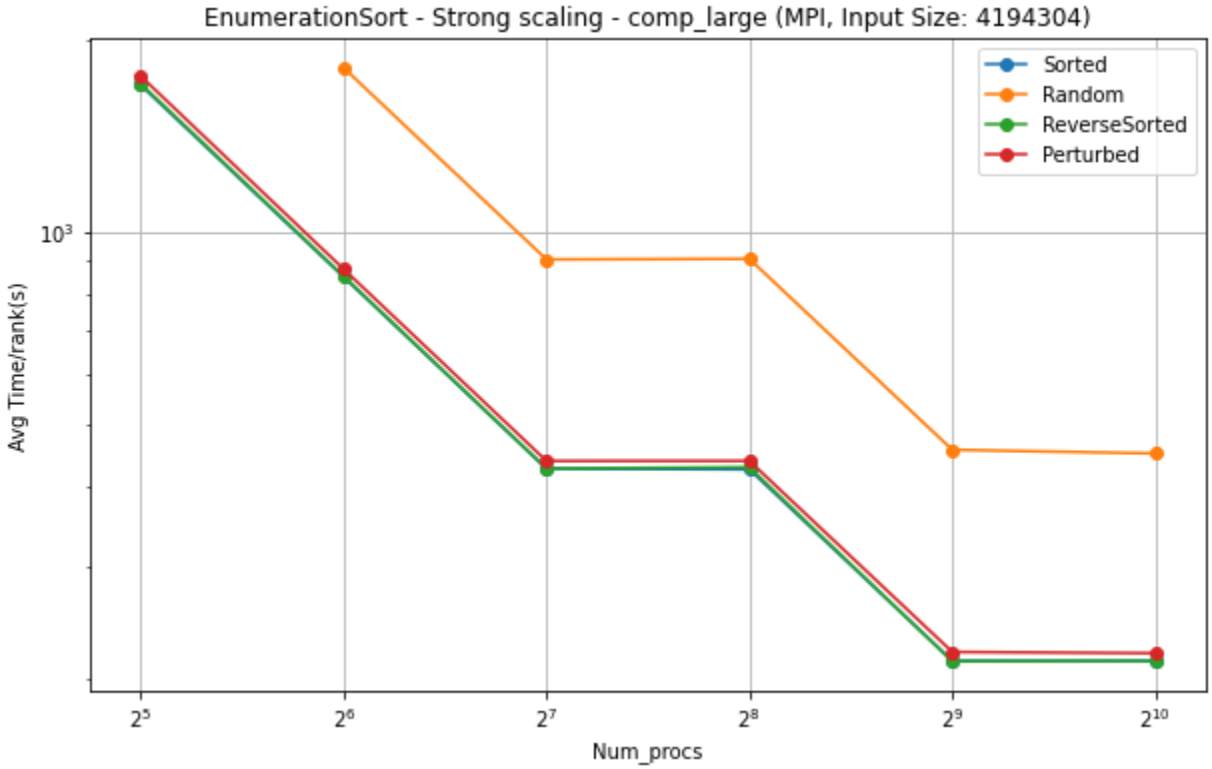

EnumerationSort - Strong scaling - comm (MPI, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{0}
10^{-1}
10^{-2}
2^{2}
2^{4}
2^{6}
2^{8}
2^{10}
Num_procs

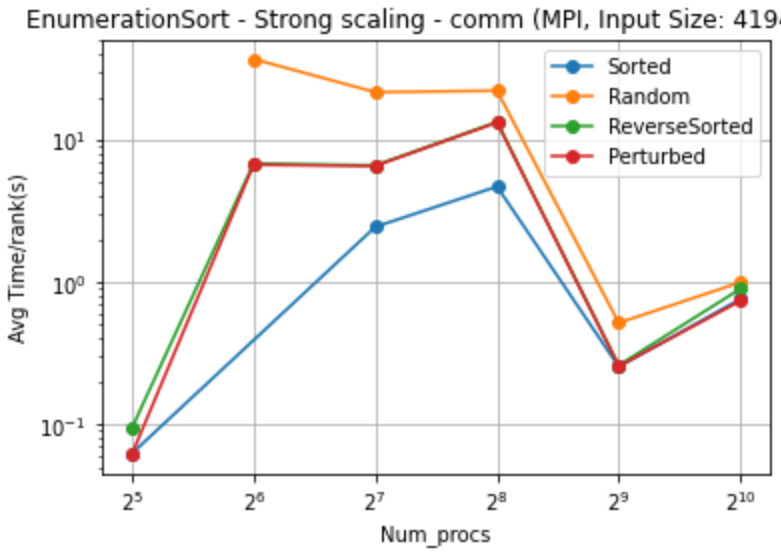

EnumerationSort - Strong scaling - main (MPI, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{2}
10^{1}
2^{2}
2^{4}
2^{6}
2^{8}
2^{10}
Num_procs

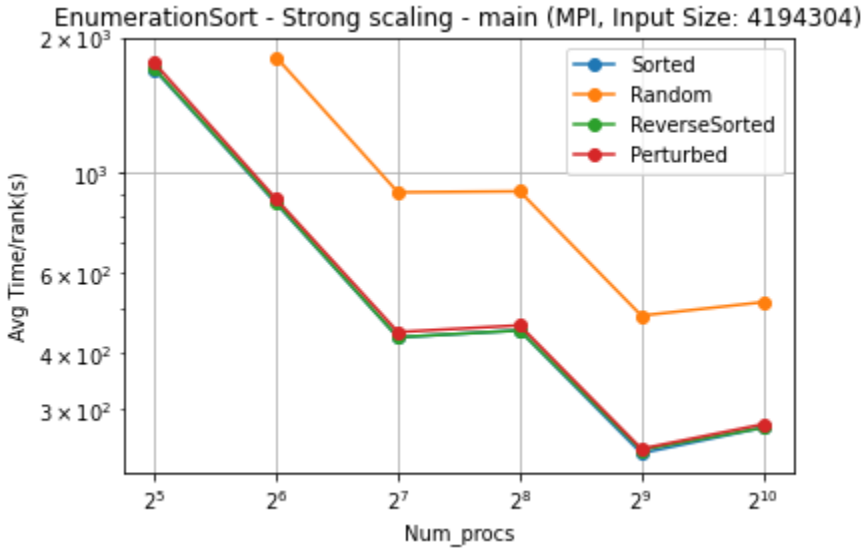

EnumerationSort - Strong scaling - comp_large (MPI, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^3
10^2
2^2
2^4
2^6
2^8
2^{10}
Num_procs


EnumerationSort - Strong scaling - comm (MPI, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^0
10^{-1}
10^{-2}
2^2
2^4
2^6
2^8
2^{10}
Num_procs

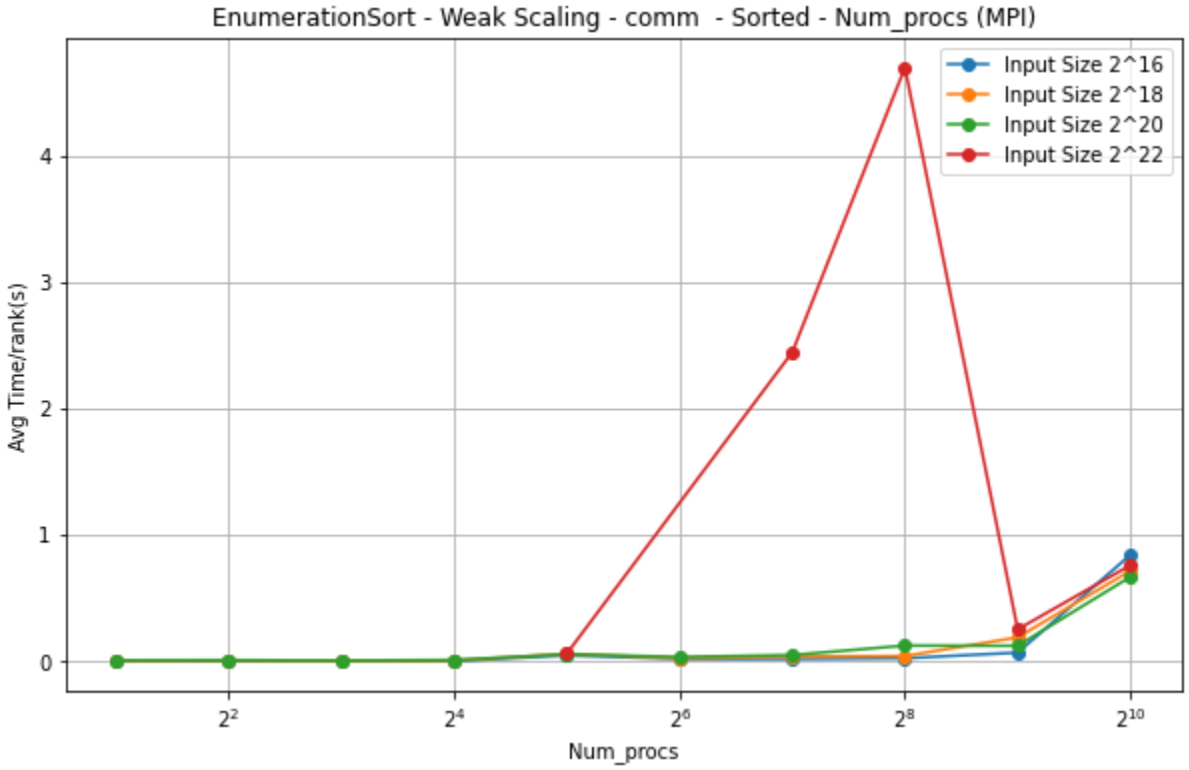

EnumerationSort - Strong scaling - main (MPI, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^3
10^2
2^2
2^4
2^6
2^8
2^{10}
Num_procs

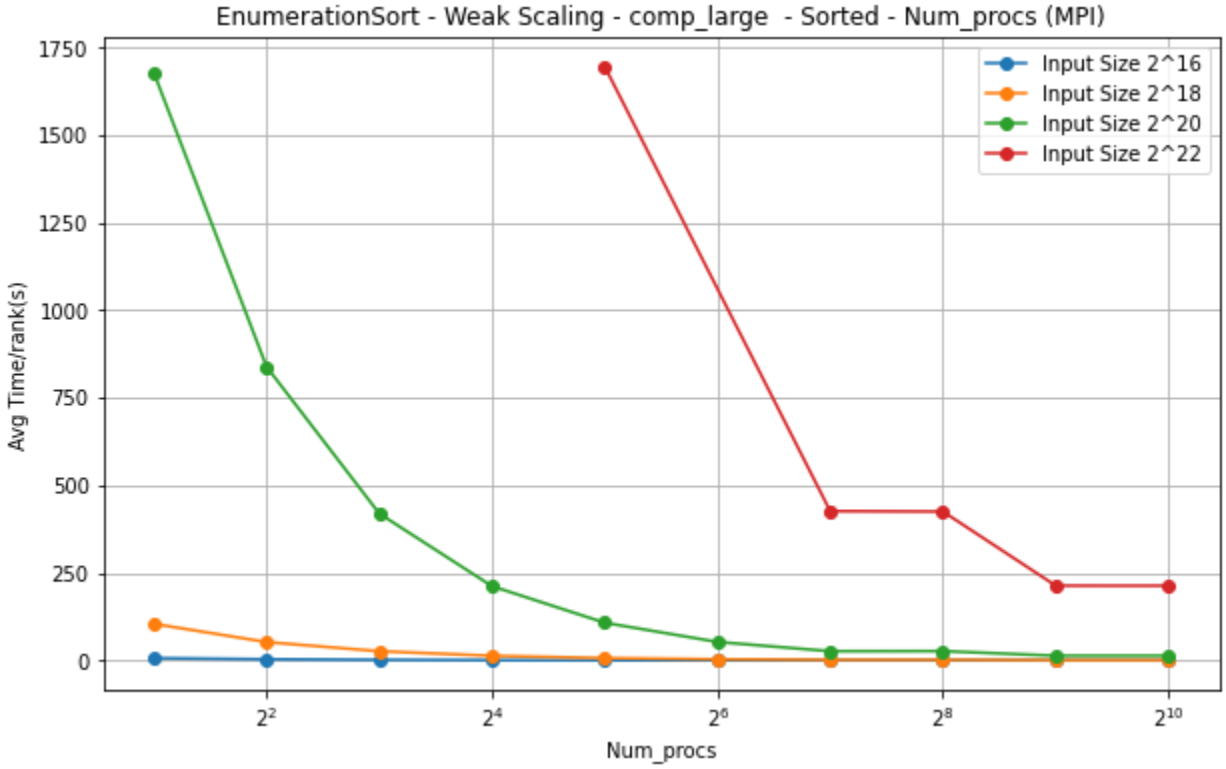

EnumerationSort - Strong scaling - comp_large (MPI, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^3
2^5
2^6
2^7
2^8
2^9
2^{10}
Num_procs

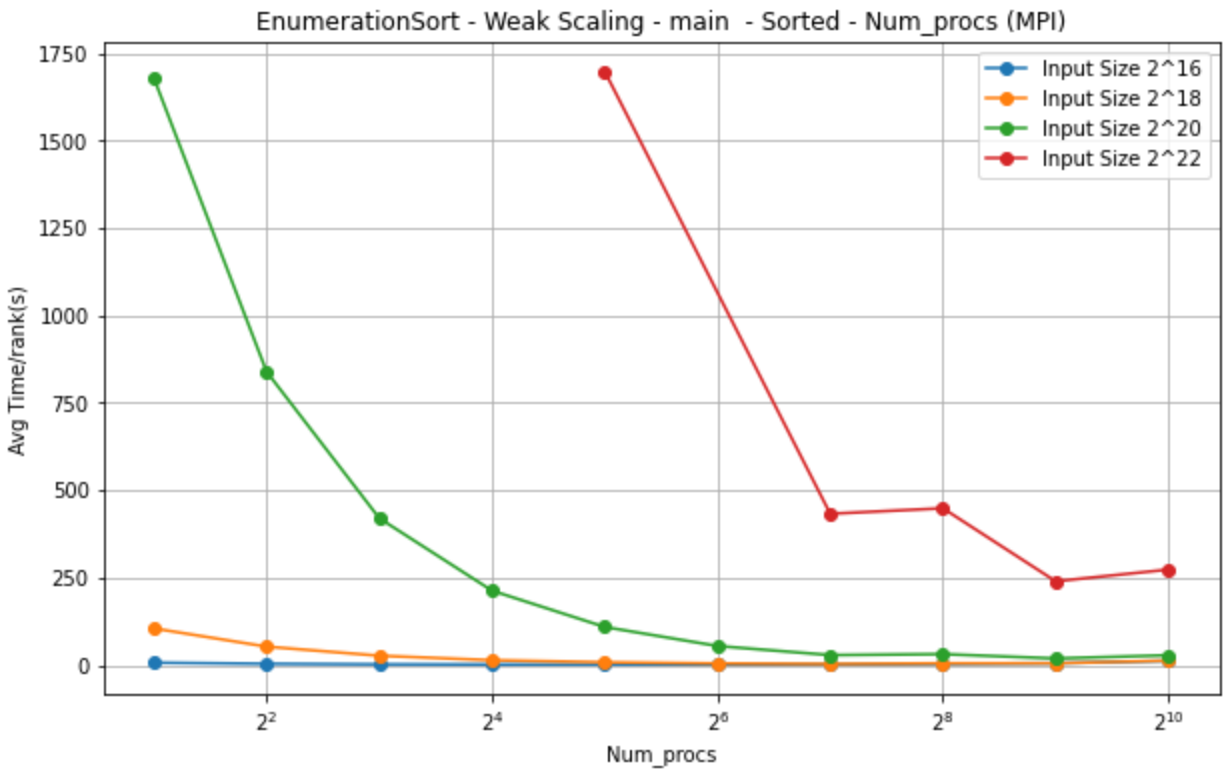

EnumerationSort - Strong scaling - comm (MPI, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
10^0
10^{-1}
2^5
2^6
2^7
2^8
2^9
2^{10}
Num_procs

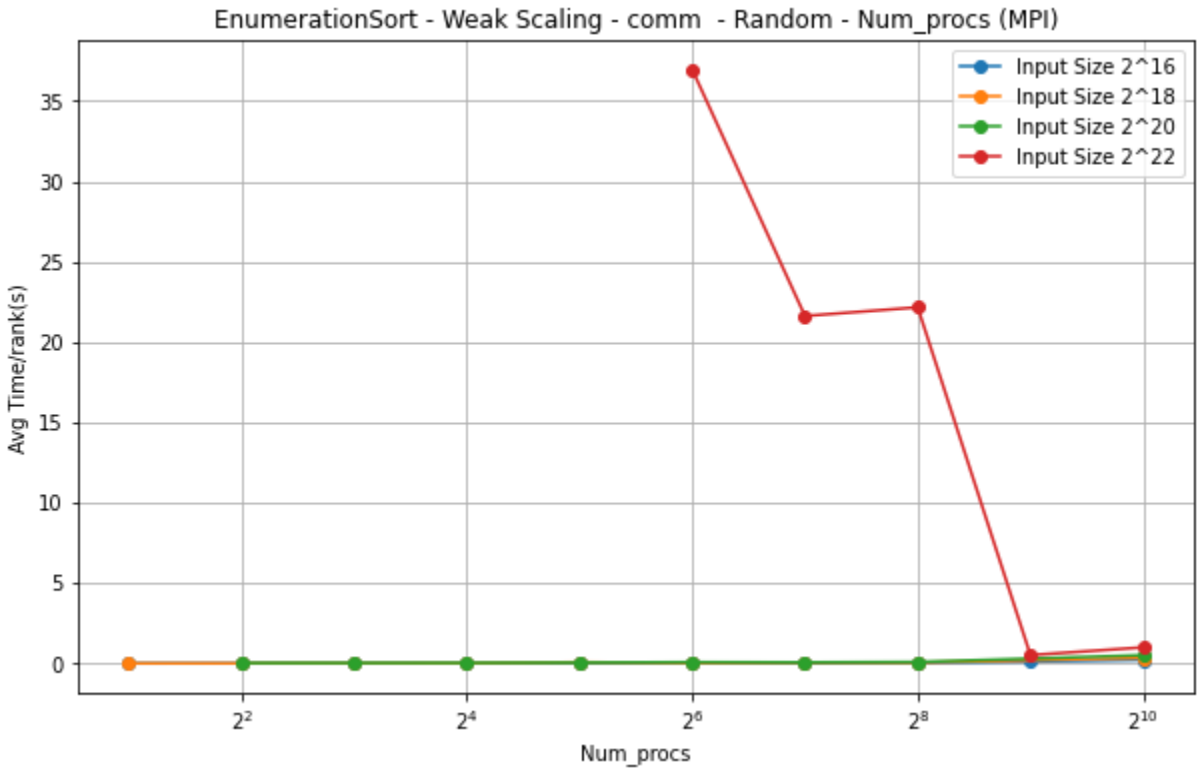

EnumerationSort - Strong scaling - main (MPI, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
2×10^3
10^3
6×10^2
4×10^2
3×10^2
2^5
2^6
2^7
2^8
2^9
2^{10}
Num_procs

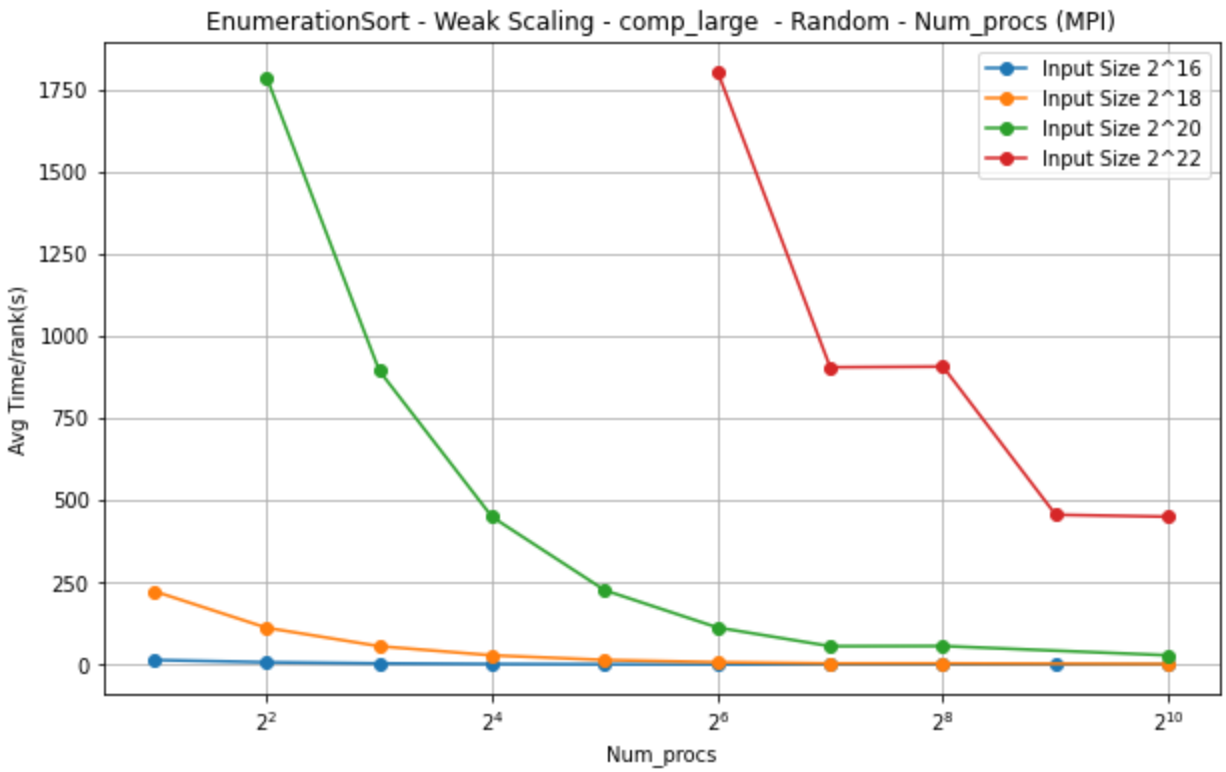
Weak Scaling:

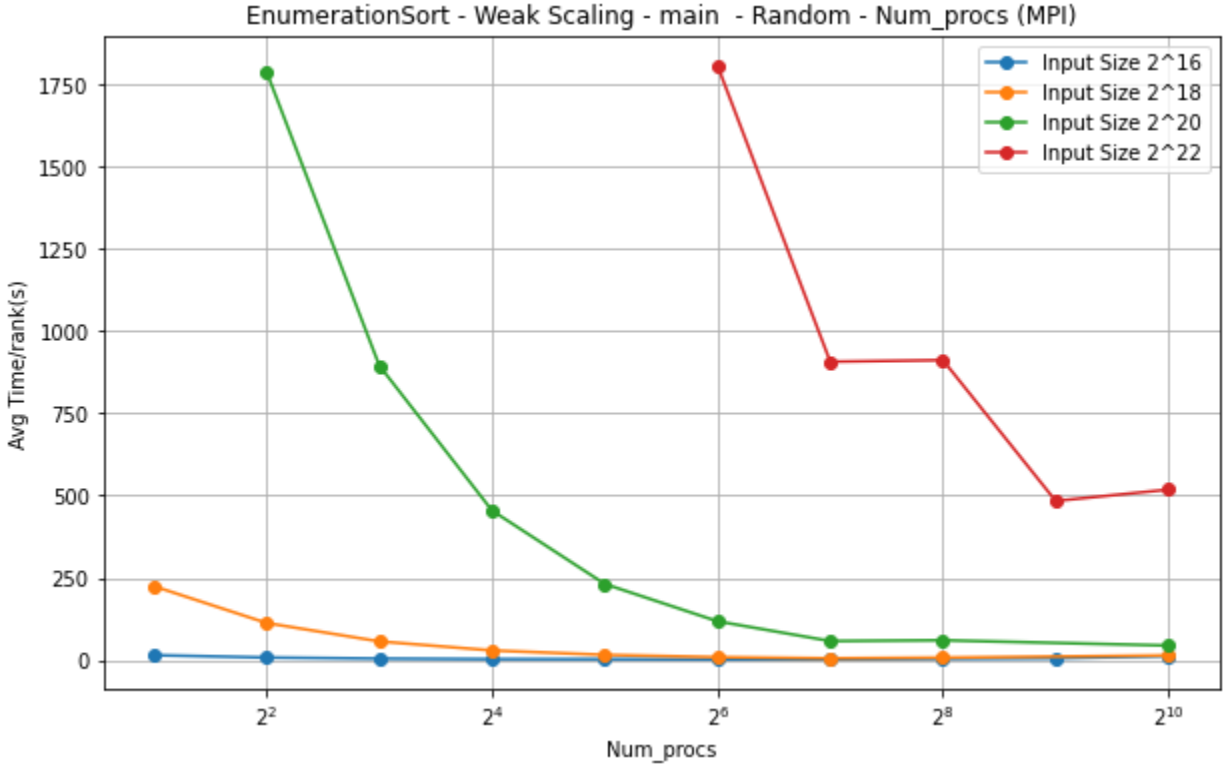


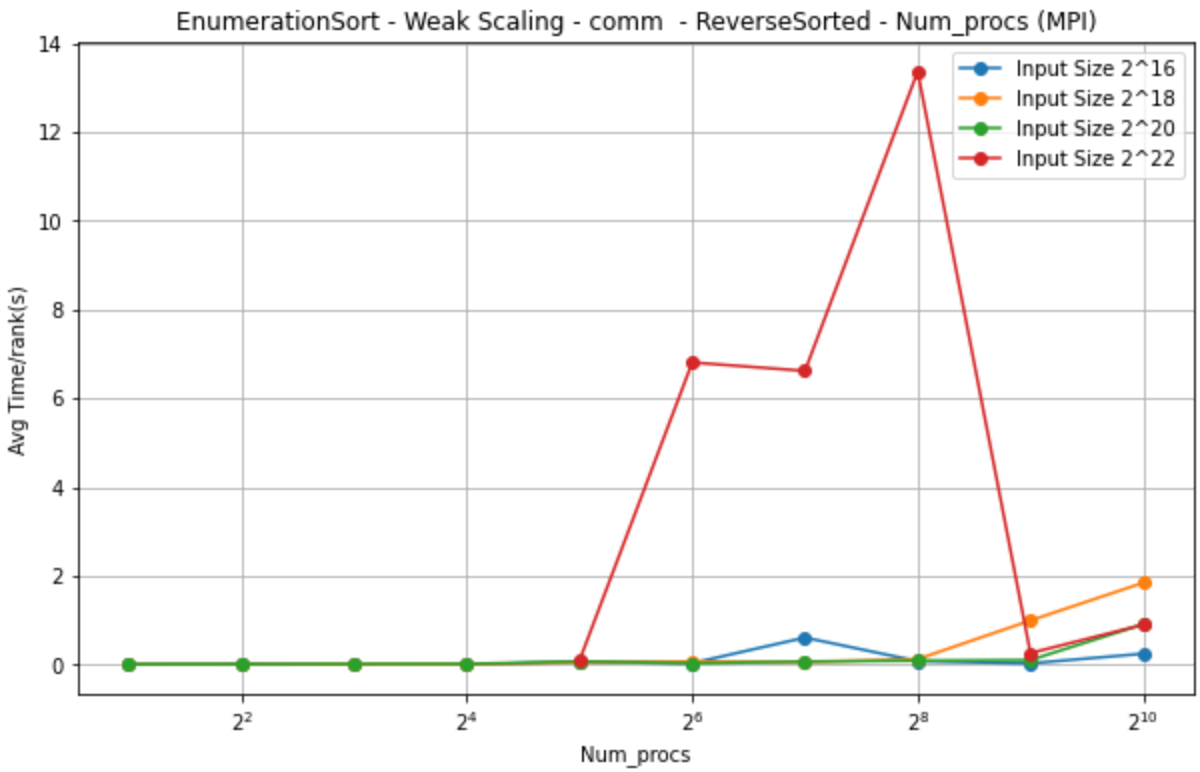


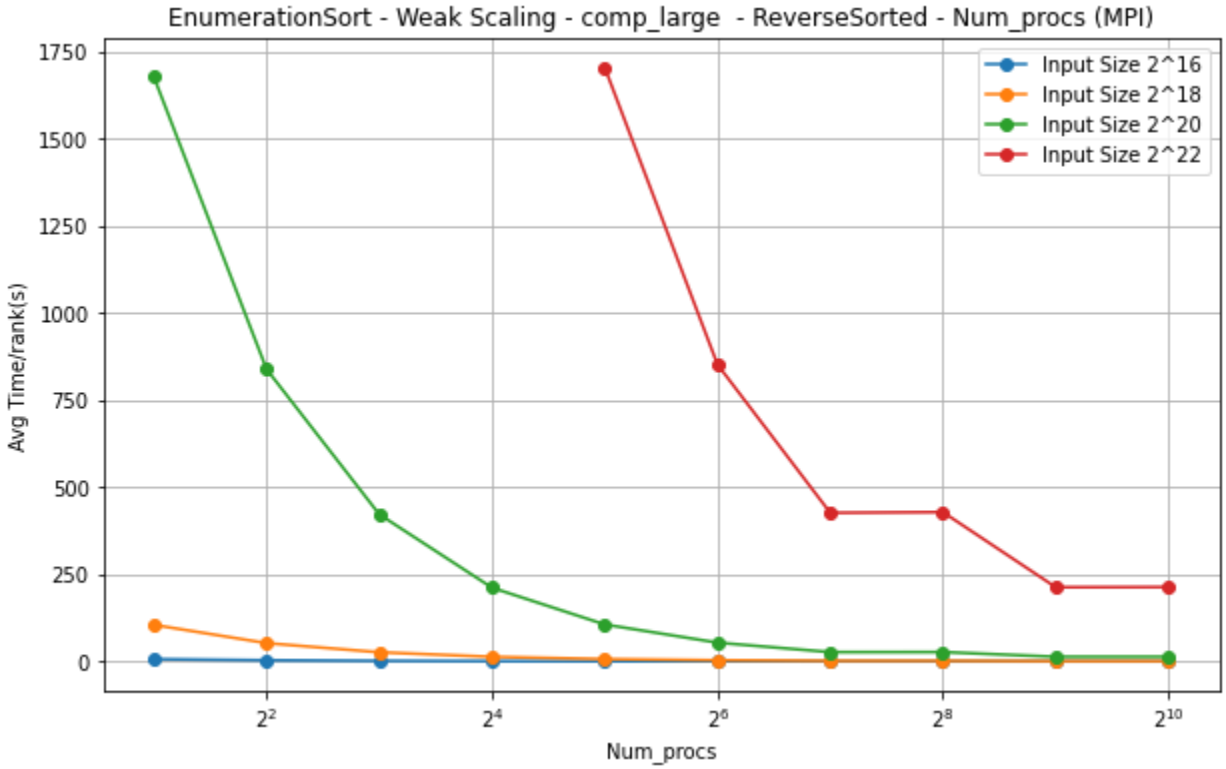

EnumerationSort - Weak Scaling - main - Sorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
Num_procs
0
250
500
750
1000
1250
1500
1750
2^2
2^4
2^6
2^8
2^{10}

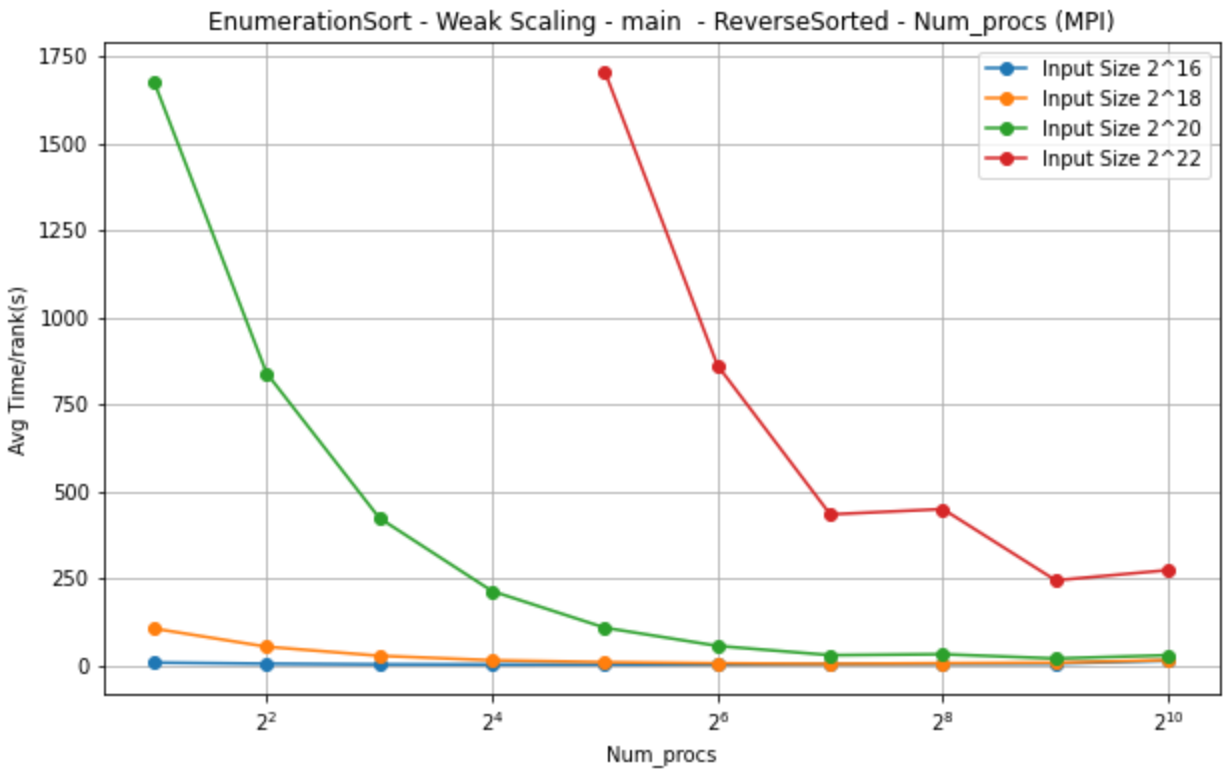

EnumerationSort - Weak Scaling - comm - Random - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
Num_procs
0
5
10
15
20
25
30
35
2^2
2^4
2^6
2^8
2^{10}

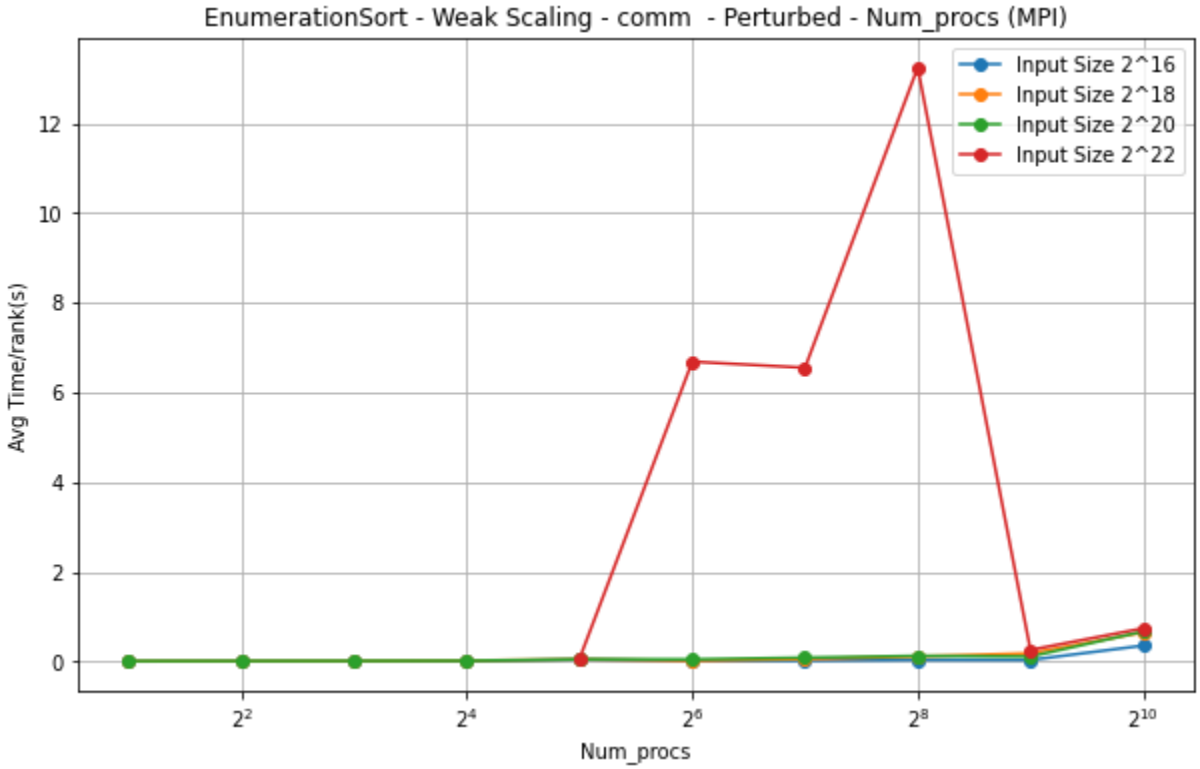

EnumerationSort - Weak Scaling - comp_large - Random - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
1750
1500
1250
1000
750
500
250
0
2^2
2^4
2^6
2^8
2^{10}
Num_procs

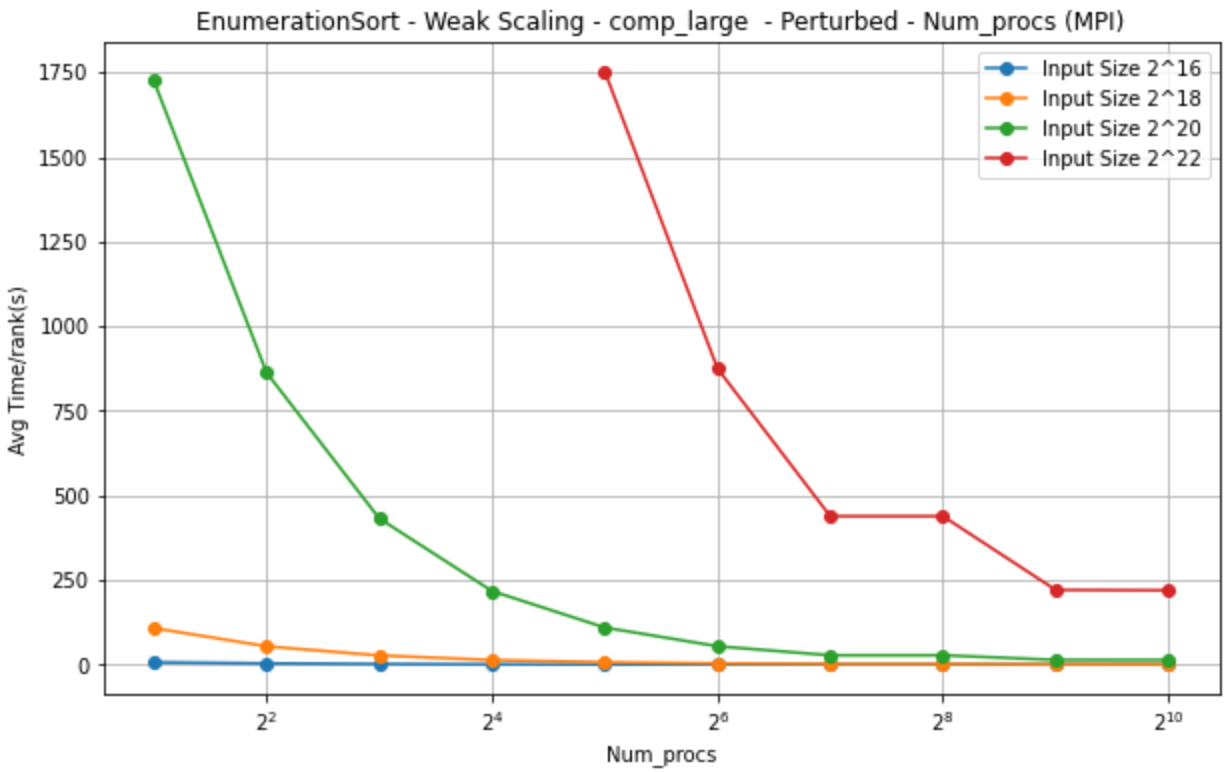

EnumerationSort - Weak Scaling - main - Random - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
1750
1500
1250
1000
750
500
250
0
2^2
2^4
2^6
2^8
2^{10}
Num_procs

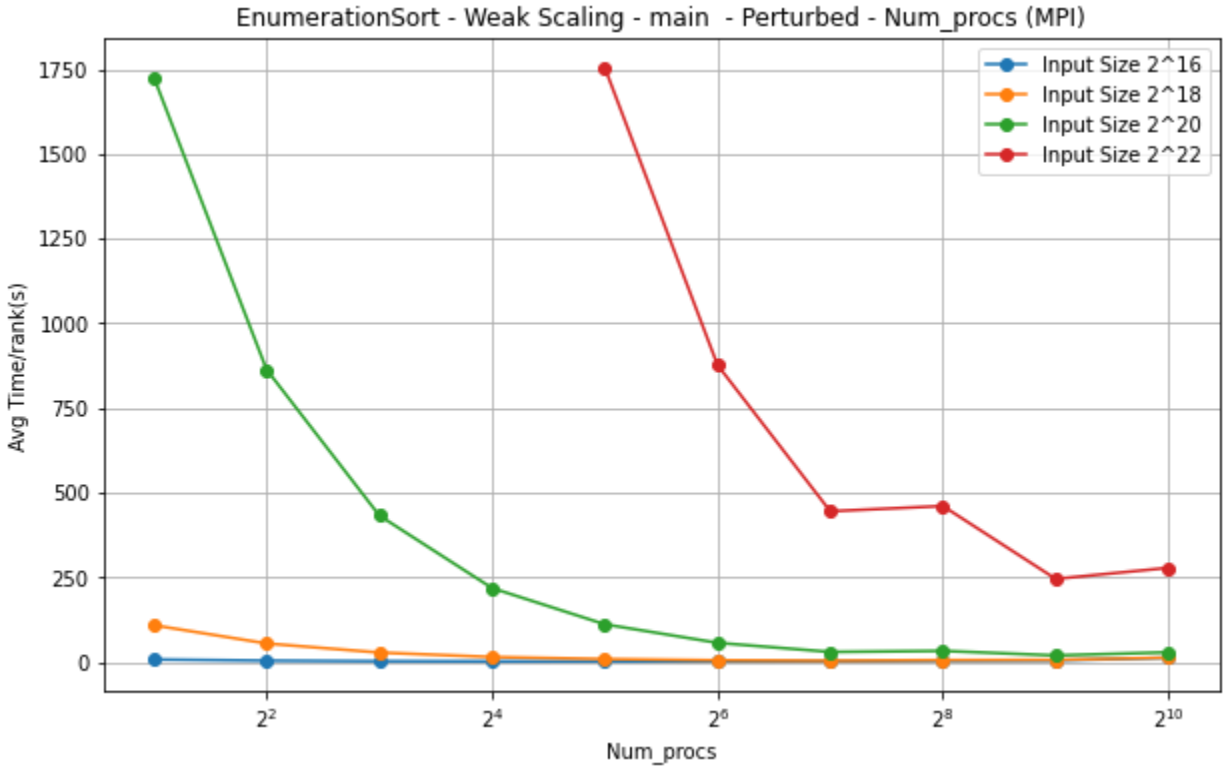

EnumerationSort - Weak Scaling - comm - ReverseSorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
Num_procs
0
2
4
6
8
10
12
14
2^2
2^4
2^6
2^8
2^{10}


EnumerationSort - Weak Scaling - comp_large - ReverseSorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
Num_procs
0
250
500
750
1000
1250
1500
1750
2^2
2^4
2^6
2^8
2^{10}

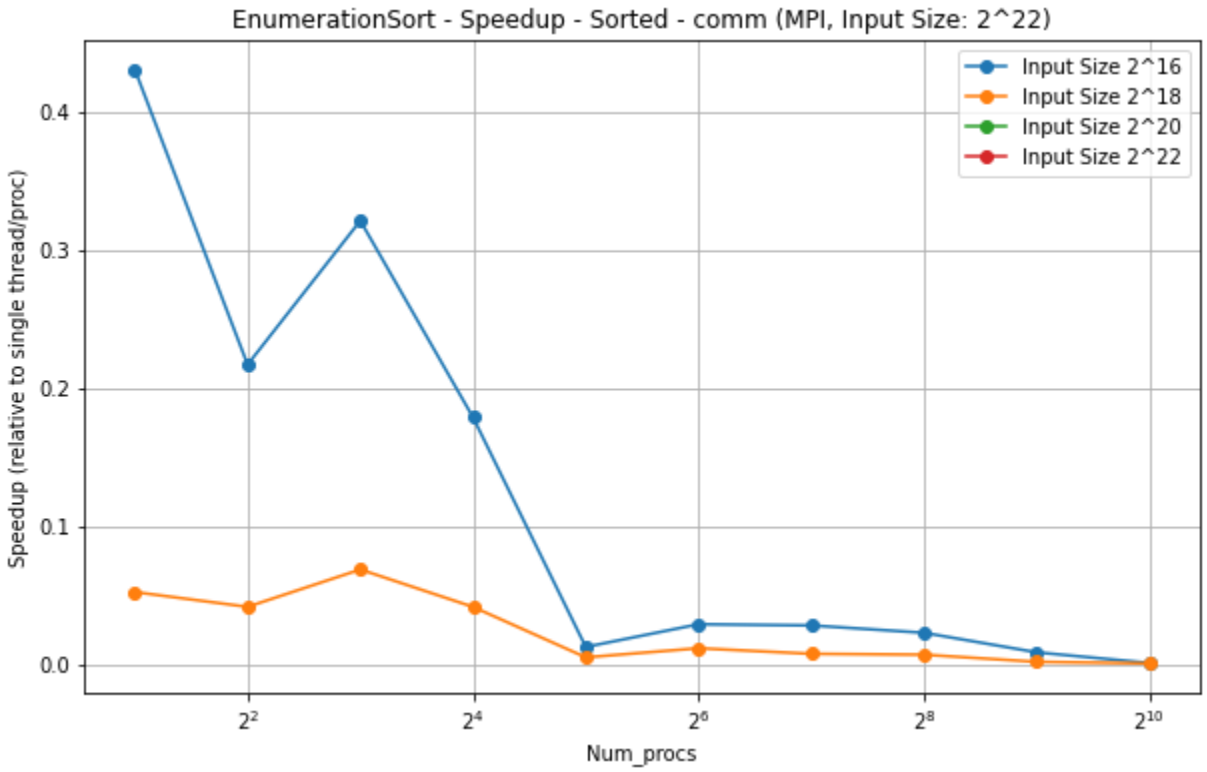

EnumerationSort - Weak Scaling - main - ReverseSorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
Num_procs
0
250
500
750
1000
1250
1500
1750
2^2
2^4
2^6
2^8
2^{10}

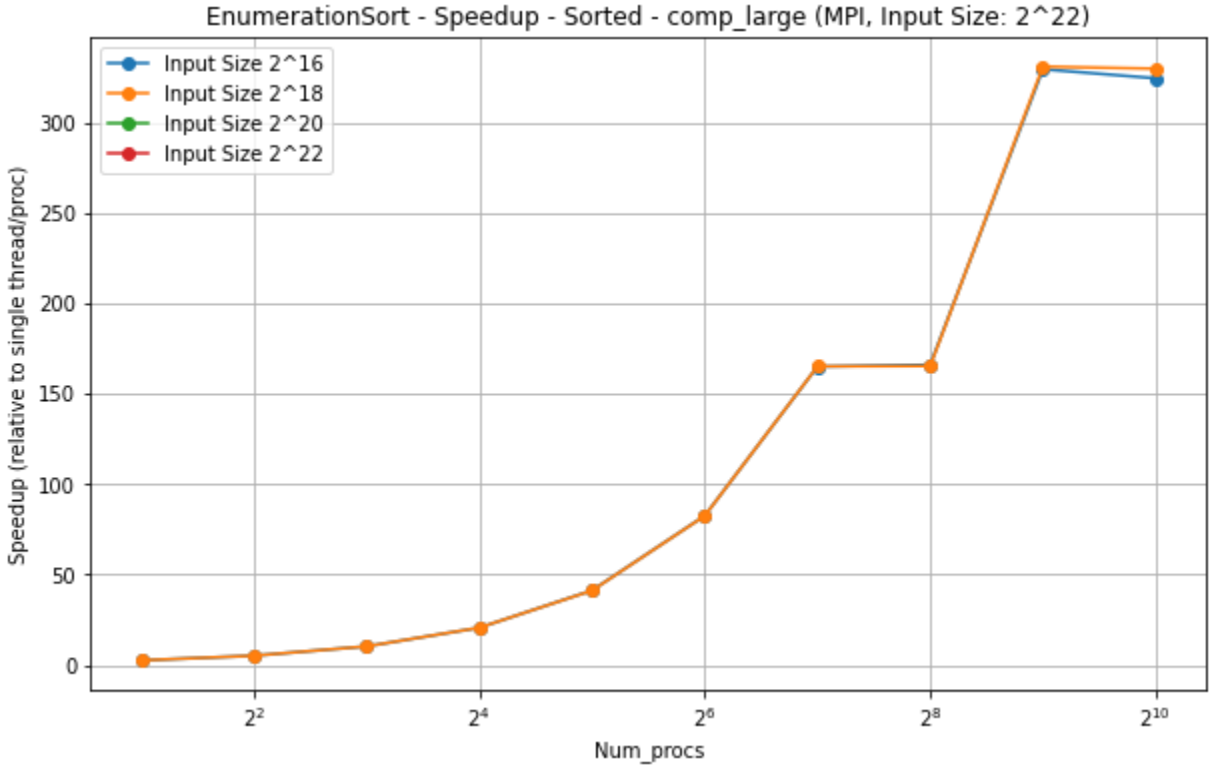

EnumerationSort - Weak Scaling - comm - Perturbed - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
Num_procs
0
2
4
6
8
10
12
2^2
2^4
2^6
2^8
2^{10}

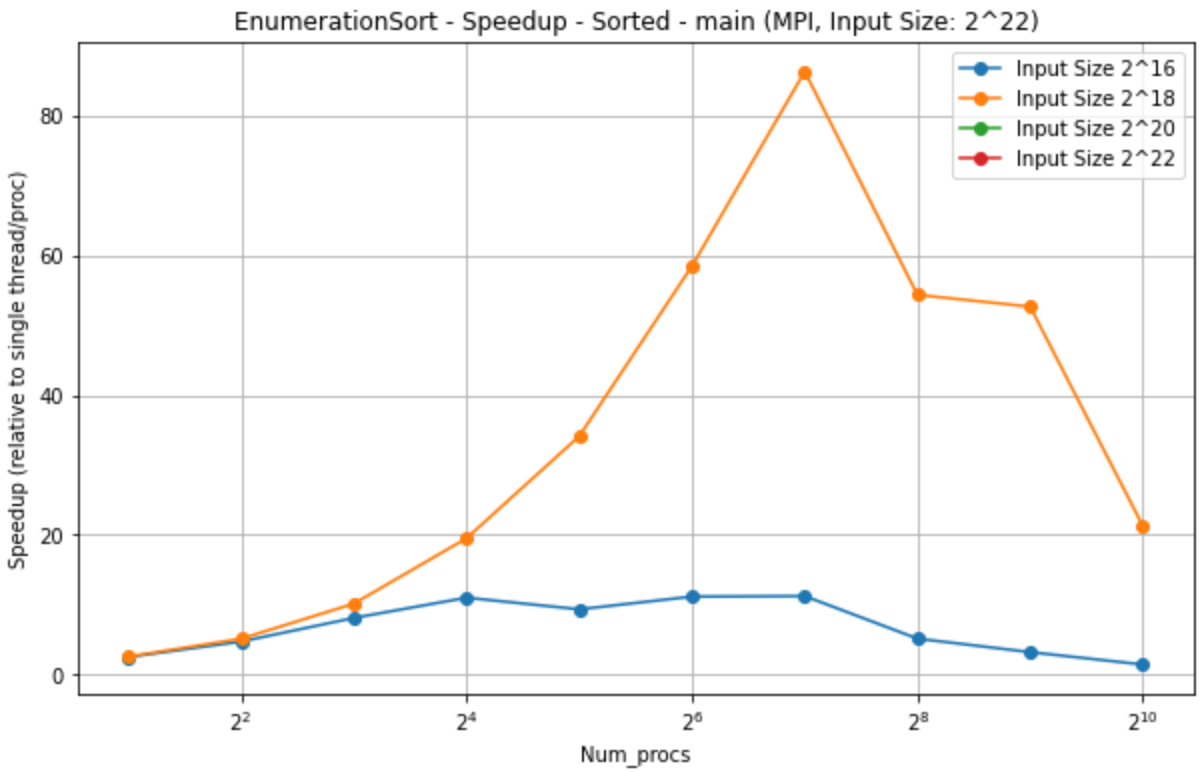

EnumerationSort - Weak Scaling - comp_large - Perturbed - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
1750
1500
1250
1000
750
500
250
0
2^2
2^4
2^6
2^8
2^{10}
Num_procs

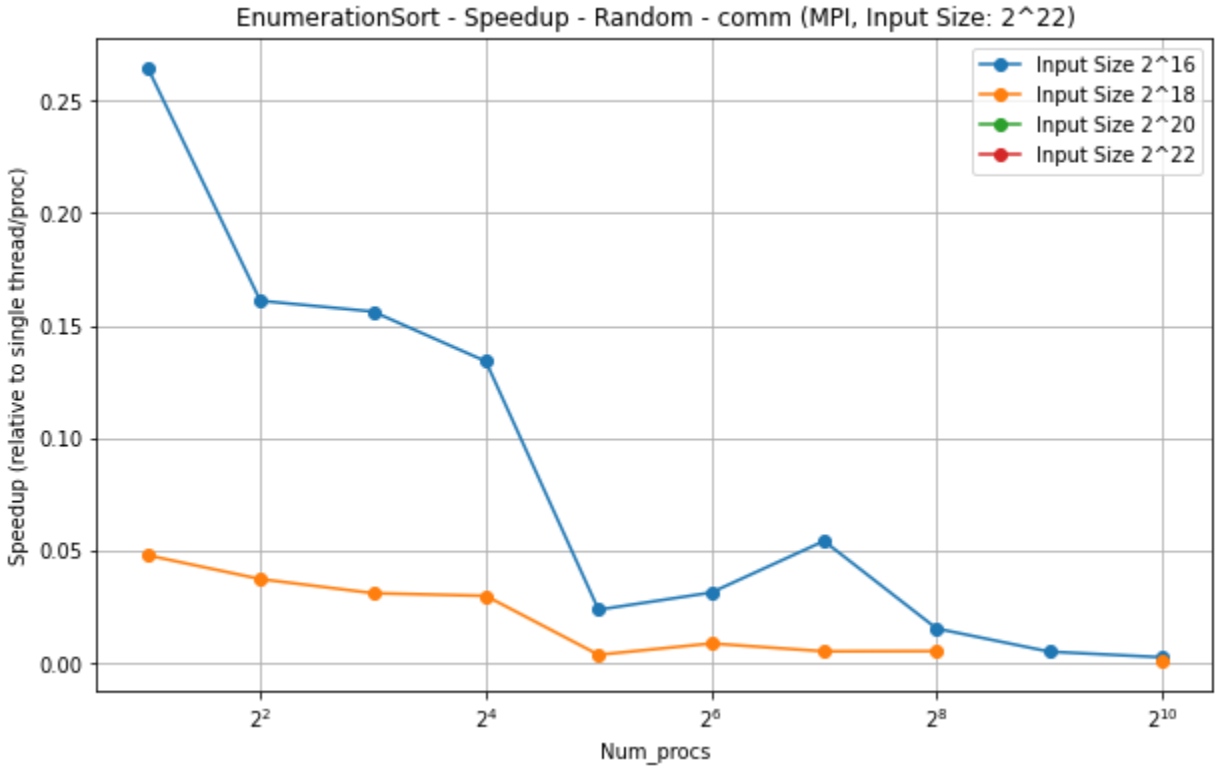

EnumerationSort - Weak Scaling - main - Perturbed - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
1750
1500
1250
1000
750
500
250
0
2^2
2^4
2^6
2^8
2^{10}
Num_procs

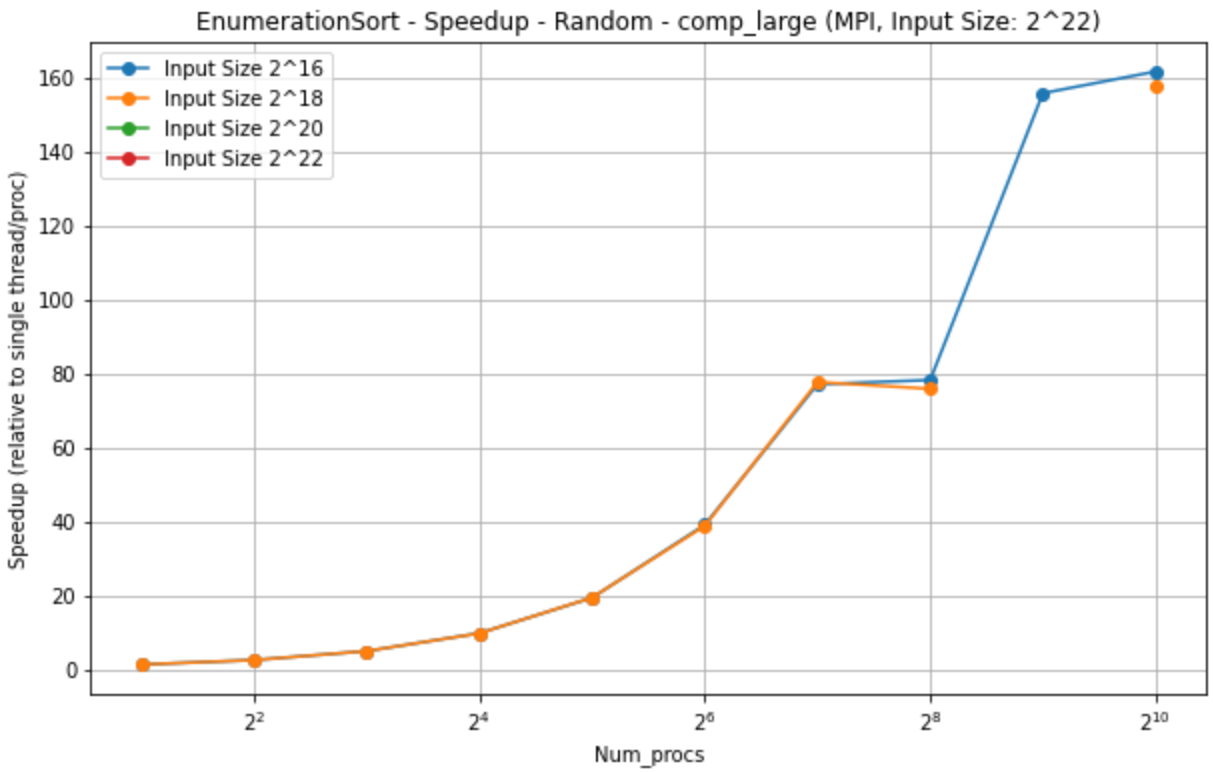
Speedup:

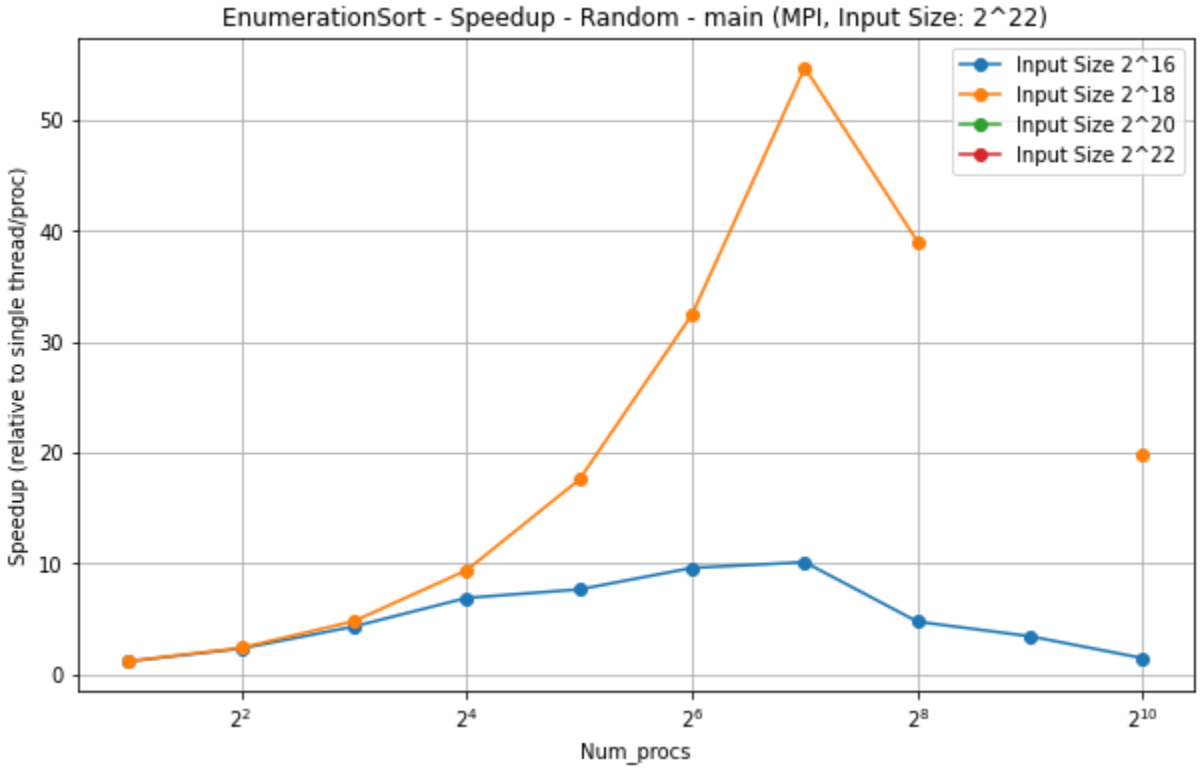

EnumerationSort - Speedup - Sorted - comm (MPI, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
0.0
0.1
0.2
0.3
0.4
2^2
2^4
2^6
2^8
2^{10}
Num_procs

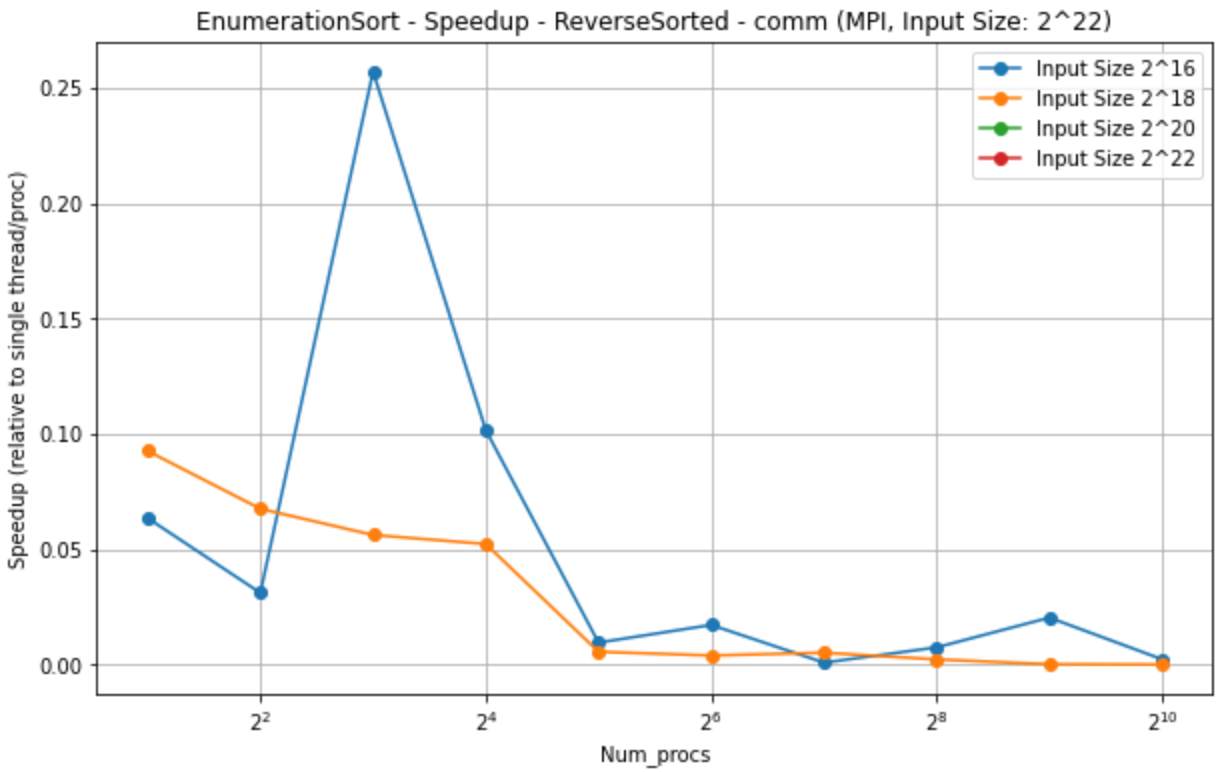

EnumerationSort - Speedup - Sorted - comp_large (MPI, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
0
50
100
150
200
250
300
2^2
2^4
2^6
2^8
2^{10}
Num_procs

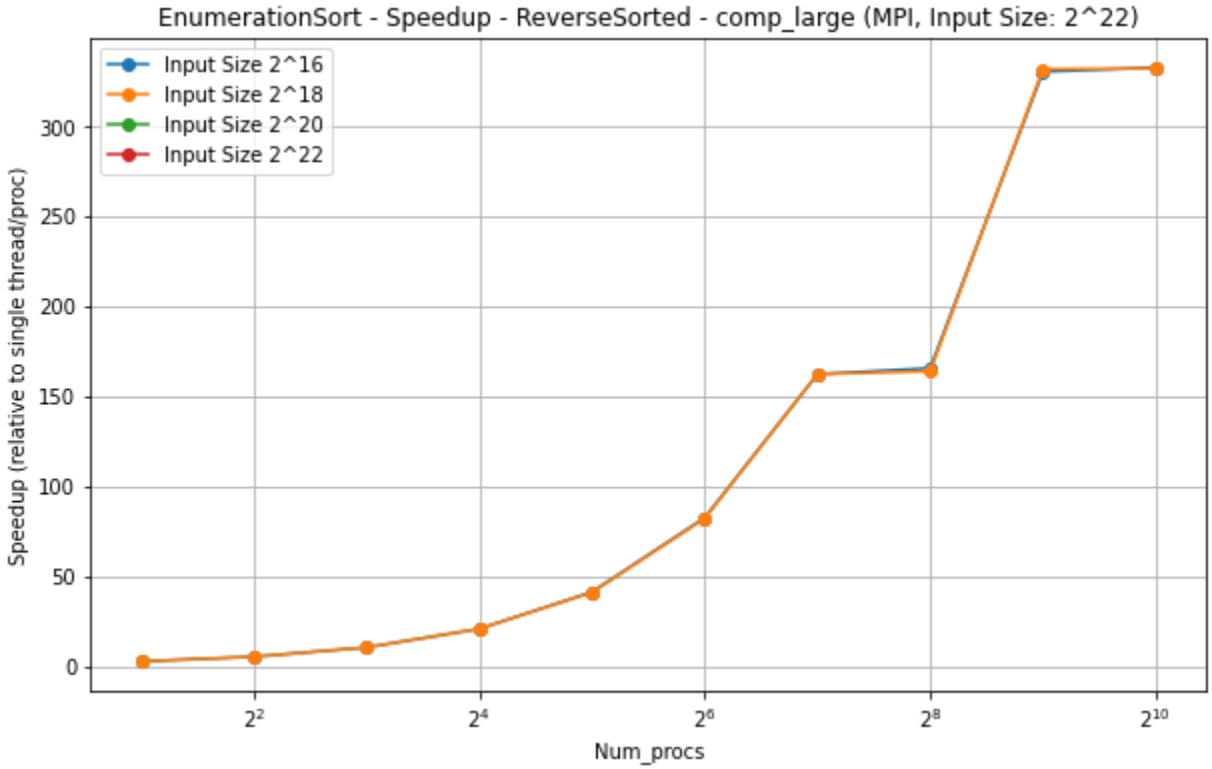

EnumerationSort - Speedup - Sorted - main (MPI, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
0
20
40
60
80
2^2
2^4
2^6
2^8
2^{10}
Num_procs

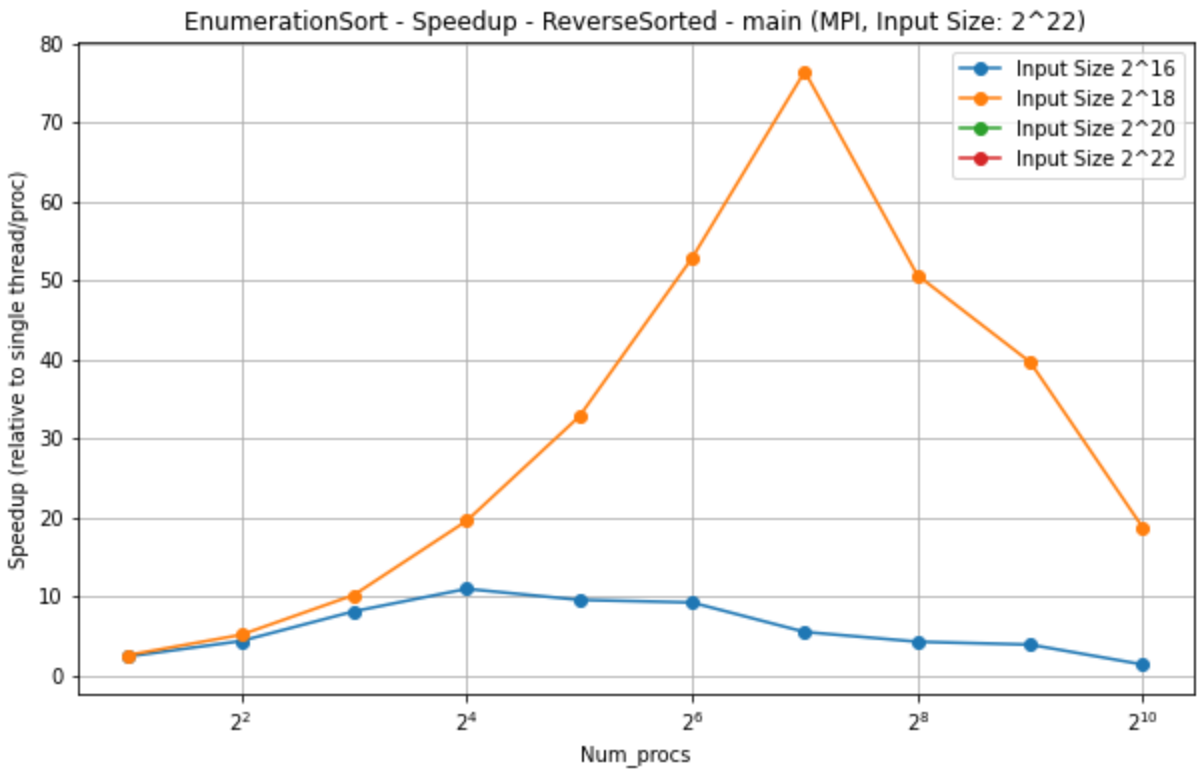

EnumerationSort - Speedup - Random - comm (MPI, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
0.00
0.05
0.10
0.15
0.20
0.25
2^2
2^4
2^6
2^8
2^{10}
Num_procs

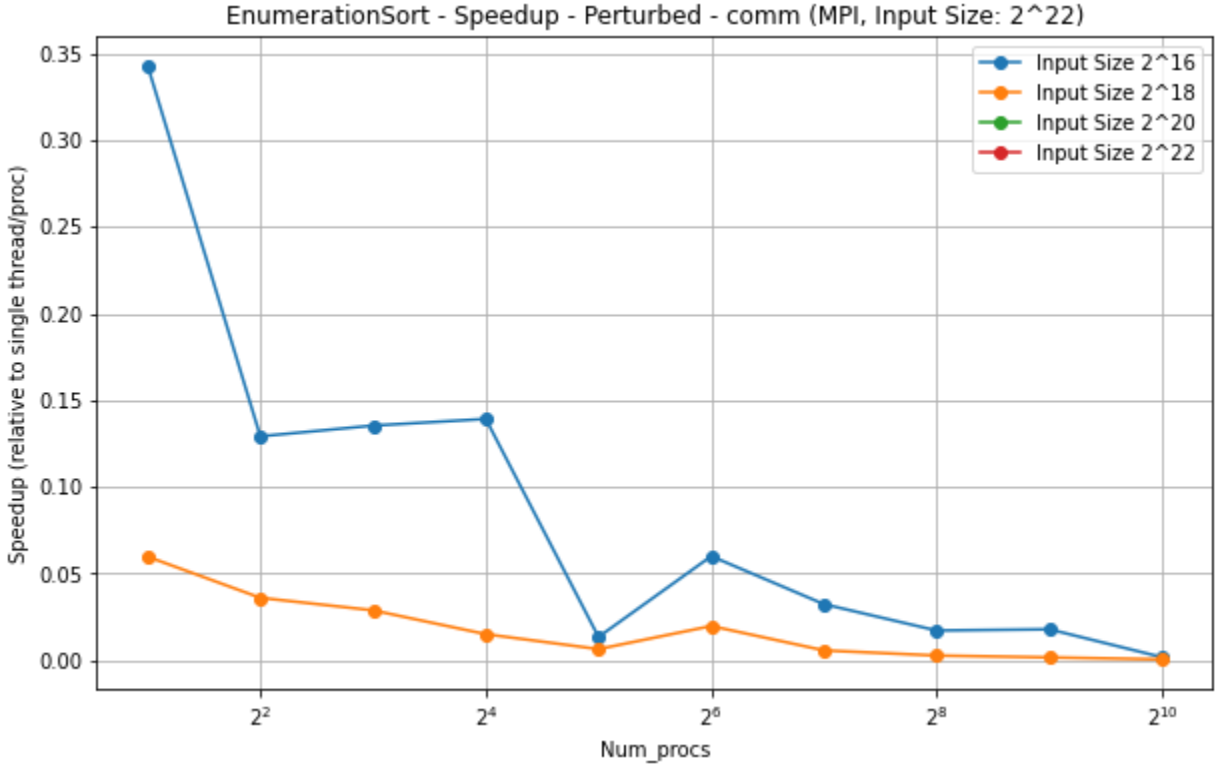

EnumerationSort - Speedup - Random - comp_large (MPI, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
Num_procs

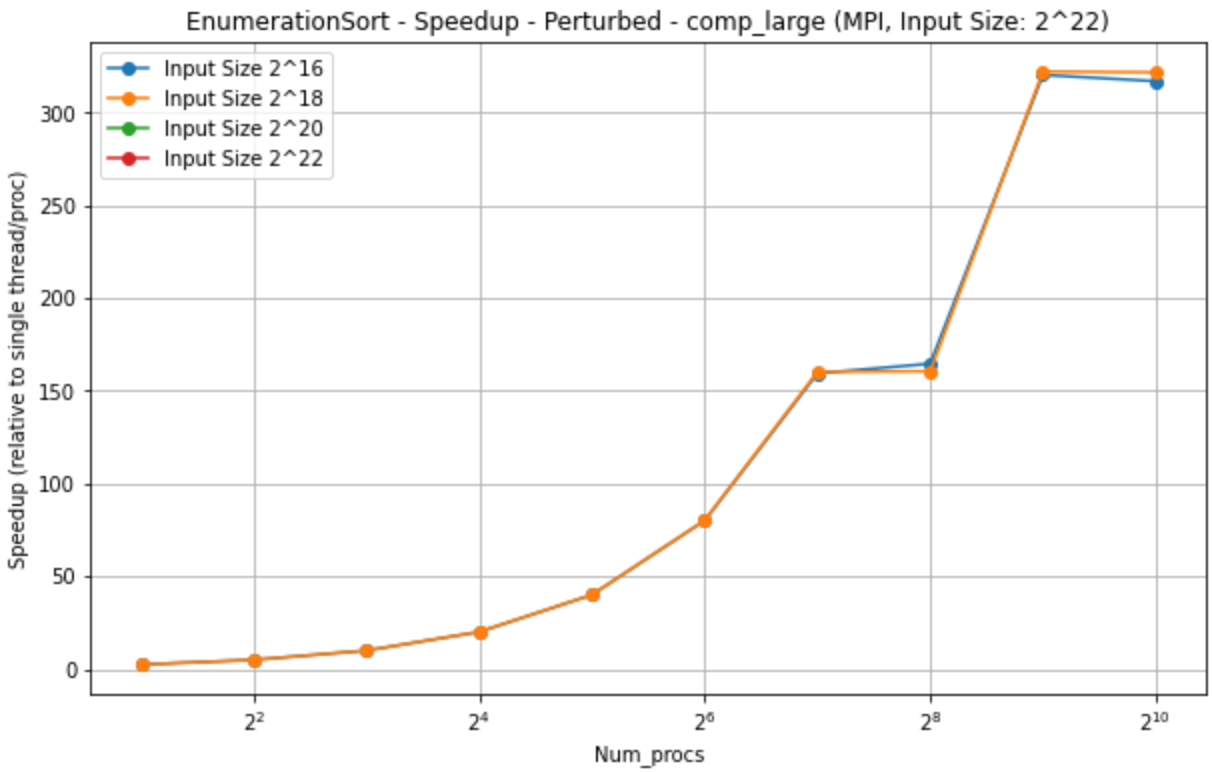

EnumerationSort - Speedup - Random - main (MPI, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
Num_procs

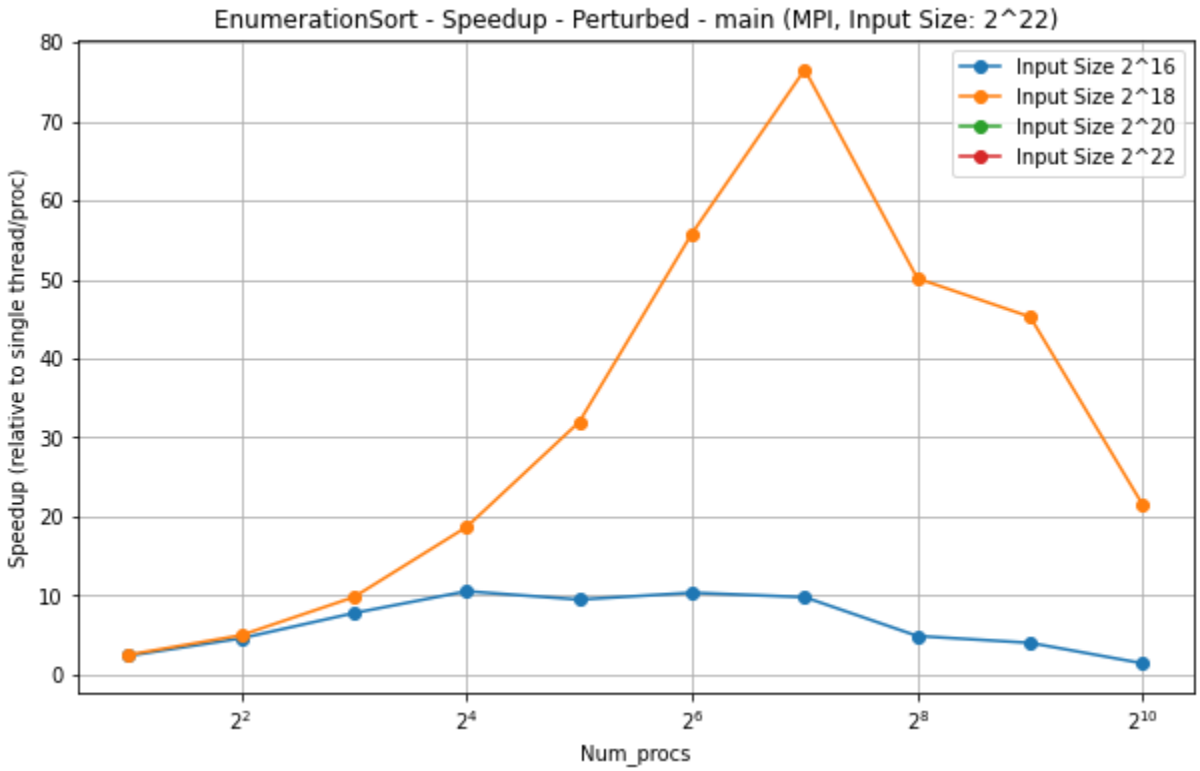

EnumerationSort - Speedup - ReverseSorted - comm (MPI, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
0.25
0.20
0.15
0.10
0.05
0.00
2^2
2^4
2^6
2^8
2^{10}
Num_procs


EnumerationSort - Speedup - ReverseSorted - comp_large (MPI, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
300
250
200
150
100
50
0
2^2
2^4
2^6
2^8
2^{10}
Num_procs


EnumerationSort - Speedup - ReverseSorted - main (MPI, Input Size: 2^22)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
0
10
20
30
40
50
60
70
80
2^2
2^4
2^6
2^8
2^{10}

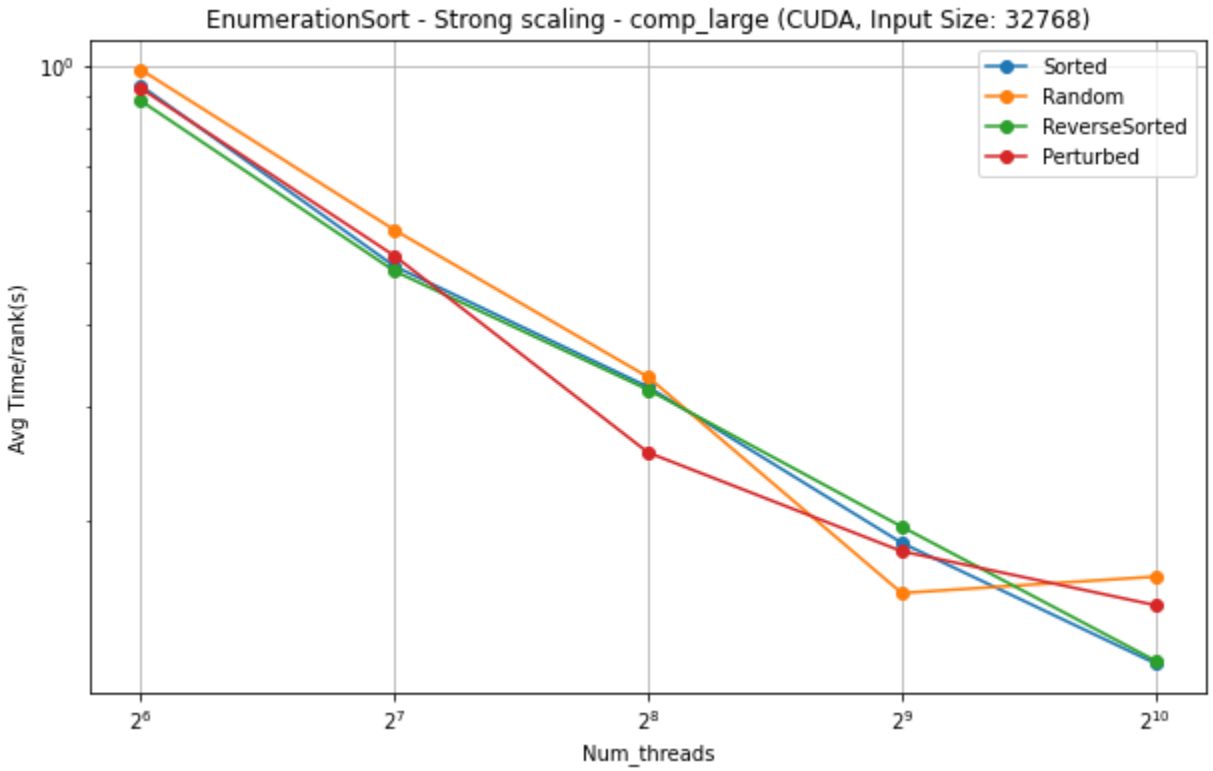

EnumerationSort - Speedup - Perturbed - comm (MPI, Input Size: 2^22)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
2^2
2^4
2^6
2^8
2^{10}

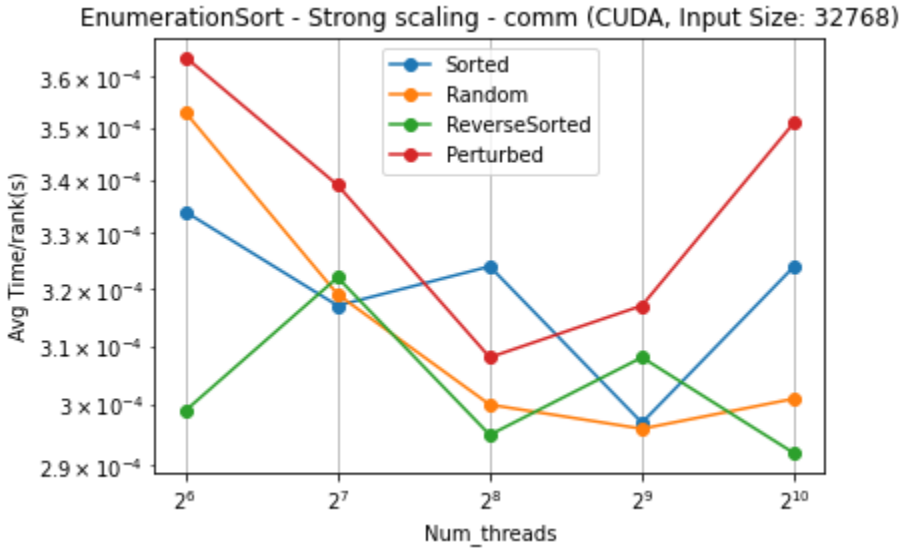

EnumerationSort - Speedup - Perturbed - comp_large (MPI, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
Num_procs

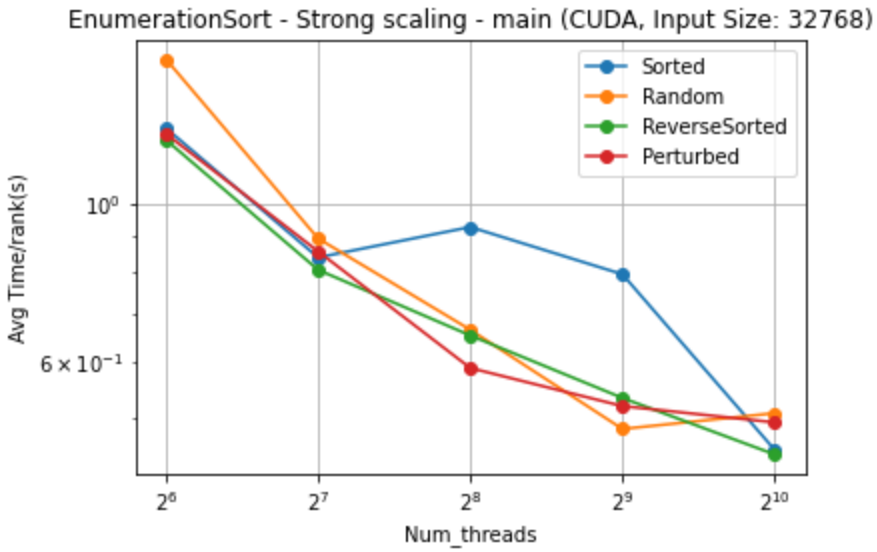

EnumerationSort - Speedup - Perturbed - main (MPI, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
Num_procs

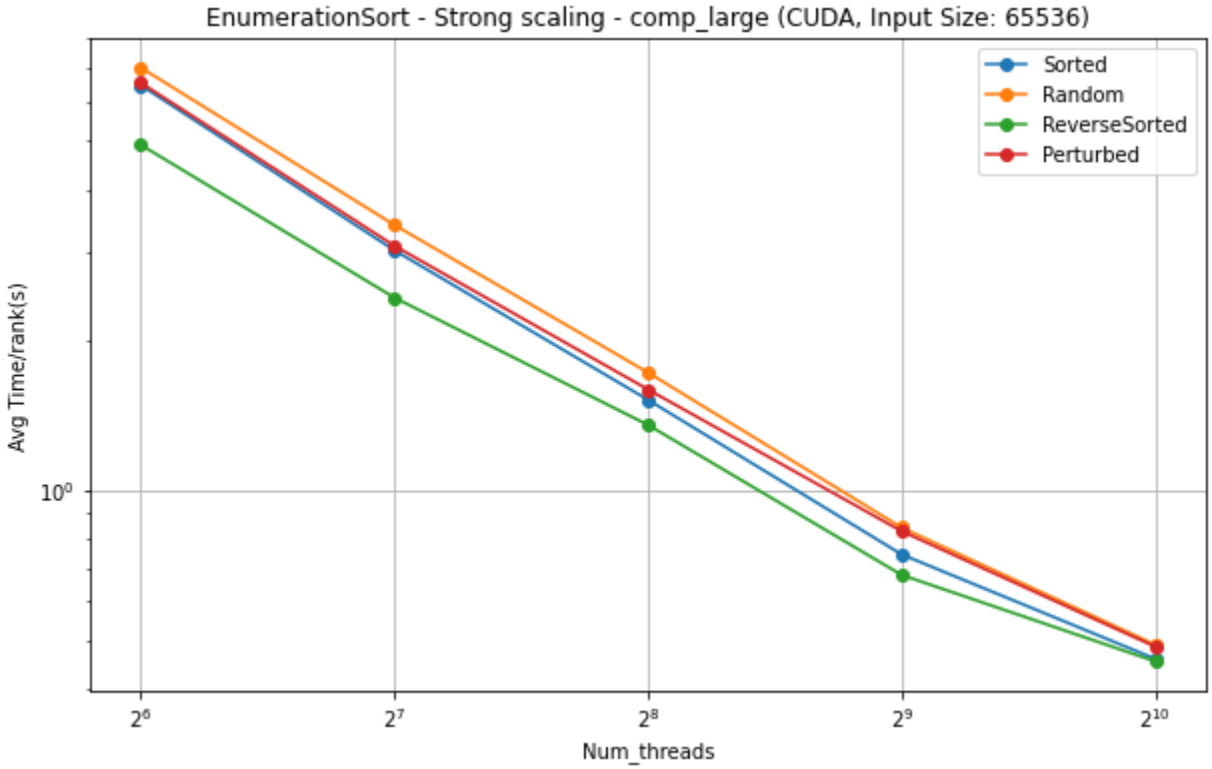
CUDA:

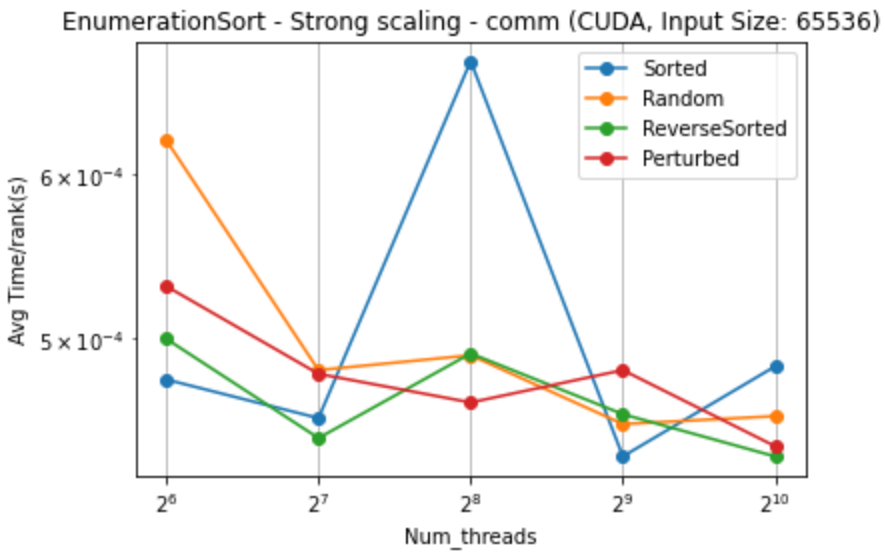
Strong Scaling:

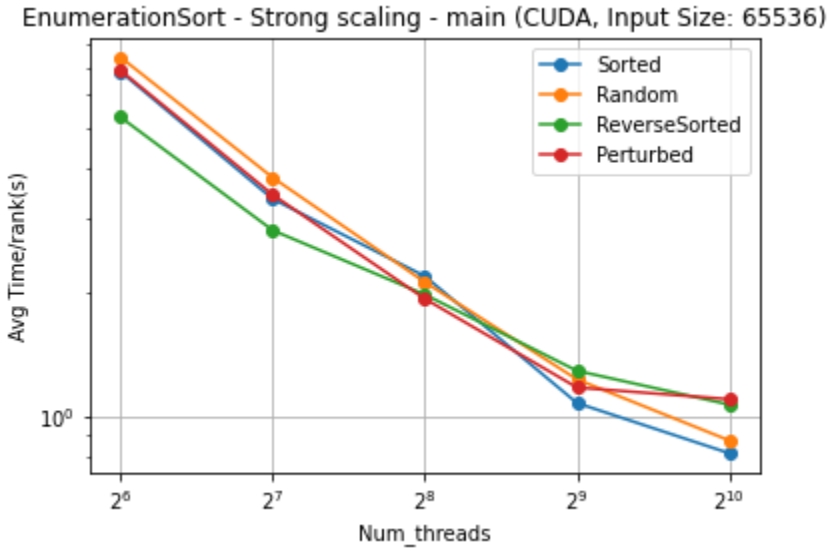


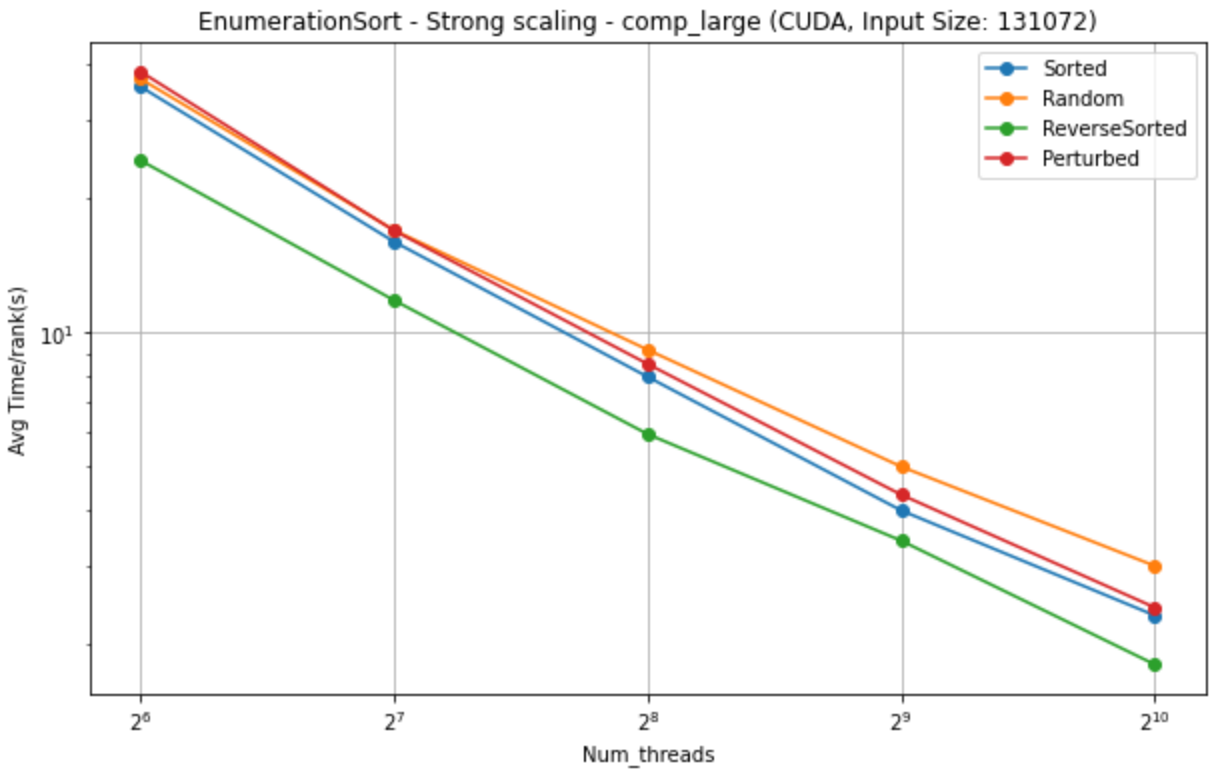


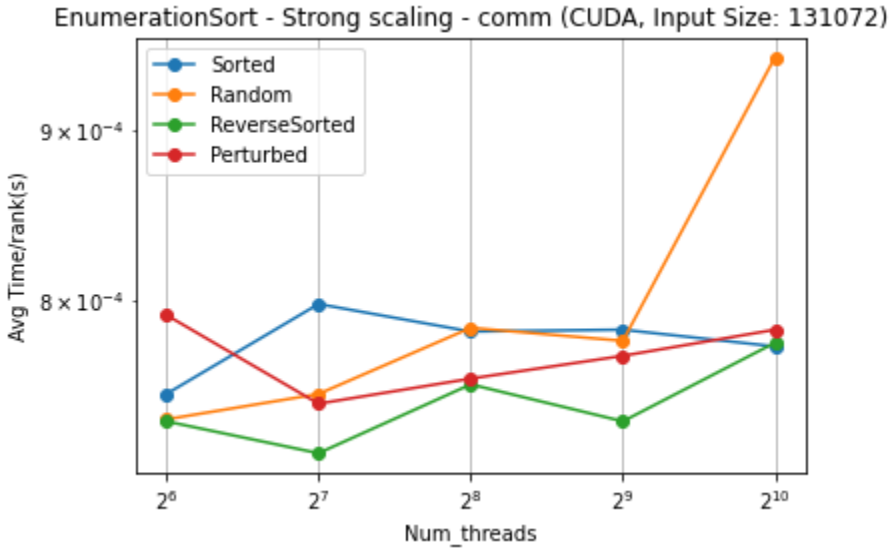

EnumerationSort - Strong scaling - main (CUDA, Input Size: 32768)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^0
6×10^{-1}
2^6
2^7
2^8
2^9
2^{10}
Num_threads

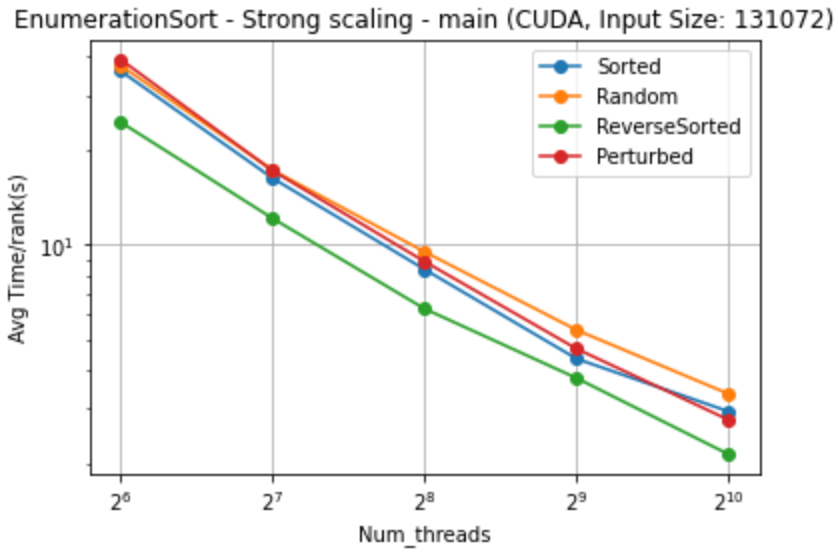

EnumerationSort - Strong scaling - comp_large (CUDA, Input Size: 65536)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^0
2^6
2^7
2^8
2^9
2^{10}
Num_threads

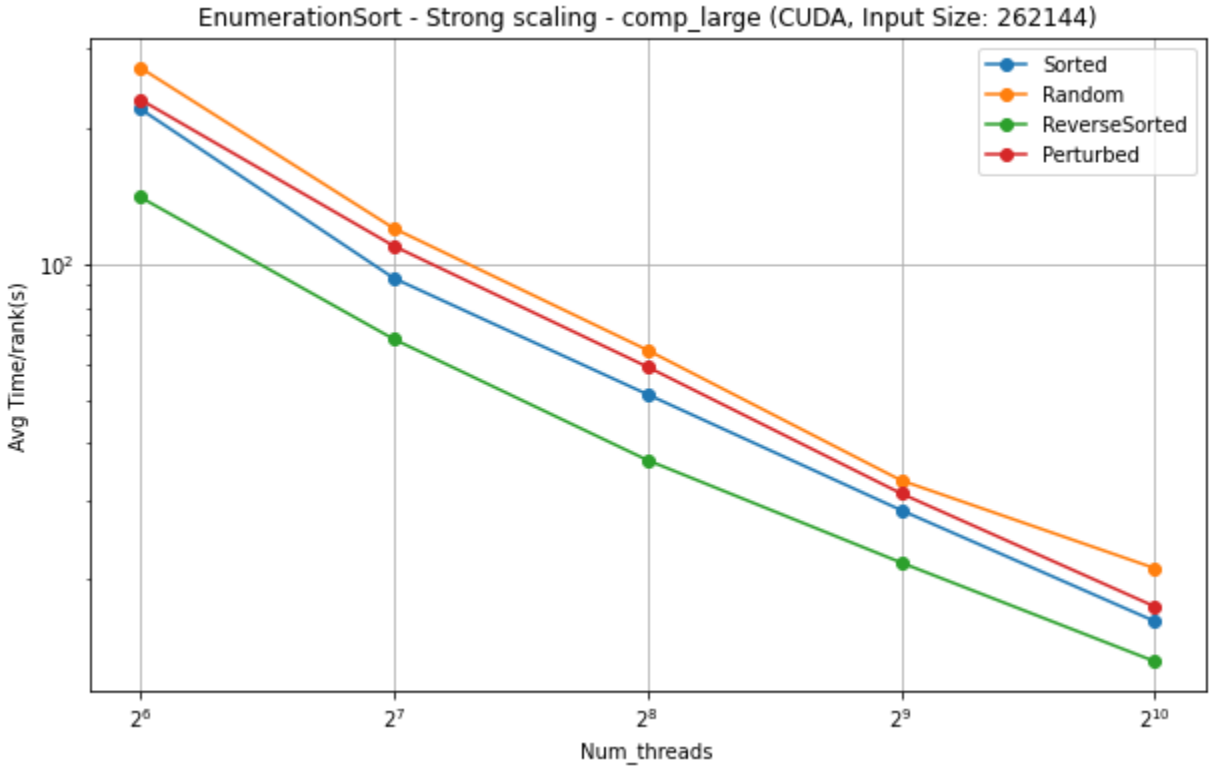

EnumerationSort - Strong scaling - comm (CUDA, Input Size: 65536)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
6×10^{-4}
5×10^{-4}
2^6
2^7
2^8
2^9
2^{10}
Num_threads

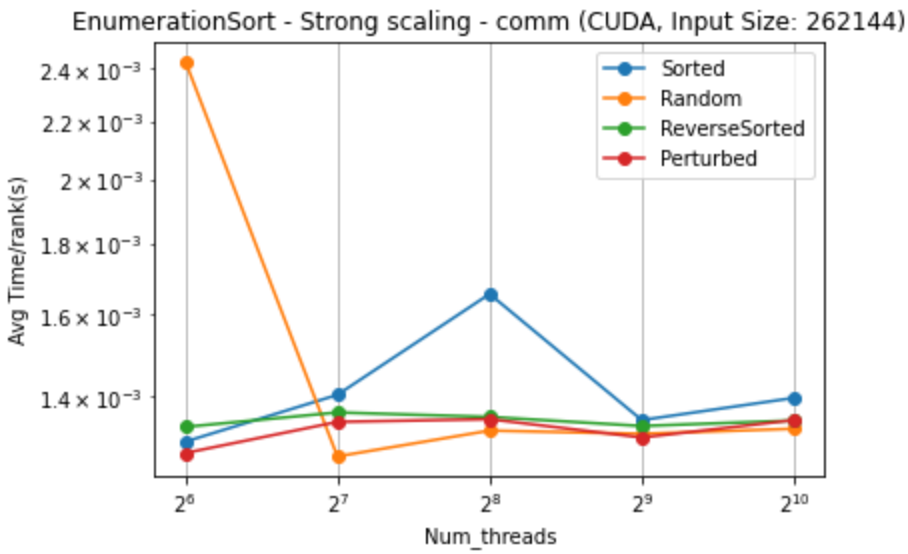

EnumerationSort - Strong scaling - main (CUDA, Input Size: 65536)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^0
2^6
2^7
2^8
2^9
2^{10}
Num_threads

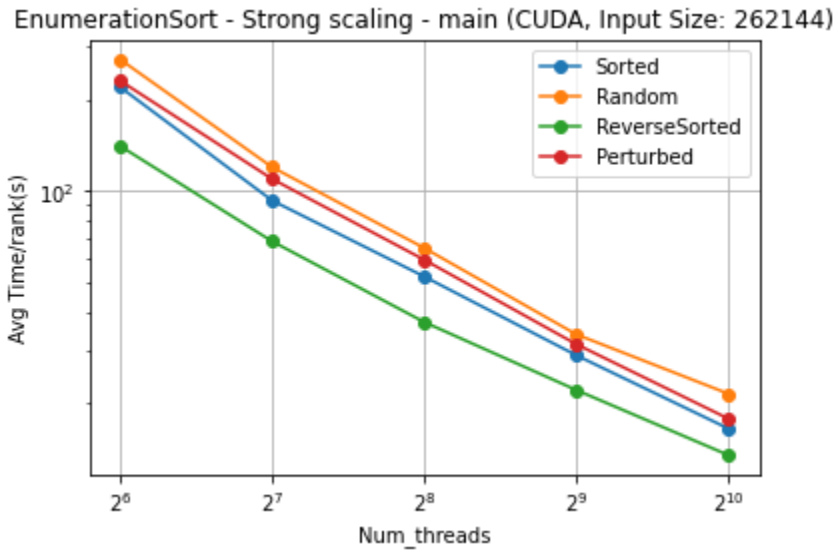

EnumerationSort - Strong scaling - comp_large (CUDA, Input Size: 131072)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
2^6
2^7
2^8
2^9
2^{10}
Num_threads

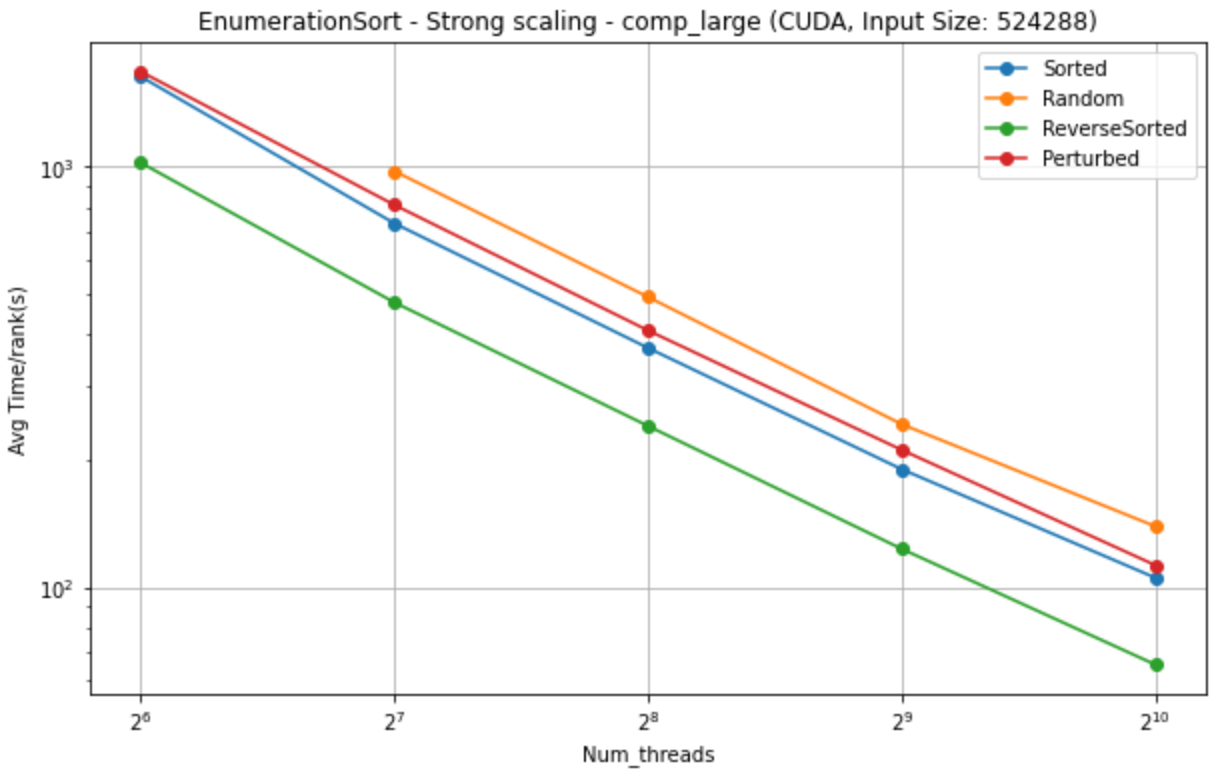

EnumerationSort - Strong scaling - comm (CUDA, Input Size: 131072)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
9×10^{-4}
8×10^{-4}
2^6
2^7
2^8
2^9
2^{10}
Num_threads

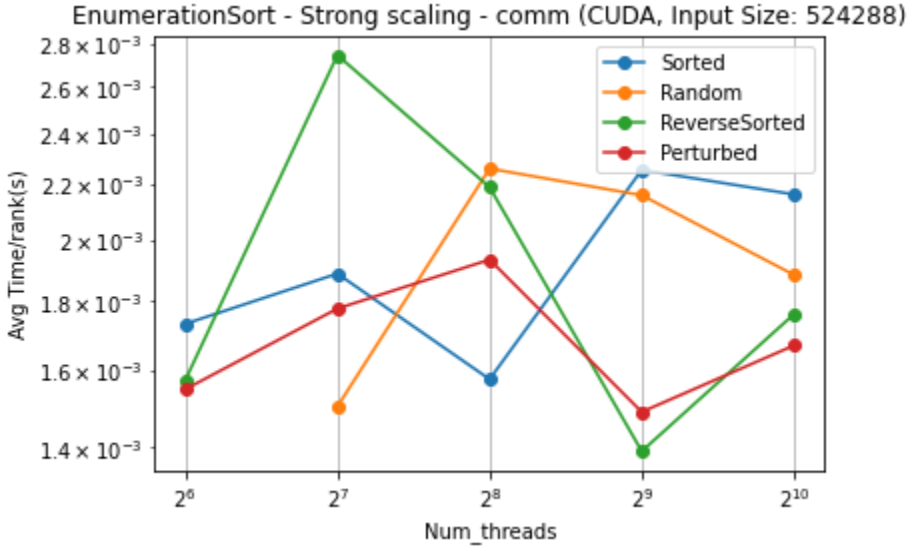

EnumerationSort - Strong scaling - main (CUDA, Input Size: 131072)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
2^6
2^7
2^8
2^9
2^{10}
Num_threads

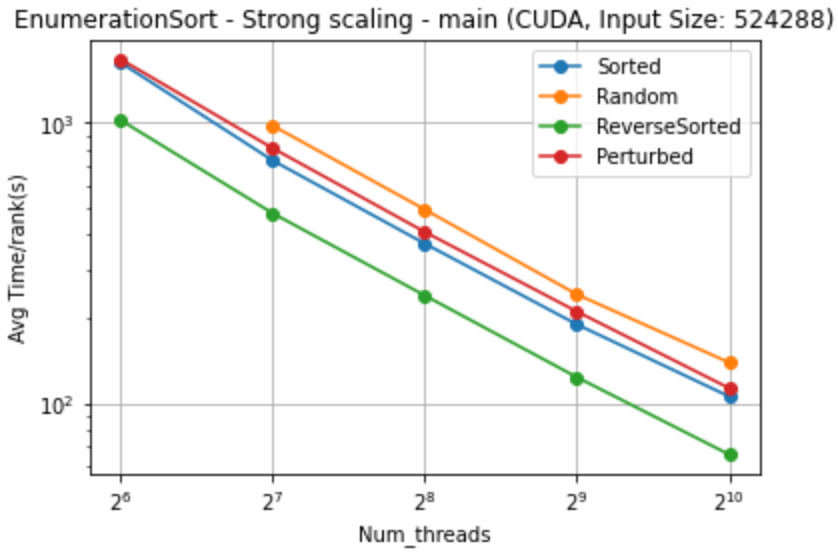

EnumerationSort - Strong scaling - comp_large (CUDA, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^2
2^6
2^7
2^8
2^9
2^{10}
Num_threads


EnumerationSort - Strong scaling - comm (CUDA, Input Size: 262144)
Avg Time/rank(s)
Num_threads
Sorted
Random
ReverseSorted
Perturbed

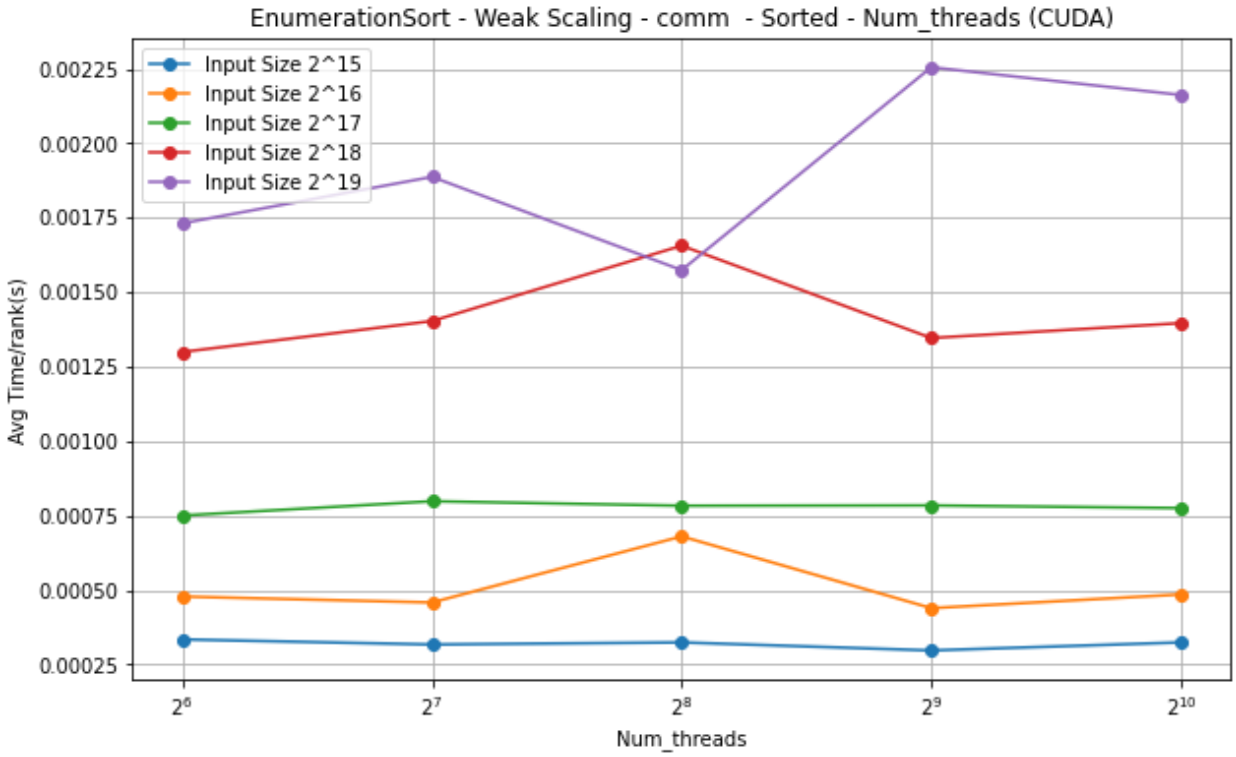

EnumerationSort - Strong scaling - main (CUDA, Input Size: 262144)
Avg Time/rank(s)
Num_threads
Sorted
Random
ReverseSorted
Perturbed

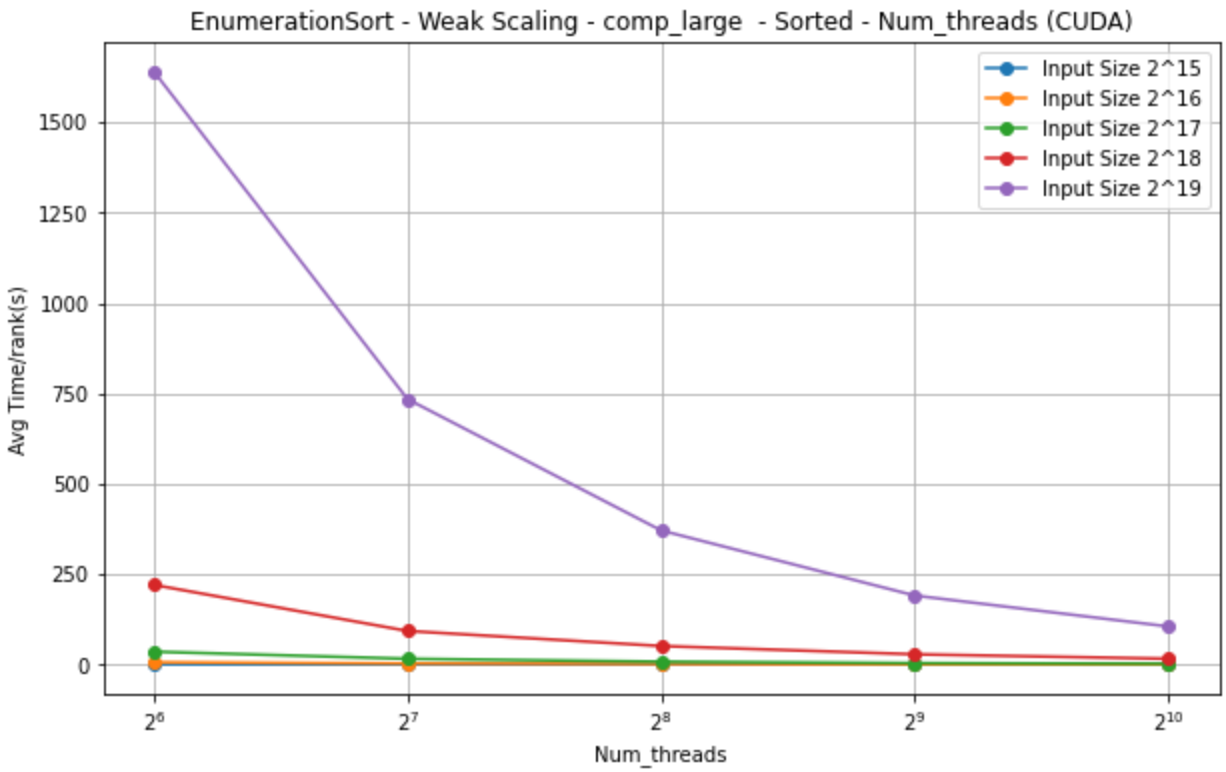

EnumerationSort - Strong scaling - comp_large (CUDA, Input Size: 524288)
Sorted
Random
ReverseSorted
Perturbed
10^3
10^2
Avg Time/rank(s)
2^6
2^7
2^8
2^9
2^{10}
Num_threads

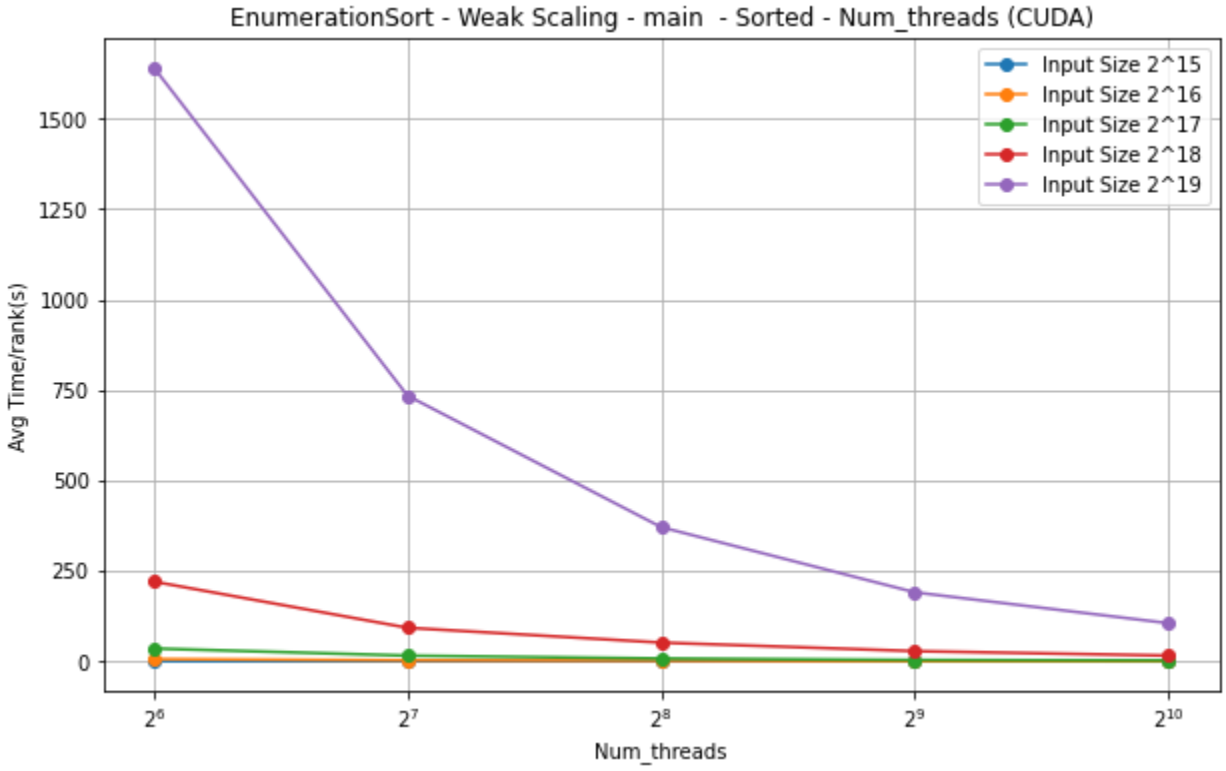

EnumerationSort - Strong scaling - comm (CUDA, Input Size: 524288)
Sorted
Random
ReverseSorted
Perturbed
2.8×10^{-3}
2.6×10^{-3}
2.4×10^{-3}
2.2×10^{-3}
2×10^{-3}
1.8×10^{-3}
1.6×10^{-3}
1.4×10^{-3}
Avg Time/rank(s)
2^6
2^7
2^8
2^9
2^{10}
Num_threads

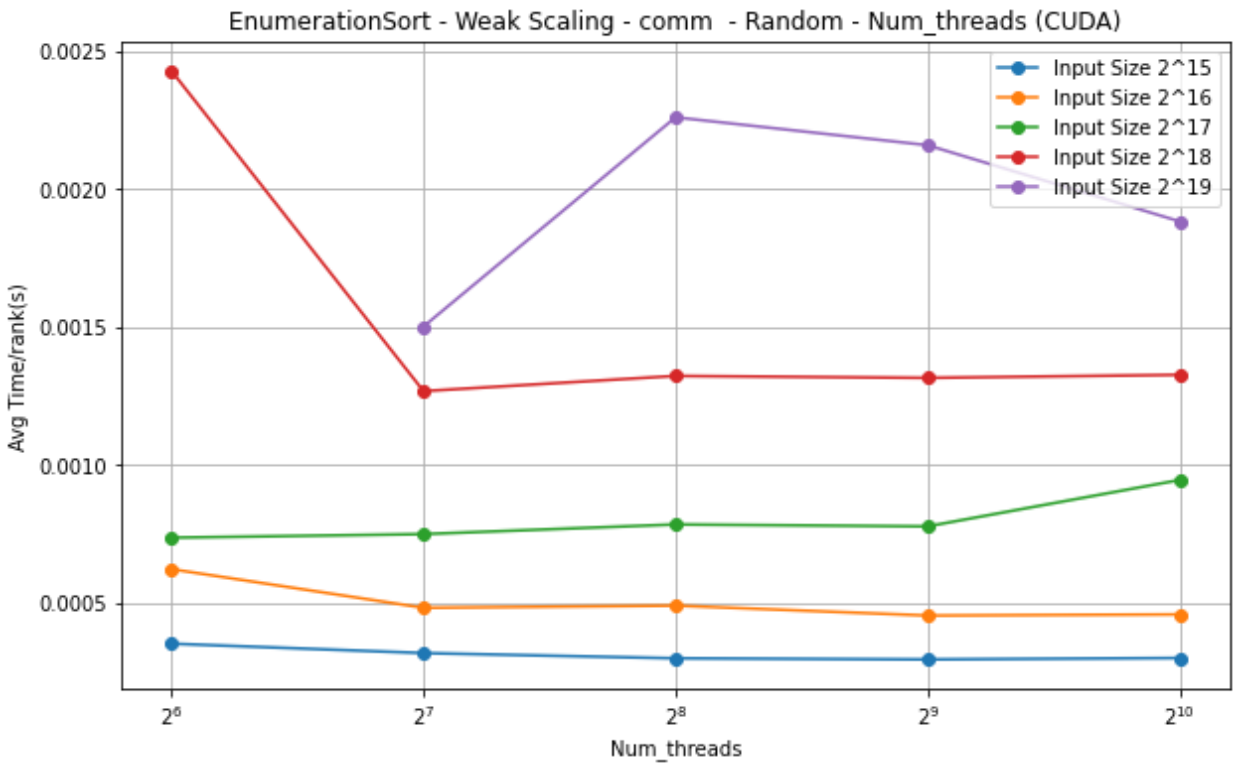


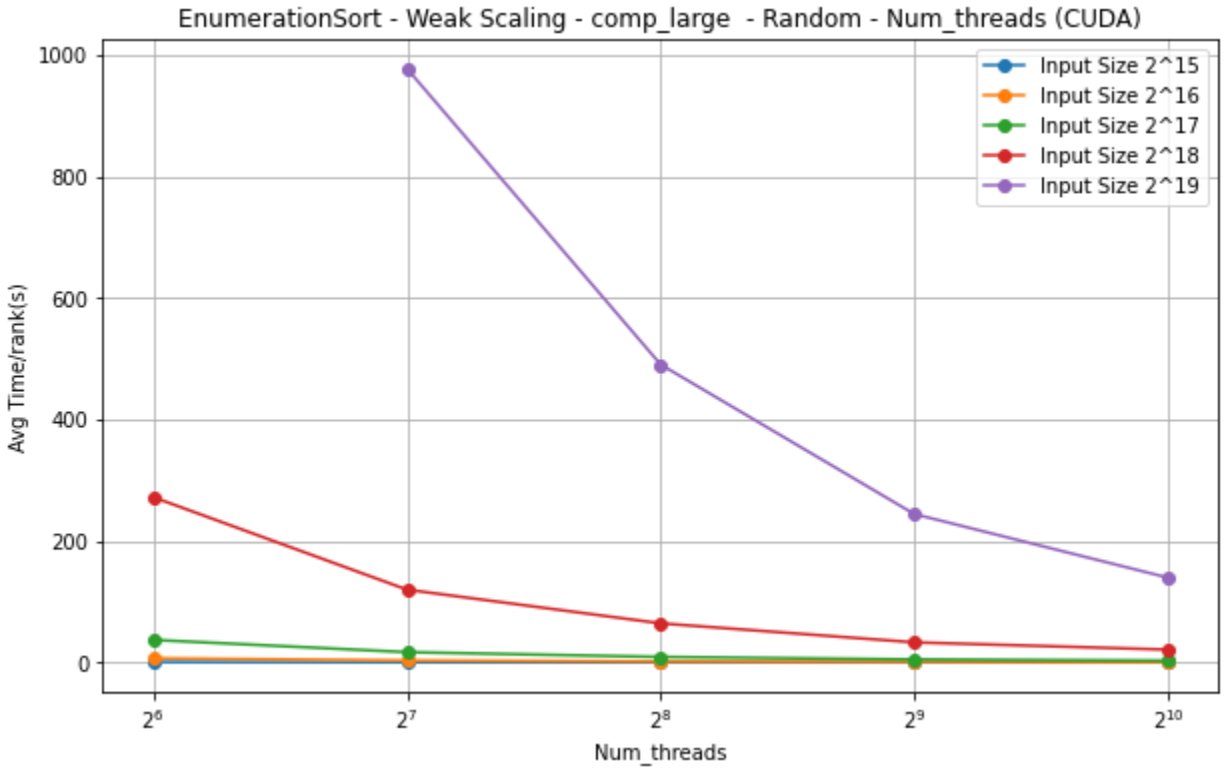
Weak Scaling:

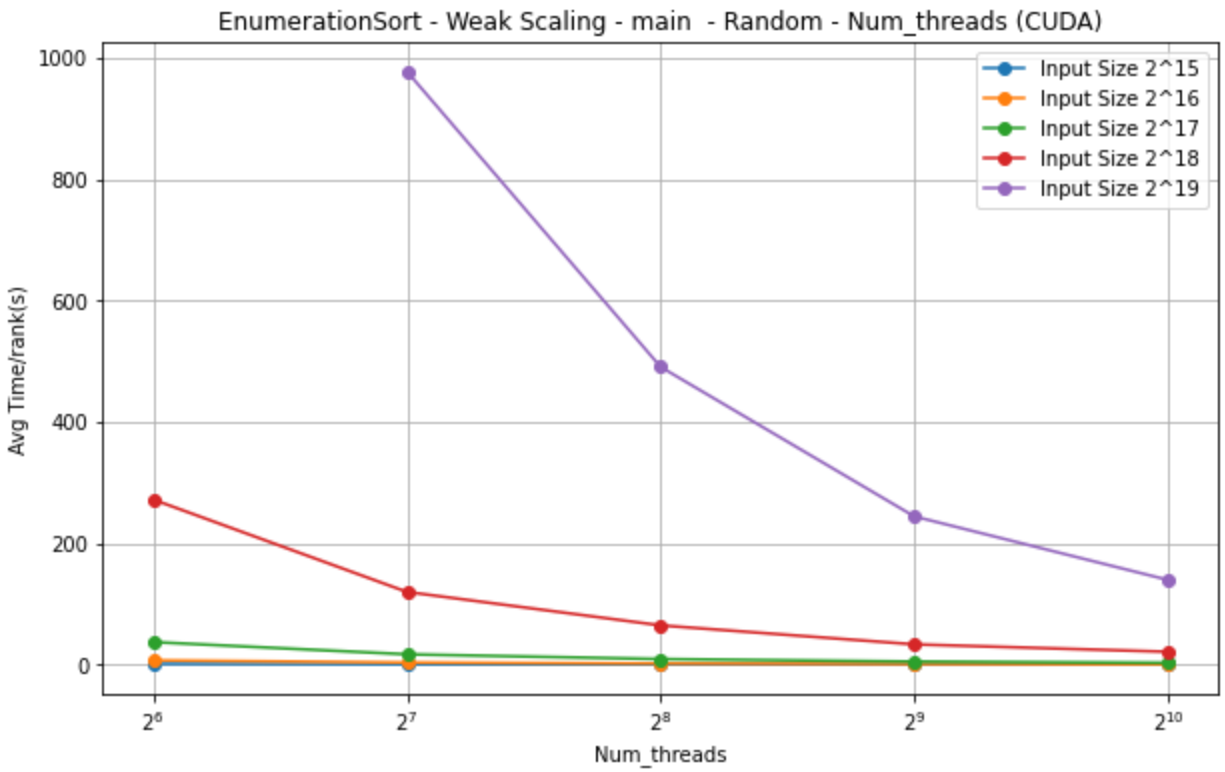


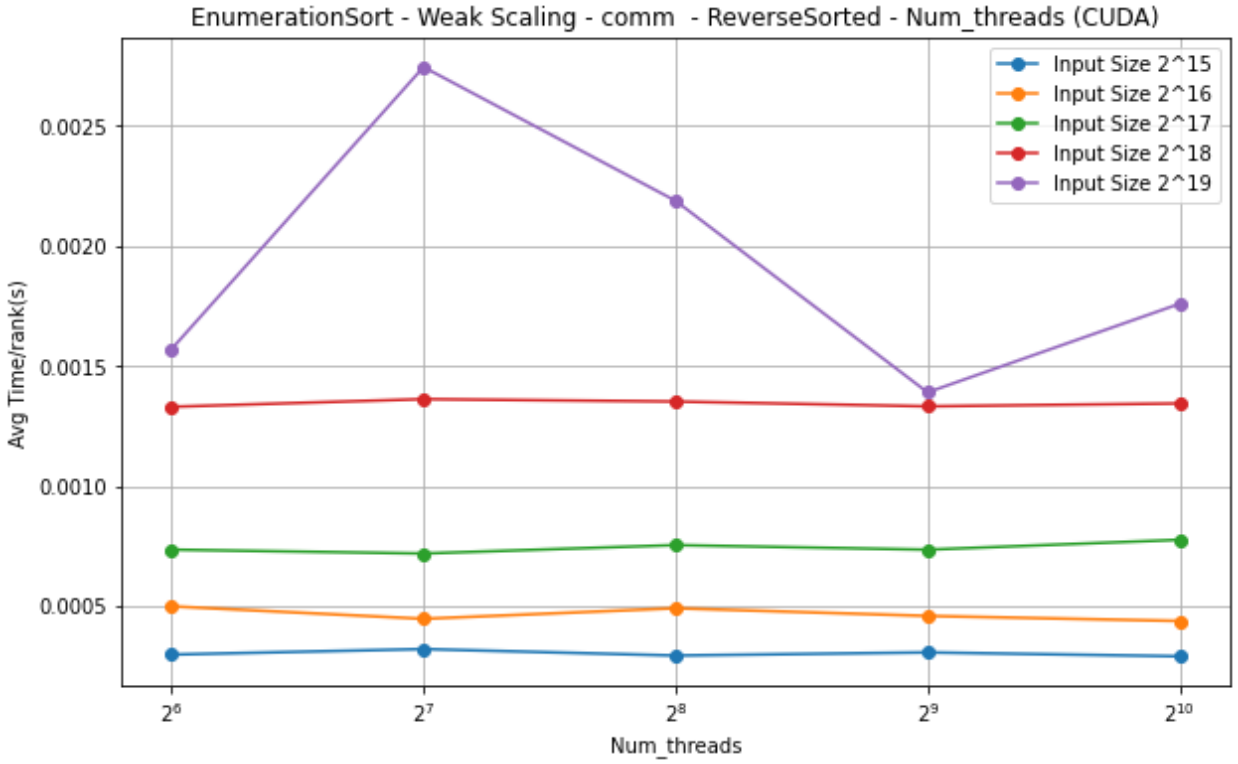

EnumerationSort - Weak Scaling - comp_large - Sorted - Num_threads (CUDA)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Avg Time/rank(s)
1500
1250
1000
750
500
250
0
2^6
2^7
2^8
2^9
2^{10}
Num_threads

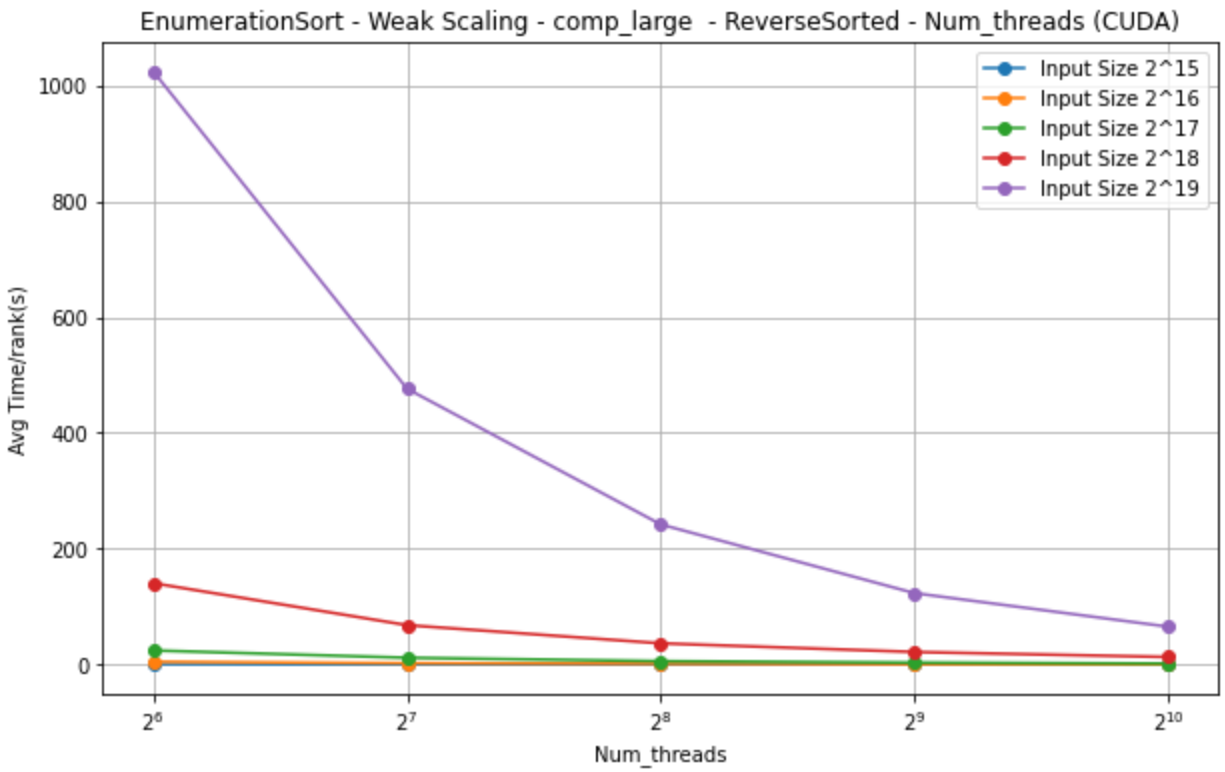

EnumerationSort - Weak Scaling - main - Sorted - Num_threads (CUDA)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Avg Time/rank(s)
1500
1250
1000
750
500
250
0
2^6
2^7
2^8
2^9
2^{10}
Num_threads

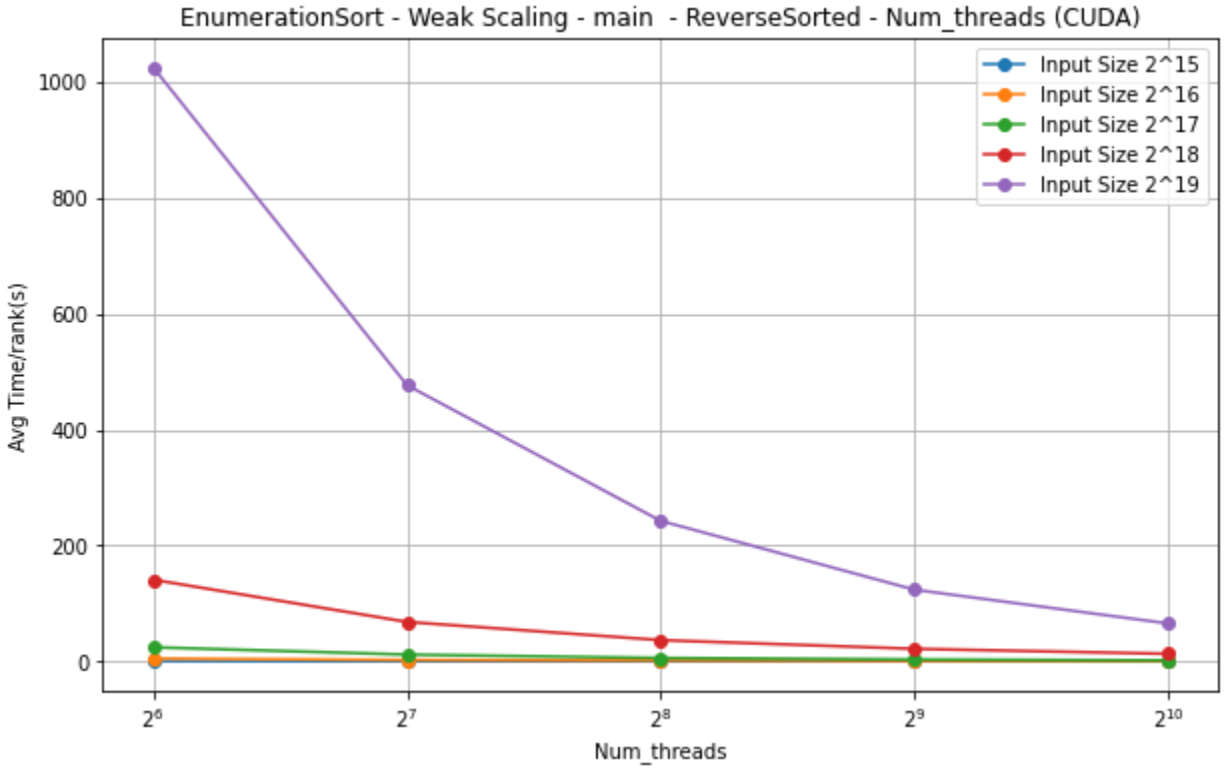

EnumerationSort - Weak Scaling - comm - Random - Num_threads (CUDA)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Avg Time/rank(s)
0.0025
0.0020
0.0015
0.0010
0.0005
2^6
2^7
2^8
2^9
2^{10}
Num_threads

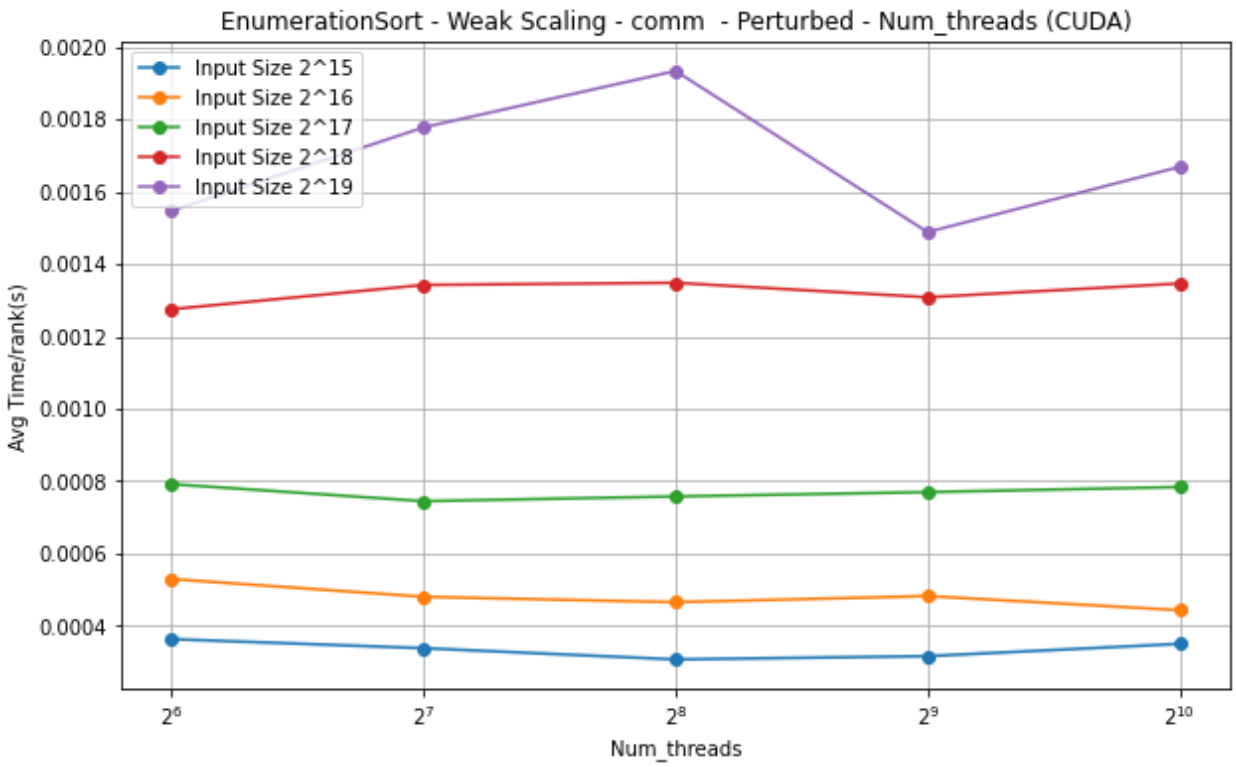

EnumerationSort - Weak Scaling - comp_large - Random - Num_threads (CUDA)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Avg Time/rank(s)
1000
800
600
400
200
0
2^6
2^7
2^8
2^9
2^{10}
Num_threads

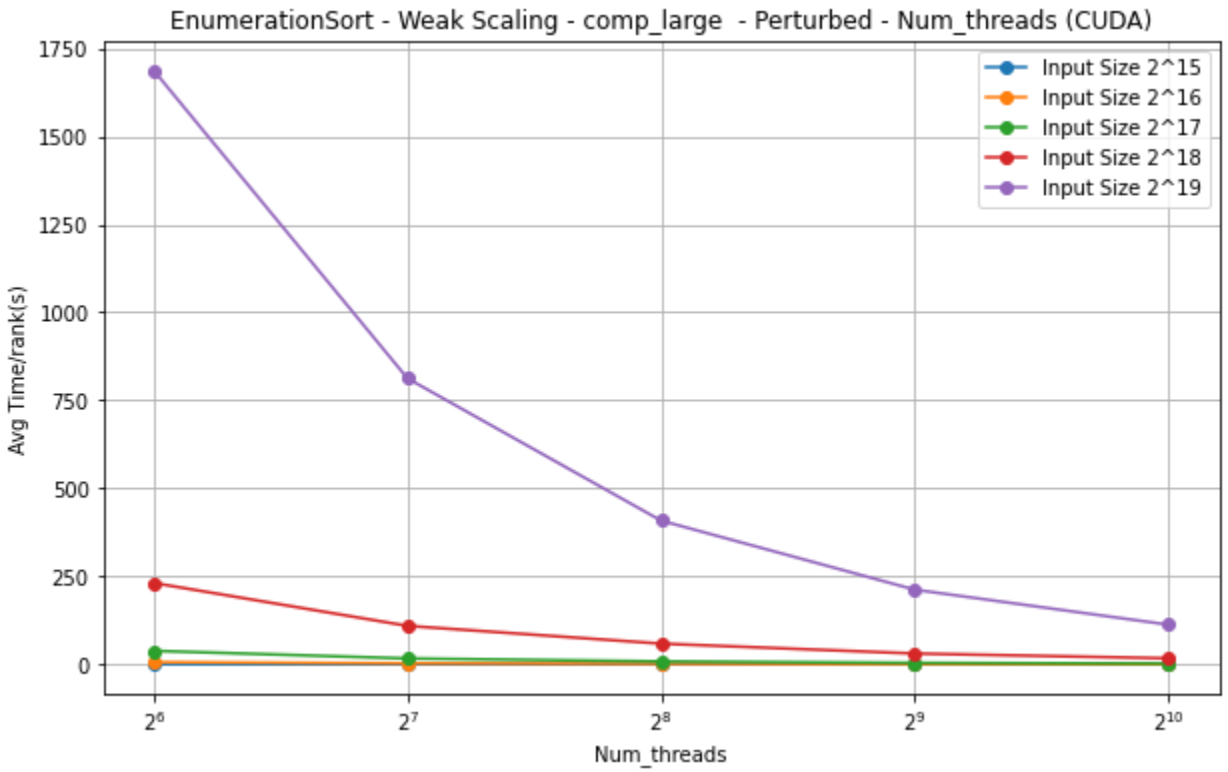

EnumerationSort - Weak Scaling - main - Random - Num_threads (CUDA)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Avg Time/rank(s)
1000
800
600
400
200
0
2^6
2^7
2^8
2^9
2^{10}
Num_threads

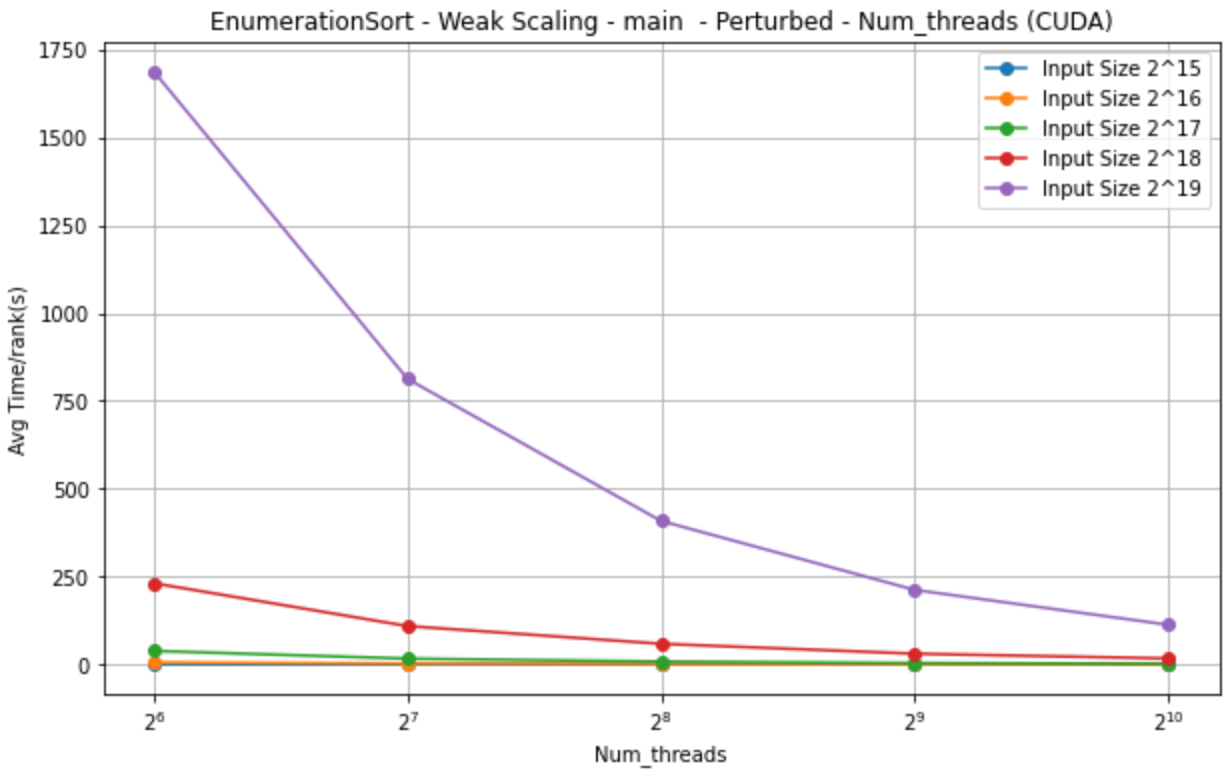

EnumerationSort - Weak Scaling - comm - ReverseSorted - Num_threads (CUDA)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Avg Time/rank(s)
0.0025
0.0020
0.0015
0.0010
0.0005
2^6
2^7
2^8
2^9
2^{10}
Num_threads


EnumerationSort - Weak Scaling - comp_large - ReverseSorted - Num_threads (CUDA)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Avg Time/rank(s)
0
200
400
600
800
1000
2^6
2^7
2^8
2^9
2^{10}
Num_threads

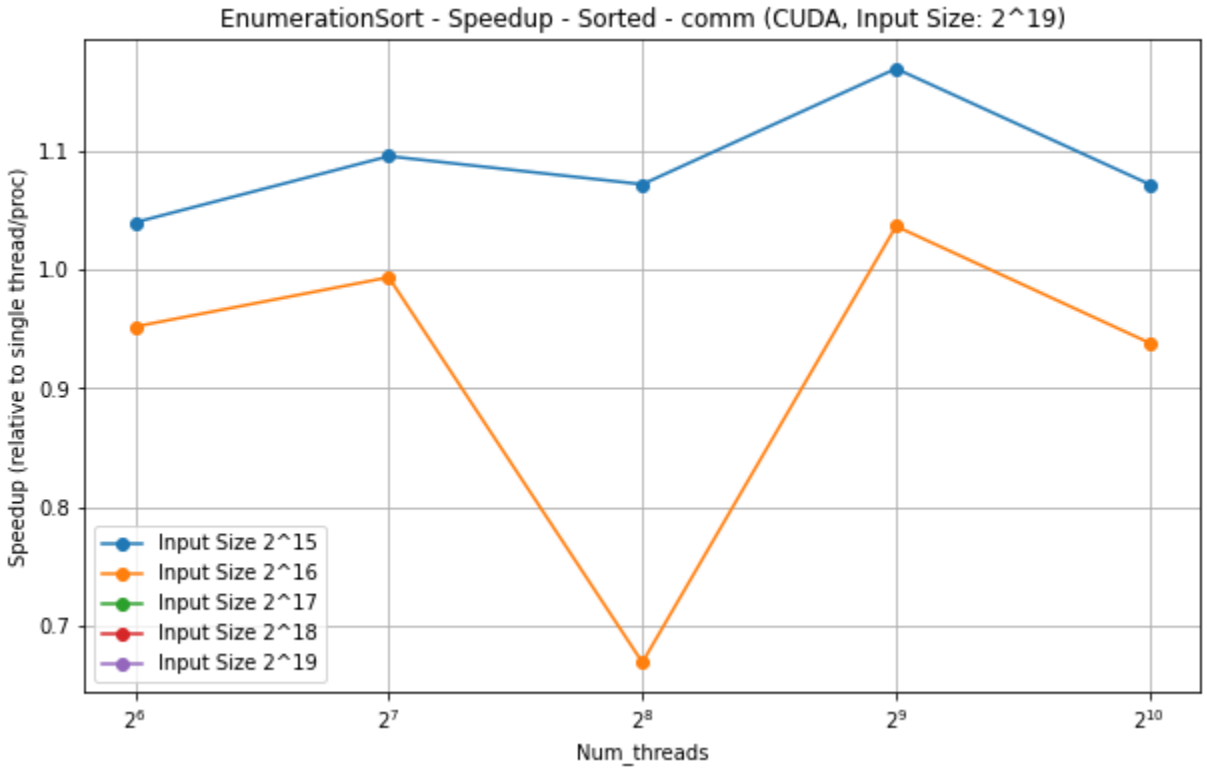

EnumerationSort - Weak Scaling - main - ReverseSorted - Num_threads (CUDA)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Avg Time/rank(s)
0
200
400
600
800
1000
2^6
2^7
2^8
2^9
2^{10}
Num_threads

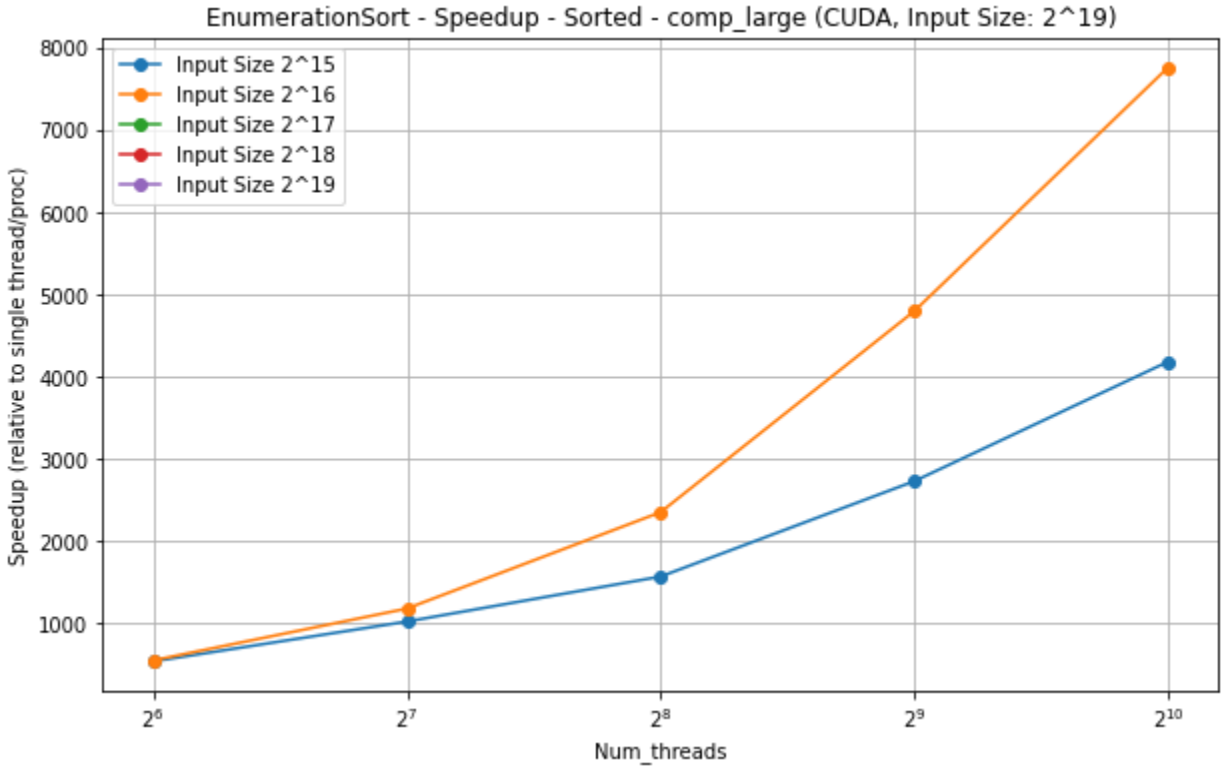

EnumerationSort - Weak Scaling - comm - Perturbed - Num_threads (CUDA)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Avg Time/rank(s)
0.0020
0.0018
0.0016
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
2^6
2^7
2^8
2^9
2^{10}
Num_threads

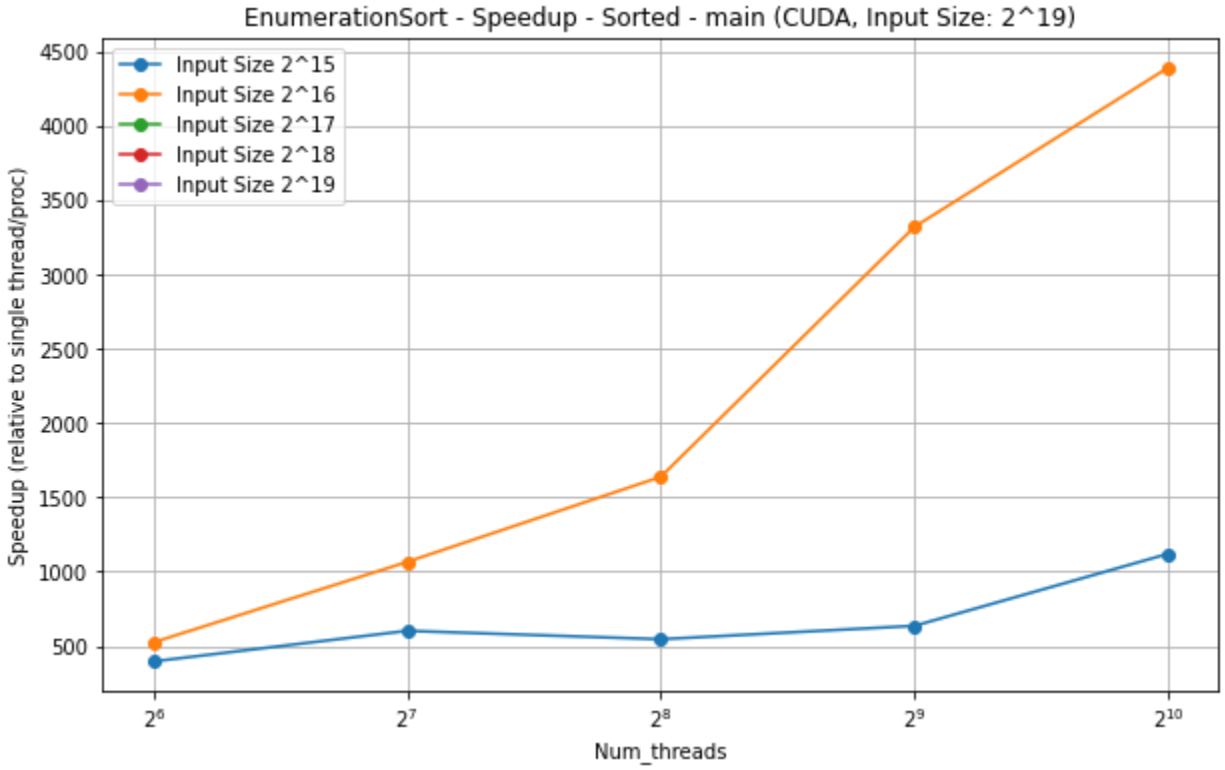

EnumerationSort - Weak Scaling - comp_large - Perturbed - Num_threads (CUDA)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Avg Time/rank(s)
1750
1500
1250
1000
750
500
250
0
2^6
2^7
2^8
2^9
2^{10}
Num_threads

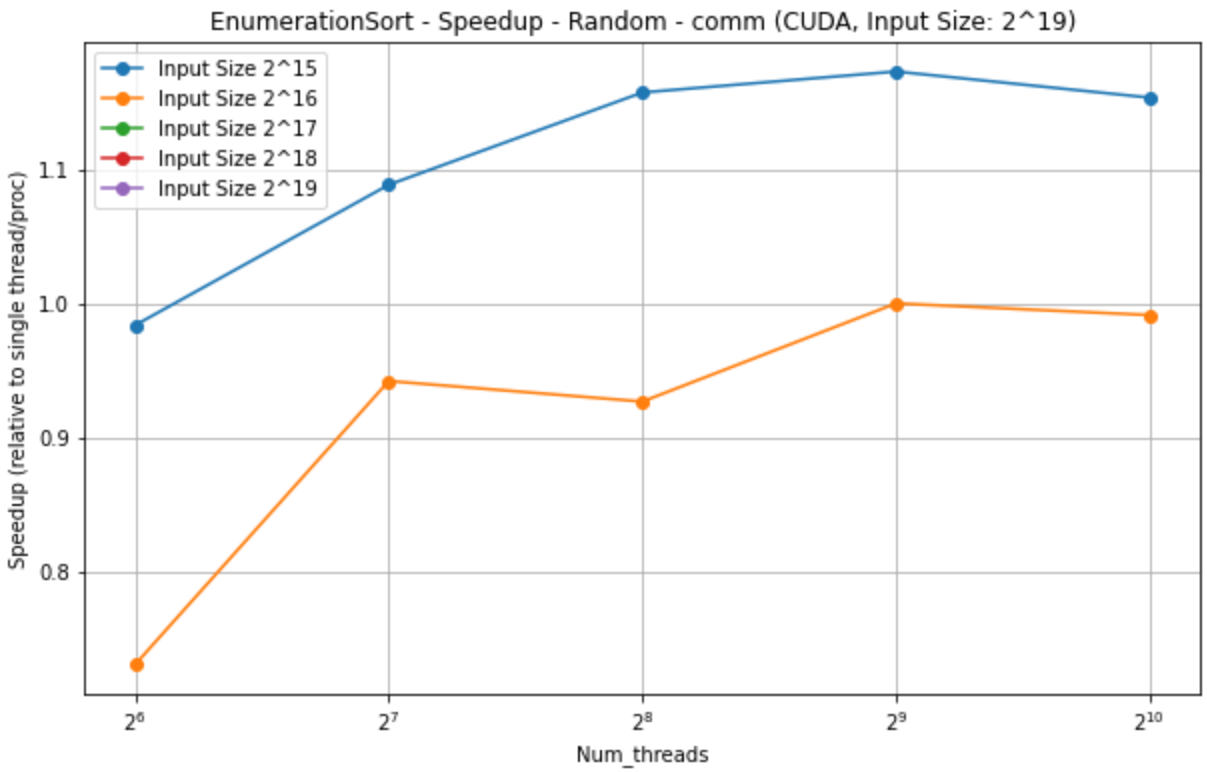

EnumerationSort - Weak Scaling - main - Perturbed - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
1750
1500
1250
1000
750
500
250
0
2^6
2^7
2^8
2^9
2^{10}

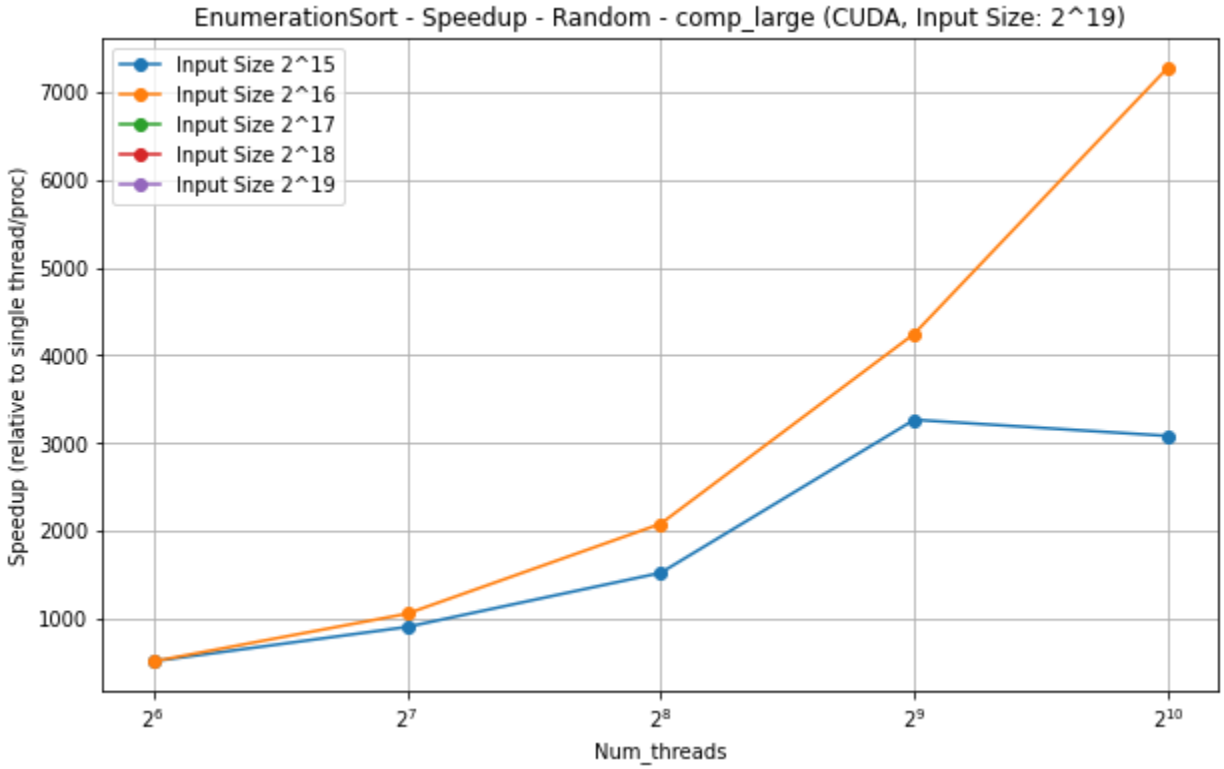
Speedup:

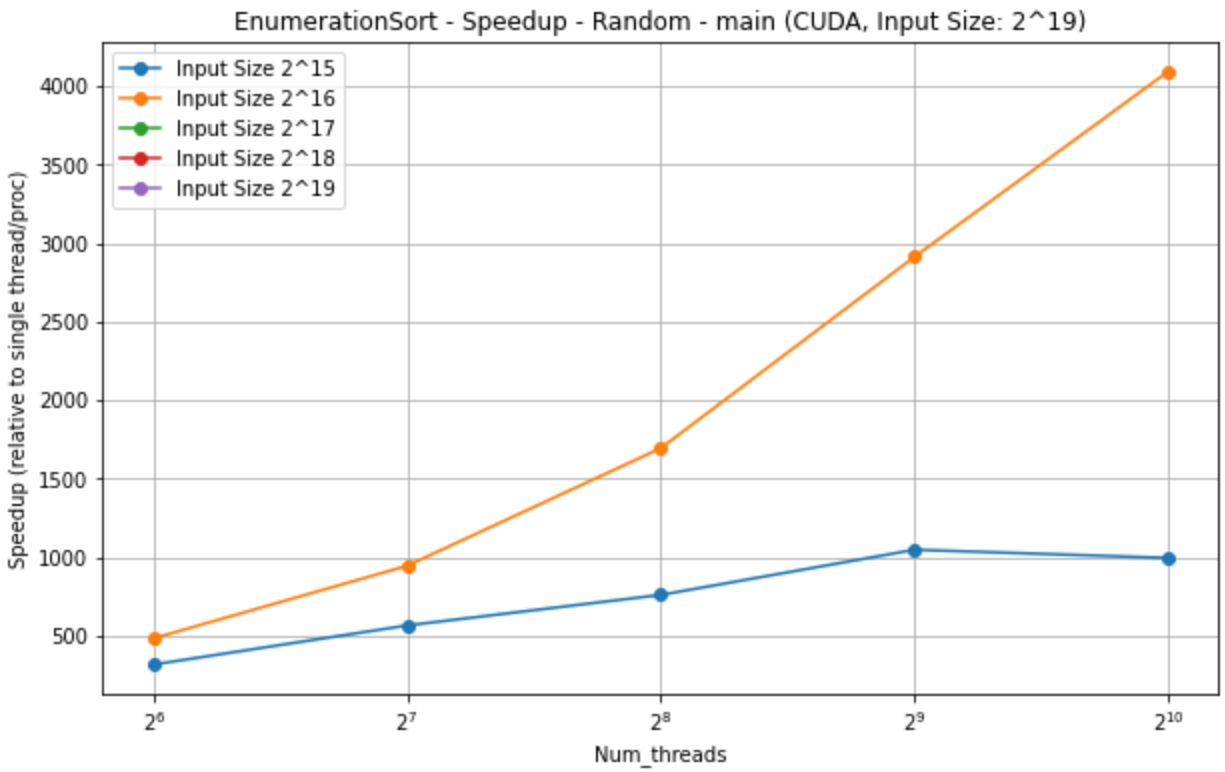

EnumerationSort - Speedup - Sorted - comm (CUDA, Input Size: 2^19)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
1.1
1.0
0.9
0.8
0.7
2^6
2^7
2^8
2^9
2^{10}

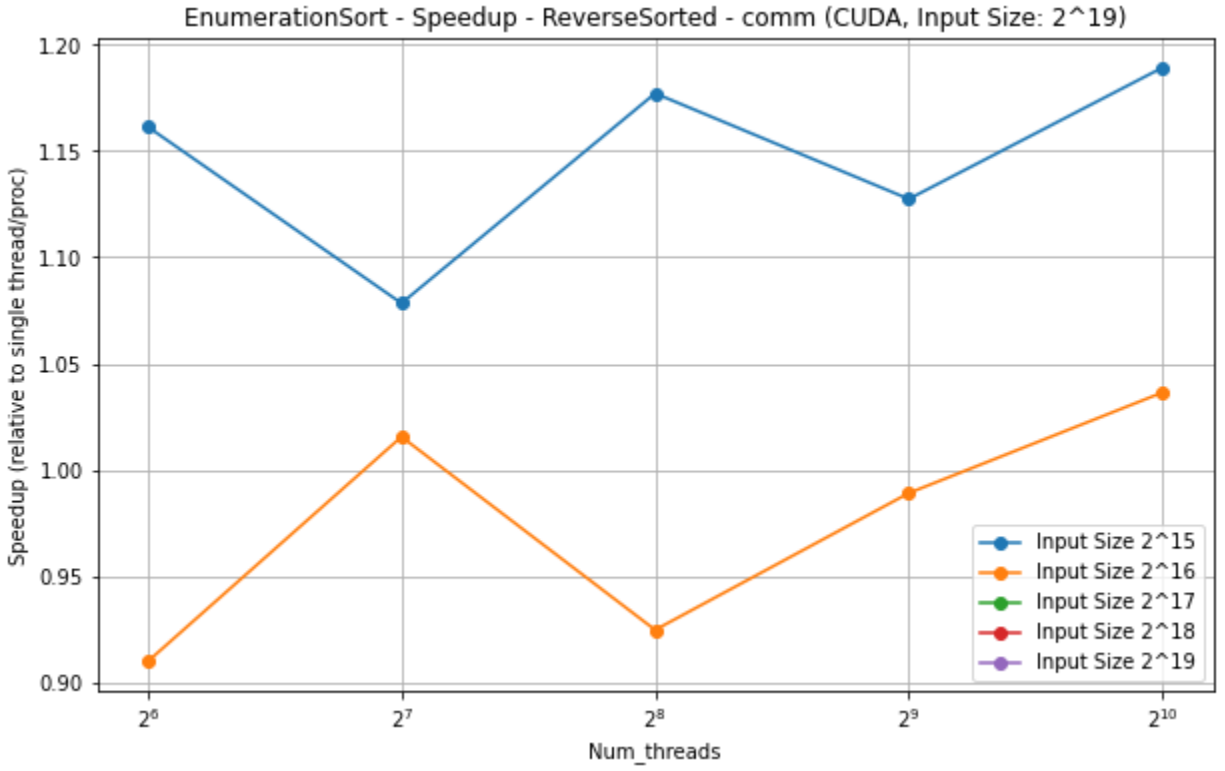

EnumerationSort - Speedup - Sorted - comp_large (CUDA, Input Size: 2^19)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Speedup (relative to single thread/proc)
8000
7000
6000
5000
4000
3000
2000
1000
2^6
2^7
2^8
2^9
2^{10}
Num_threads

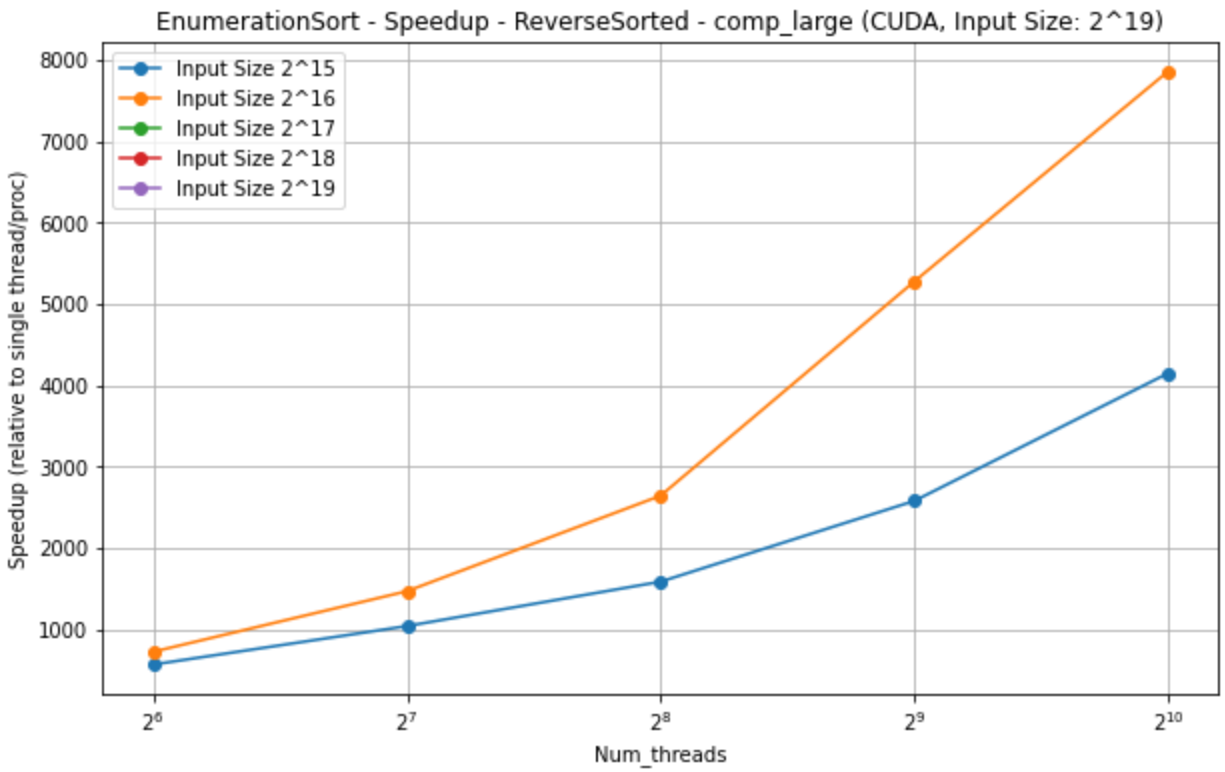

EnumerationSort - Speedup - Sorted - main (CUDA, Input Size: 2^19)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Speedup (relative to single thread/proc)
4500
4000
3500
3000
2500
2000
1500
1000
500
2^6
2^7
2^8
2^9
2^{10}
Num_threads

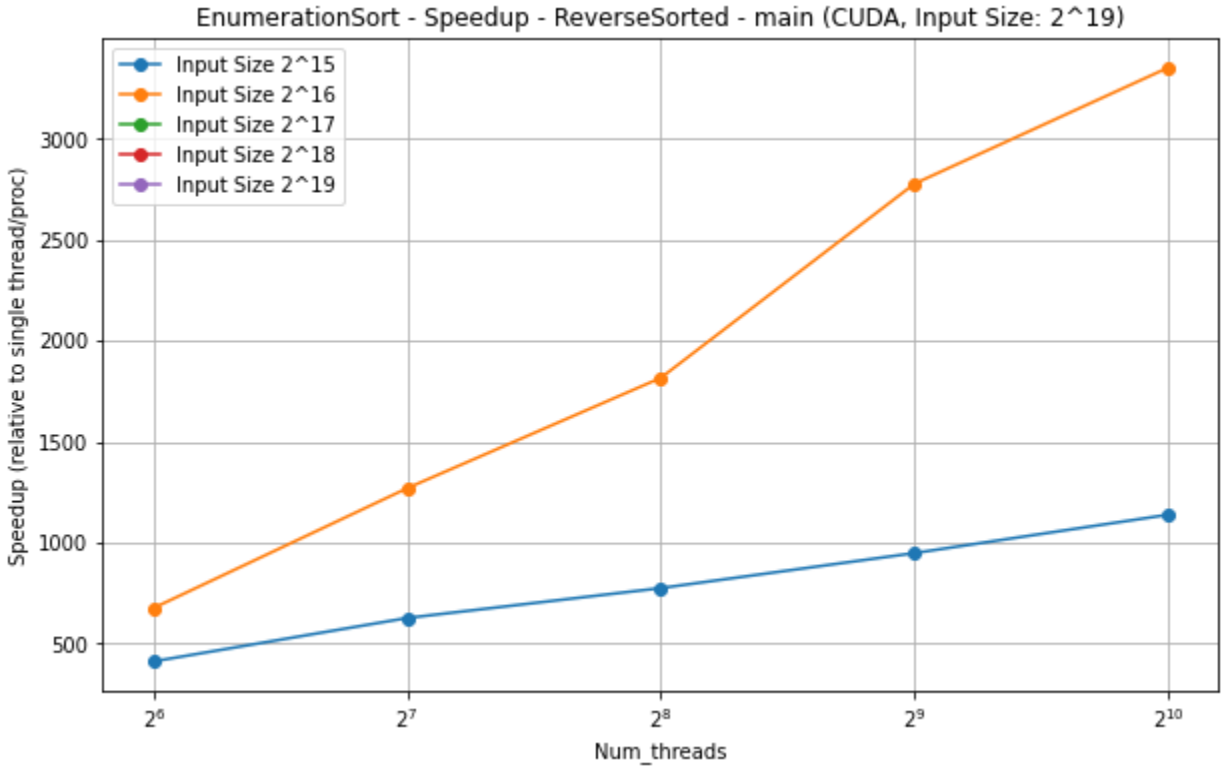

EnumerationSort - Speedup - Random - comm (CUDA, Input Size: 2^19)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Speedup (relative to single thread/proc)
1.1
1.0
0.9
0.8
2^6
2^7
2^8
2^9
2^{10}
Num_threads

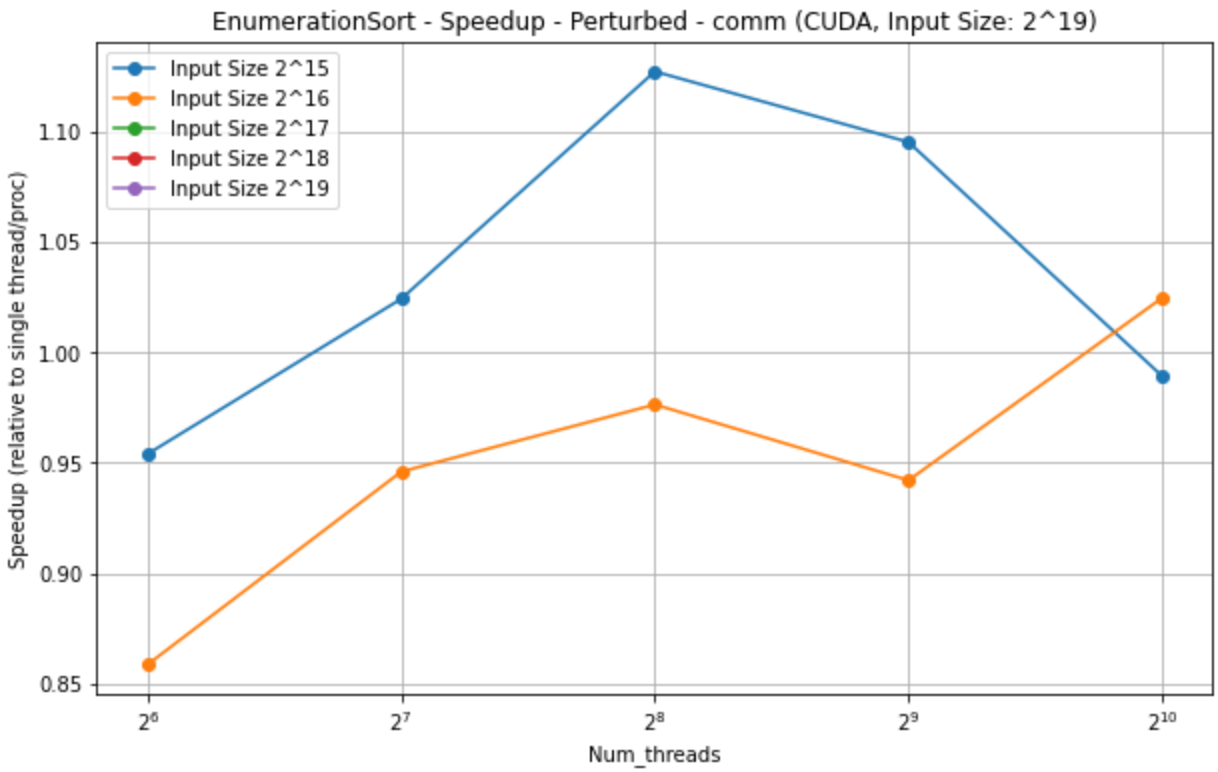

EnumerationSort - Speedup - Random - comp_large (CUDA, Input Size: 2^19)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Speedup (relative to single thread/proc)
7000
6000
5000
4000
3000
2000
1000
2^6
2^7
2^8
2^9
2^{10}
Num_threads

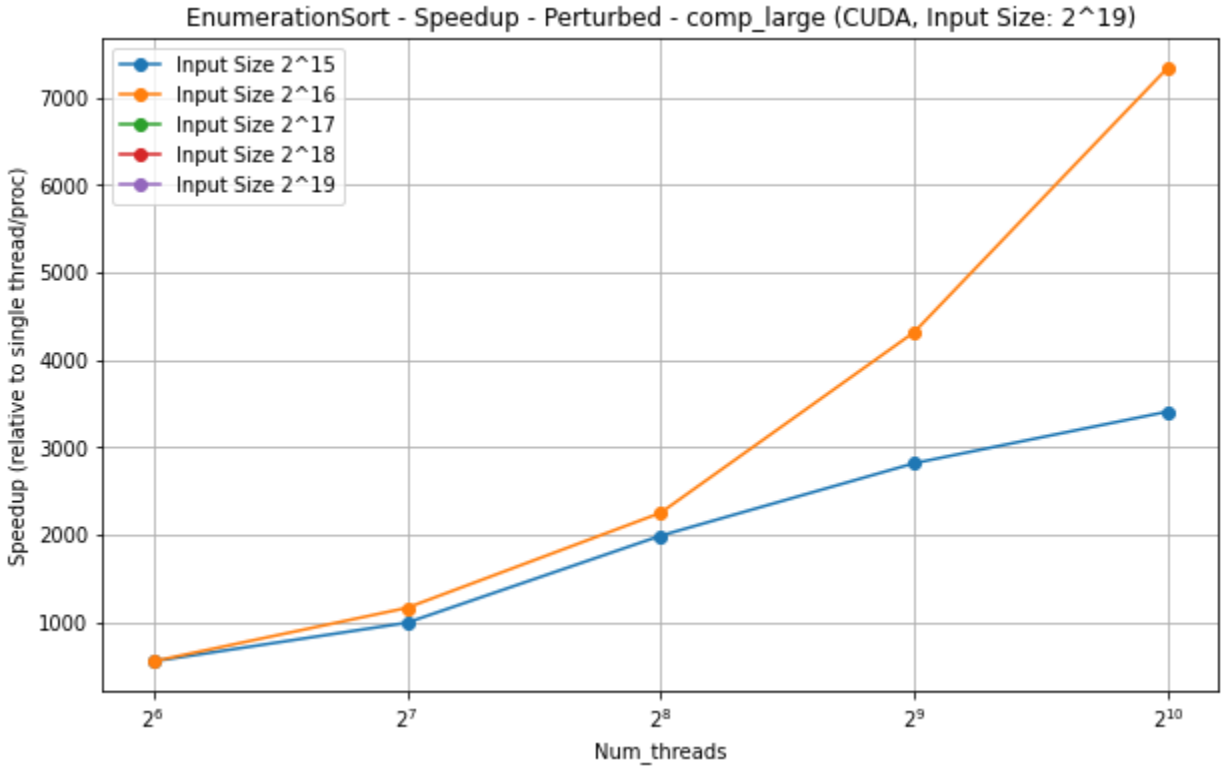

EnumerationSort - Speedup - Random - main (CUDA, Input Size: 2^19)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Speedup (relative to single thread/proc)
4000
3500
3000
2500
2000
1500
1000
500
2^6
2^7
2^8
2^9
2^{10}
Num_threads

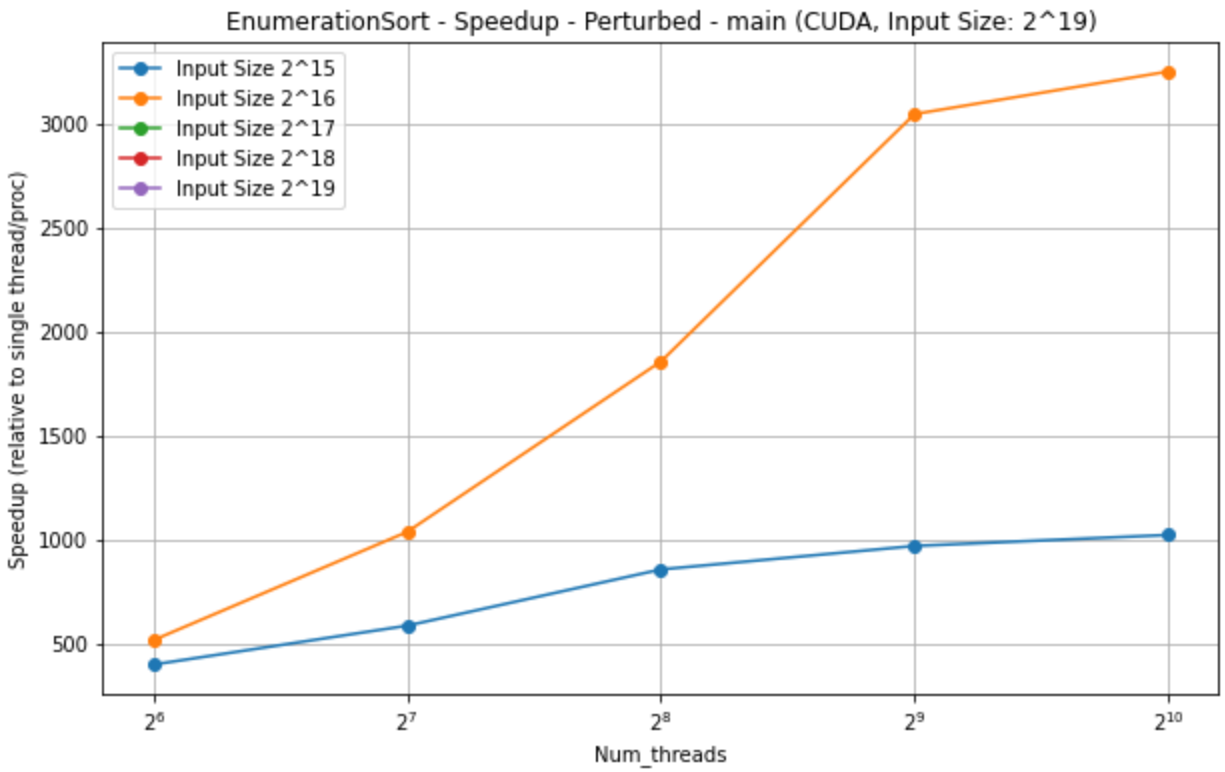

EnumerationSort - Speedup - ReverseSorted - comm (CUDA, Input Size: 2^19)
Speedup (relative to single thread/proc)
1.20
1.15
1.10
1.05
1.00
0.95
0.90
2^6
2^7
2^8
2^9
2^{10}
Num_threads
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19


EnumerationSort - Speedup - ReverseSorted - comp_large (CUDA, Input Size: 2^19)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Speedup (relative to single thread/proc)
8000
7000
6000
5000
4000
3000
2000
1000
2^6
2^7
2^8
2^9
2^{10}
Num_threads


EnumerationSort - Speedup - ReverseSorted - main (CUDA, Input Size: 2^19)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Speedup (relative to single thread/proc)
3000
2500
2000
1500
1000
500
2^6
2^7
2^8
2^9
2^{10}
Num_threads


EnumerationSort - Speedup - Perturbed - comm (CUDA, Input Size: 2^19)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Speedup (relative to single thread/proc)
1.10
1.05
1.00
0.95
0.90
0.85
2^6
2^7
2^8
2^9
2^{10}
Num_threads


EnumerationSort - Speedup - Perturbed - comm_large (CUDA, Input Size: 2^19)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Speedup (relative to single thread/proc)
7000
6000
5000
4000
3000
2000
1000
2^6
2^7
2^8
2^9
2^{10}
Num_threads


EnumerationSort - Speedup - Perturbed - main (CUDA, Input Size: 2^19)
Input Size 2^15
Input Size 2^16
Input Size 2^17
Input Size 2^18
Input Size 2^19
Speedup (relative to single thread/proc)
3000
2500
2000
1500
1000
500
2^6
2^7
2^8
2^9
2^{10}
Num_threads

Odd-Even Transposition Sort:

Odd-Even transposition sort is an algorithm that is based on the parallel bubble sorting algorithm in which blocks of the array are broken up and sorted in parallel in even and odd kernels. The kernels are then combined and sorted and more kernels are created and sorted. The time complexity of this algorithm is O(n^2). This algorithm's strength lies in sorting at higher input sizes rather than smaller input sizes. The MPI cali files were able to be generated for input types sorted, random, reverse sorted, and perturbed. The MPI files were all able to be generated with the exception of 2 cali files being timed out. The CUDA files were not all able to be generated due to the cali files being timed out before the processes could finish. This is because the time limit was set to 30 minutes before timing out. The files that could be generated measured a different number of processes for different amounts of array values for the input types of sorted, random, reverse sorted and perturbed.

Considerations:

Odd-Even Transposition sort is an algorithm that has different strengths and weaknesses depending on the implementation being CUDA or MPI. For the CUDA implementation as the number of values increases, the algorithm is able to better take advantage of parallelism. For the MPI implementation, the algorithm is able to sort faster on a smaller number of processes.

Not all the CUDA cali files were able to be generated due to a timeout error. The MPI files were able to be mostly generated. As a result of some of the cali files not, some trends in the plots may be skewed by outliers in the data, however, the overall shape and correlations of the plots will still be able to be seen.

Performance Analysis:

The strengths of Odd-Even Transposition sort is that it is able to take advantage of parallelism to sort arrays of large input types.

For the MPI implementation of odd-even transposition sort as the number of processes increases, the average time will increase with it across all input types. This trend is consistent with strong scaling, weak scaling, and speedup. An explanation for why this is the case is that as the number of processes increases, there will be an increased communication overhead between the processes that will begin to dominate and increase the amount of time taken. Strong scaling also has limitations that after a certain size, there will be a saturation point that will not help performance. The speedup will decrease as the number of processes decreases.

For the CUDA implementation of odd-even transposition sort, as the number of threads increases, the average time taken to sort the algorithm will decrease across all input types. This trend is consistent with strong scaling, weak scaling. The CUDA implementation is able to

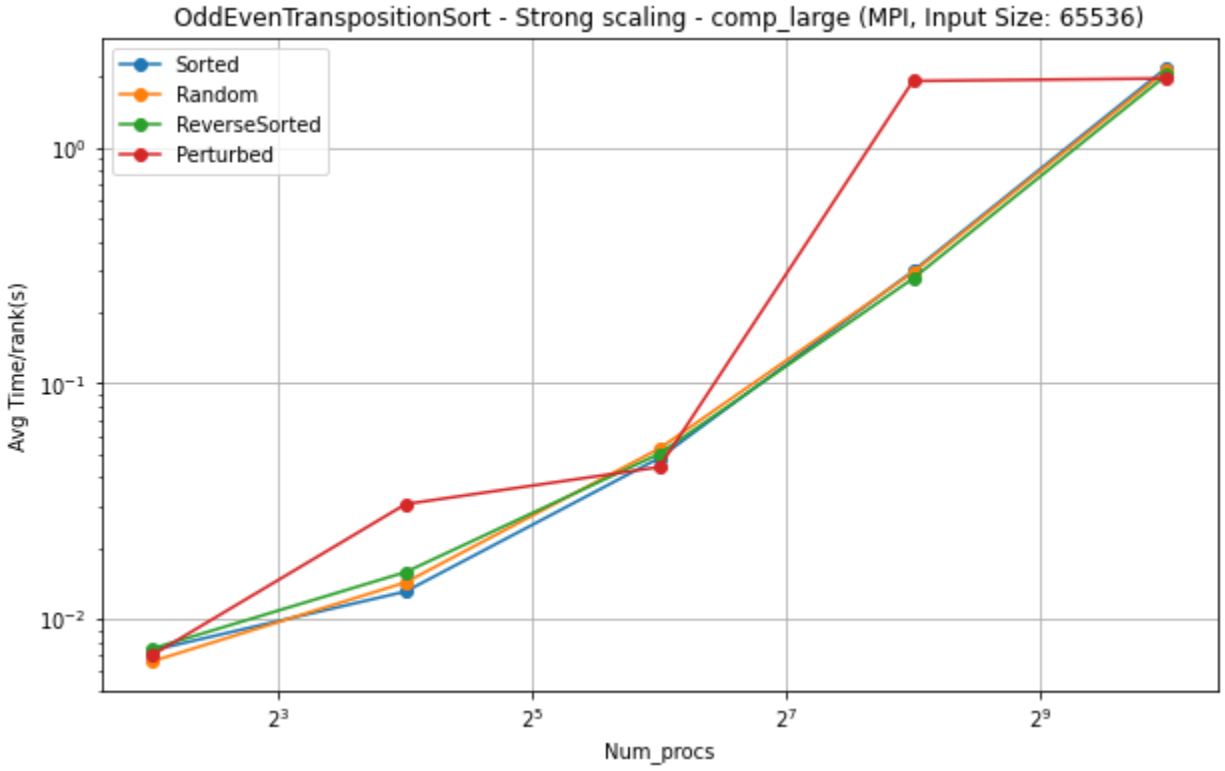
benefit largely from parallelism on large input sizes. The speedup plots for CUDA did not scale well as a result of CUDA core limitations, kernel launch overhead, and communication costs overhead affecting the speedup.

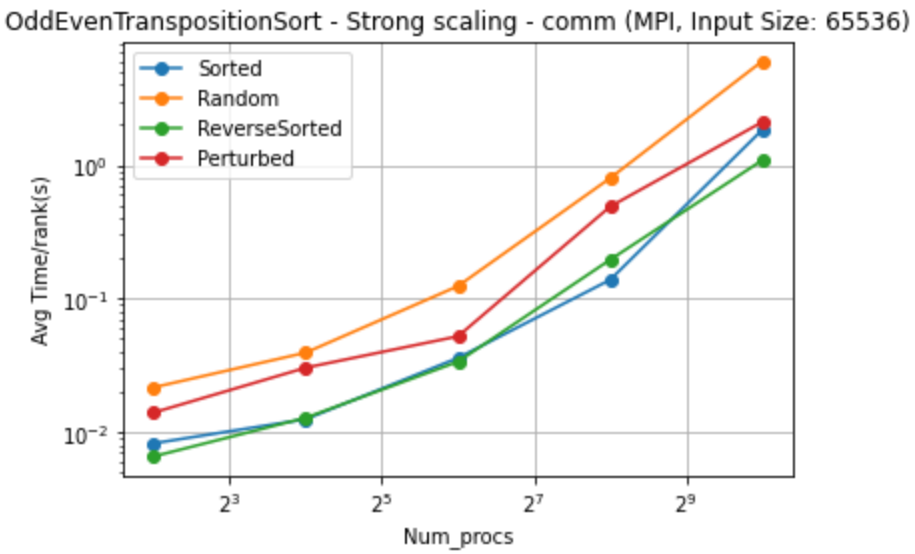
Speedup analysis:
The general trend from the CUDA speedup plots is that the higher the input size the higher the speedup. For MPI, the input size will not affect the speedup significantly, however as the number of processes increases, the speedup will decrease exponentially. As the number of processes increases in MPI, it becomes less efficient to use this algorithm. The CUDA algorithm is much more efficient than its sequential counterpart and is able to effectively take advantage of GPU parallelism and resource utilization.

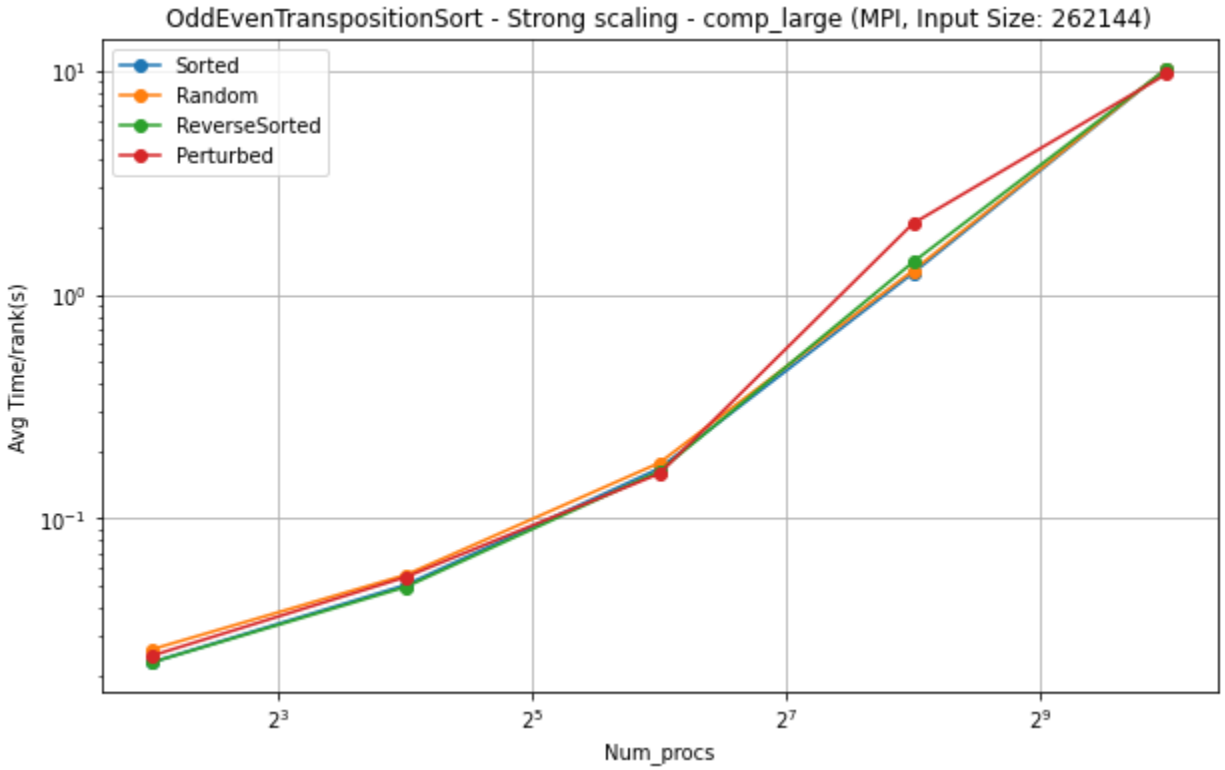
Odd-Even Transposition Sort Plots:

MPI:

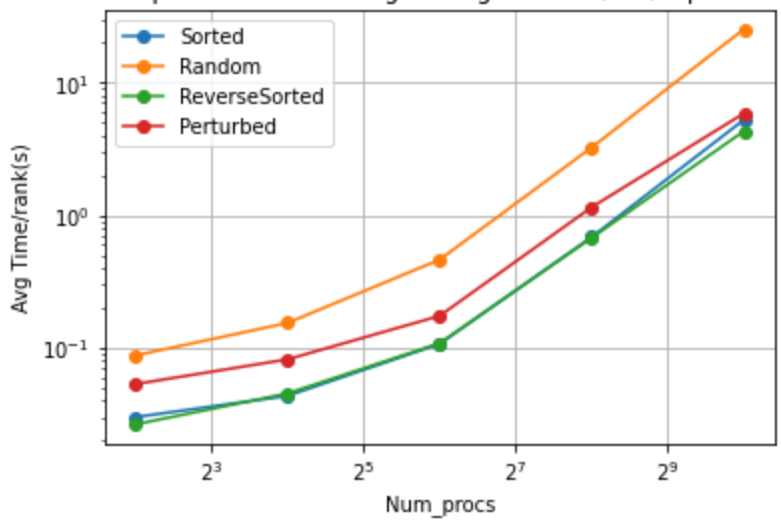
Strong Scaling:



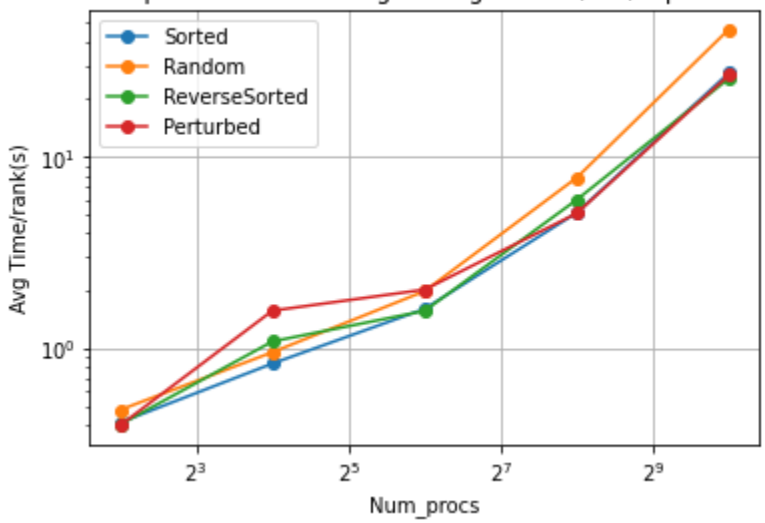

OddEvenTranspositionSort - Strong scaling - comm (MPI, Input Size: 65536)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{0}
10^{-1}
10^{-2}
2^{3}
2^{5}
2^{7}
2^{9}
Num_procs

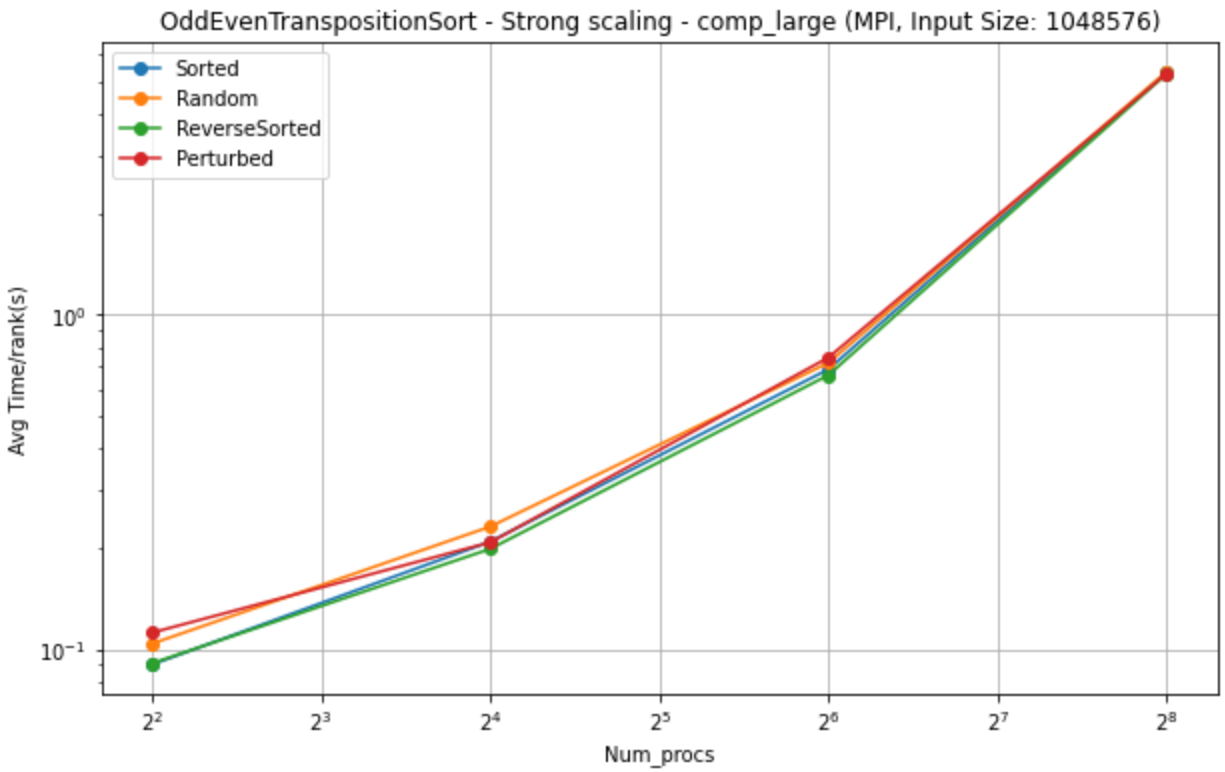

OddEvenTranspositionSort - Strong scaling - comp_large (MPI, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{1}
10^{0}
10^{-1}
2^{3}
2^{5}
2^{7}
2^{9}
Num_procs

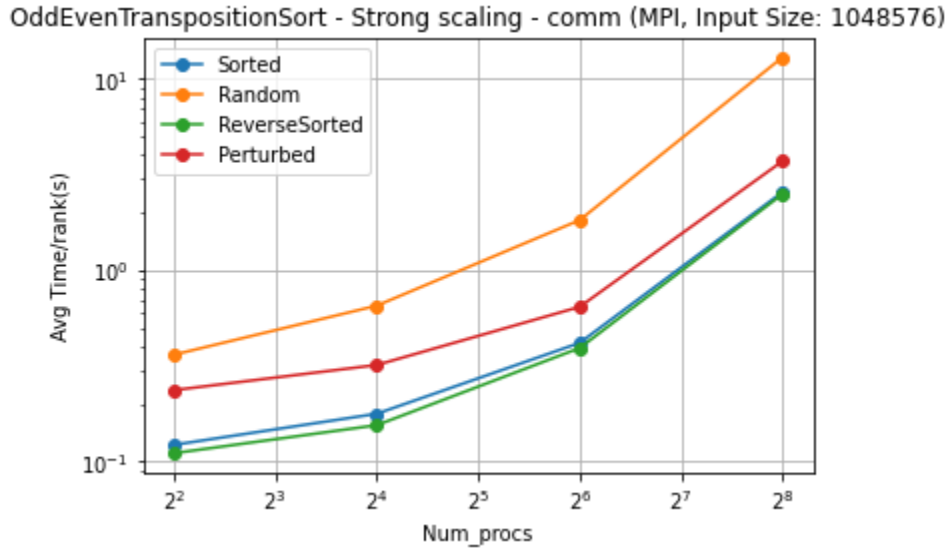
OddEvenTranspositionSort - Strong scaling - comm (MPI, Input Size: 262144)

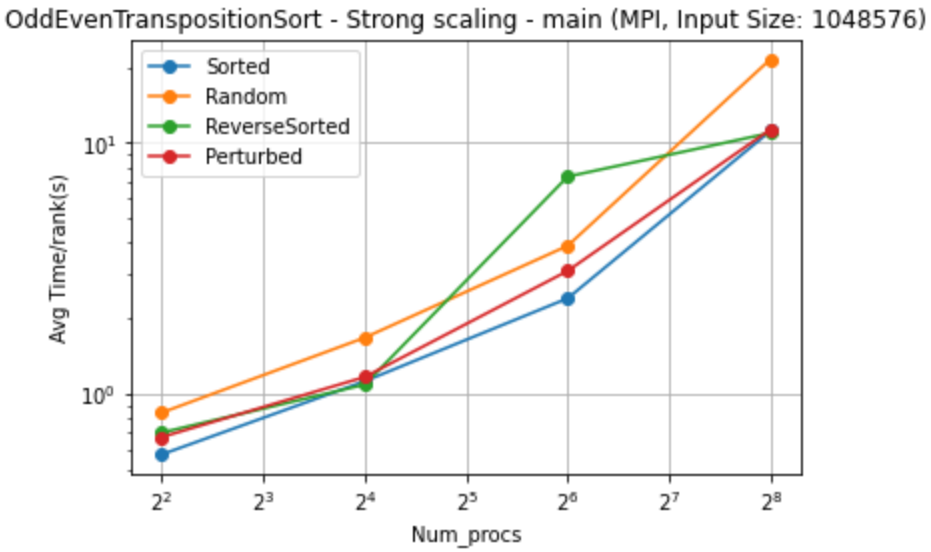


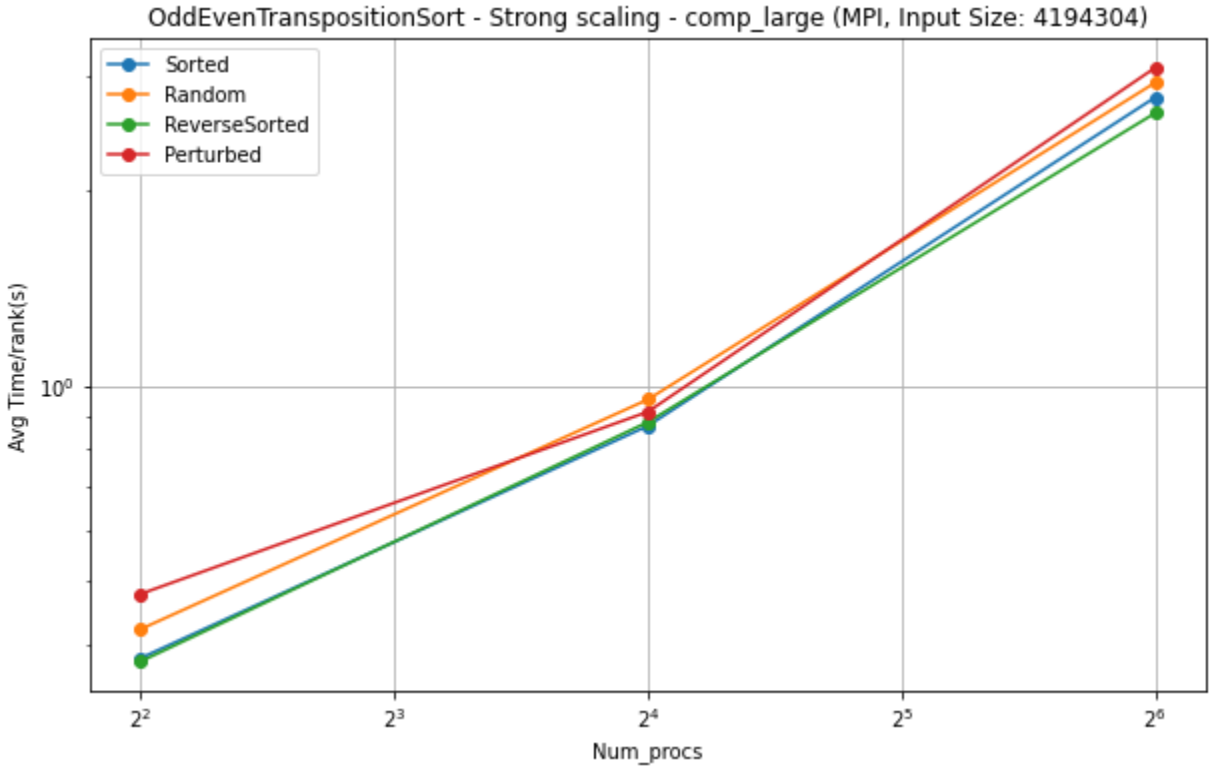
OddEvenTranspositionSort - Strong scaling - main (MPI, Input Size: 262144)



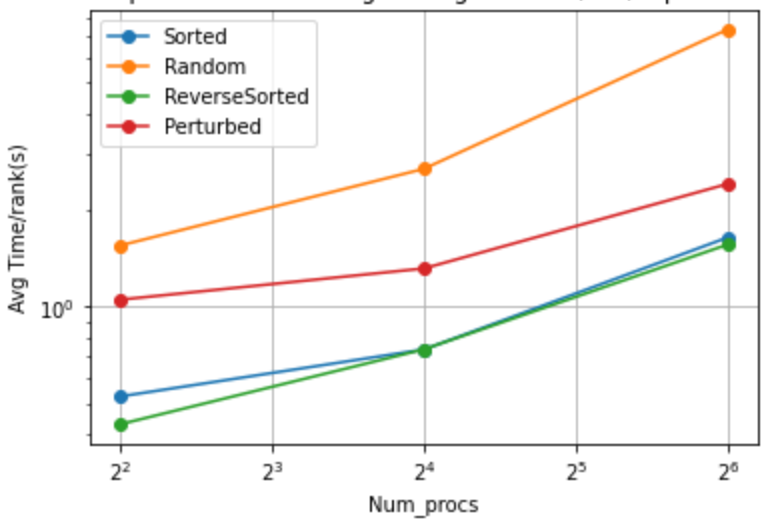

OddEvenTranspositionSort - Strong scaling - comp_large (MPI, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^0
10^{-1}
2^2
2^3
2^4
2^5
2^6
2^7
2^8
Num_procs


OddEvenTranspositionSort - Strong scaling - comm (MPI, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
10^0
10^{-1}
2^2
2^3
2^4
2^5
2^6
2^7
2^8
Num_procs

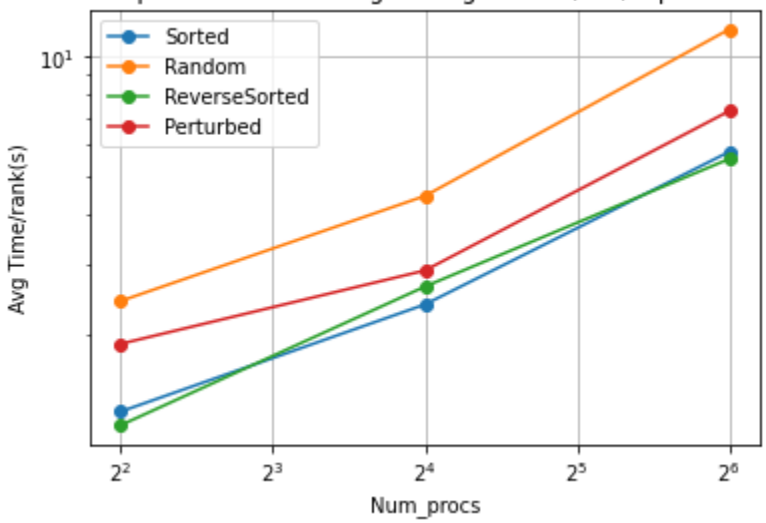

OddEvenTranspositionSort - Strong scaling - main (MPI, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
10^0
2^2
2^3
2^4
2^5
2^6
2^7
2^8
Num_procs

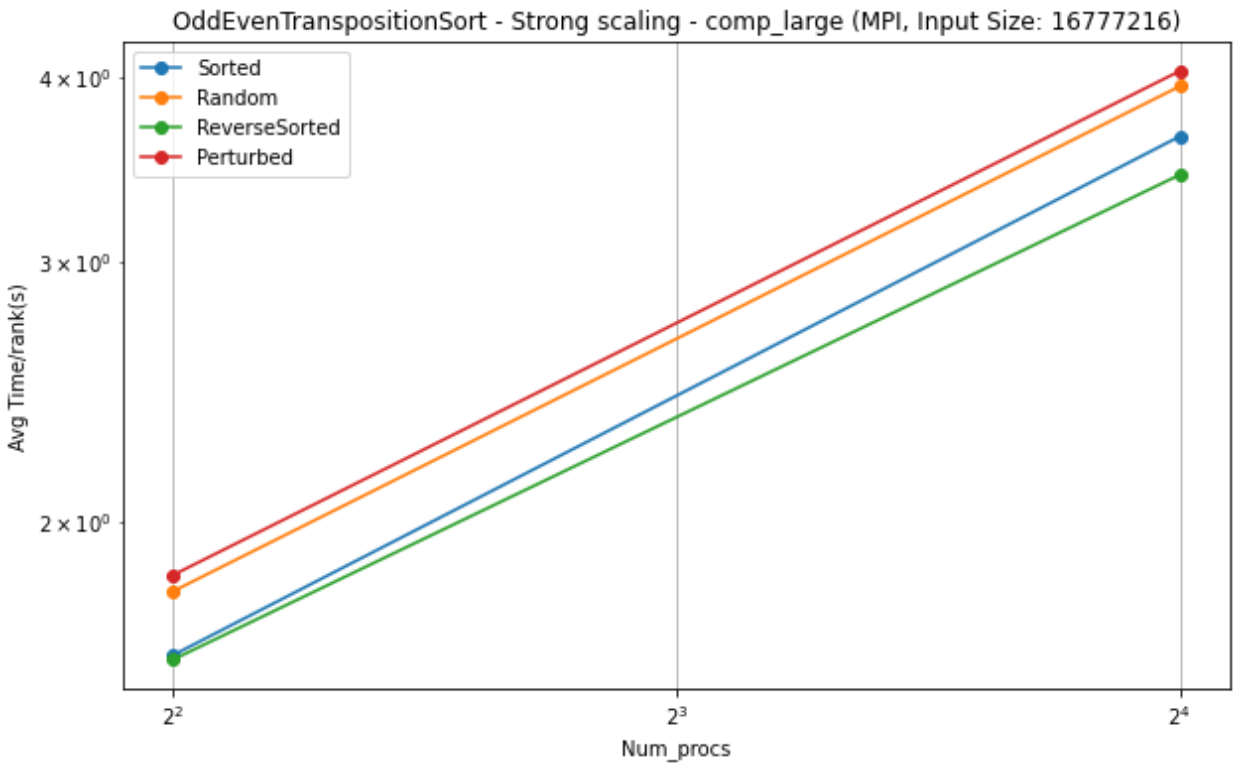

OddEvenTranspositionSort - Strong scaling - comp_large (MPI, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^0
2^2
2^3
2^4
2^5
2^6
Num_procs

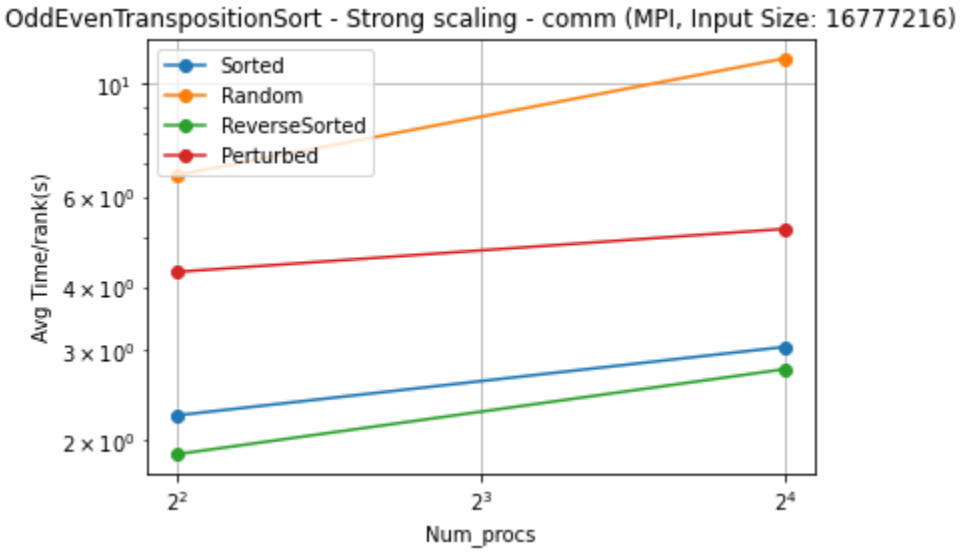
OddEvenTranspositionSort - Strong scaling - comm (MPI, Input Size: 4194304)

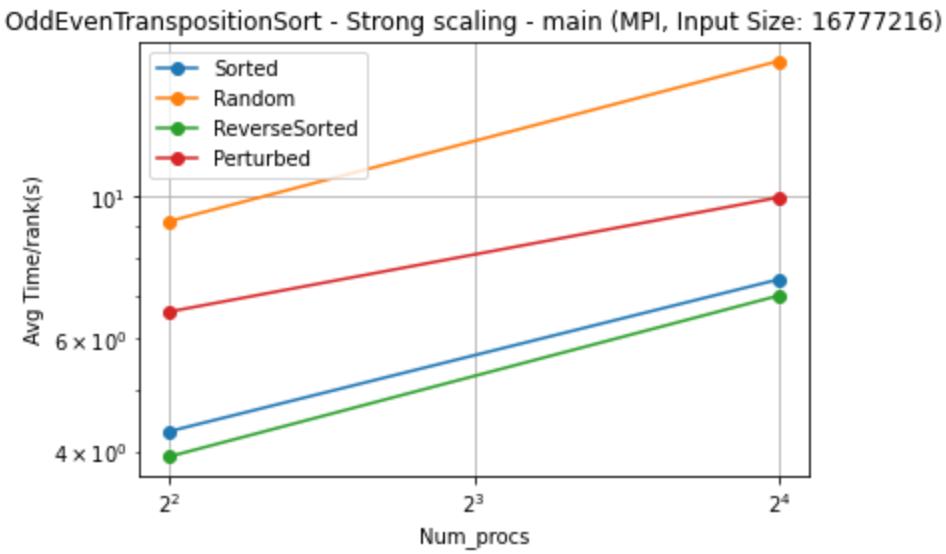


OddEvenTranspositionSort - Strong scaling - main (MPI, Input Size: 4194304)

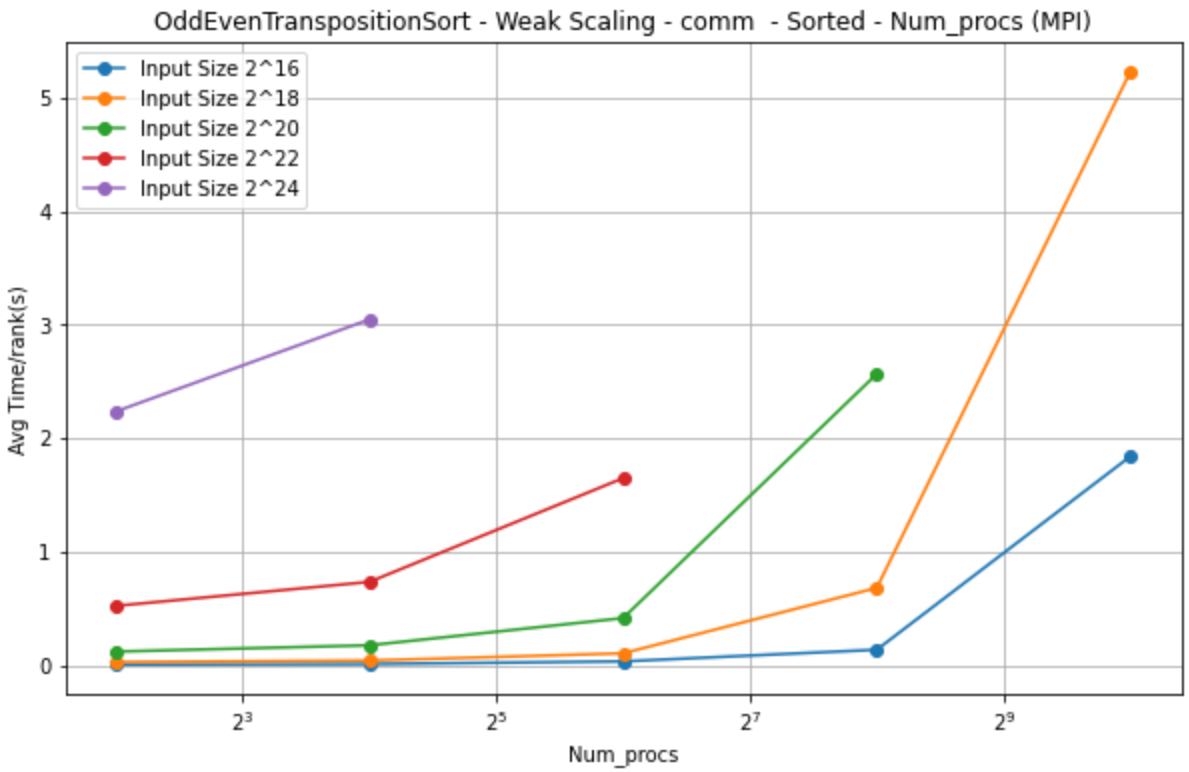


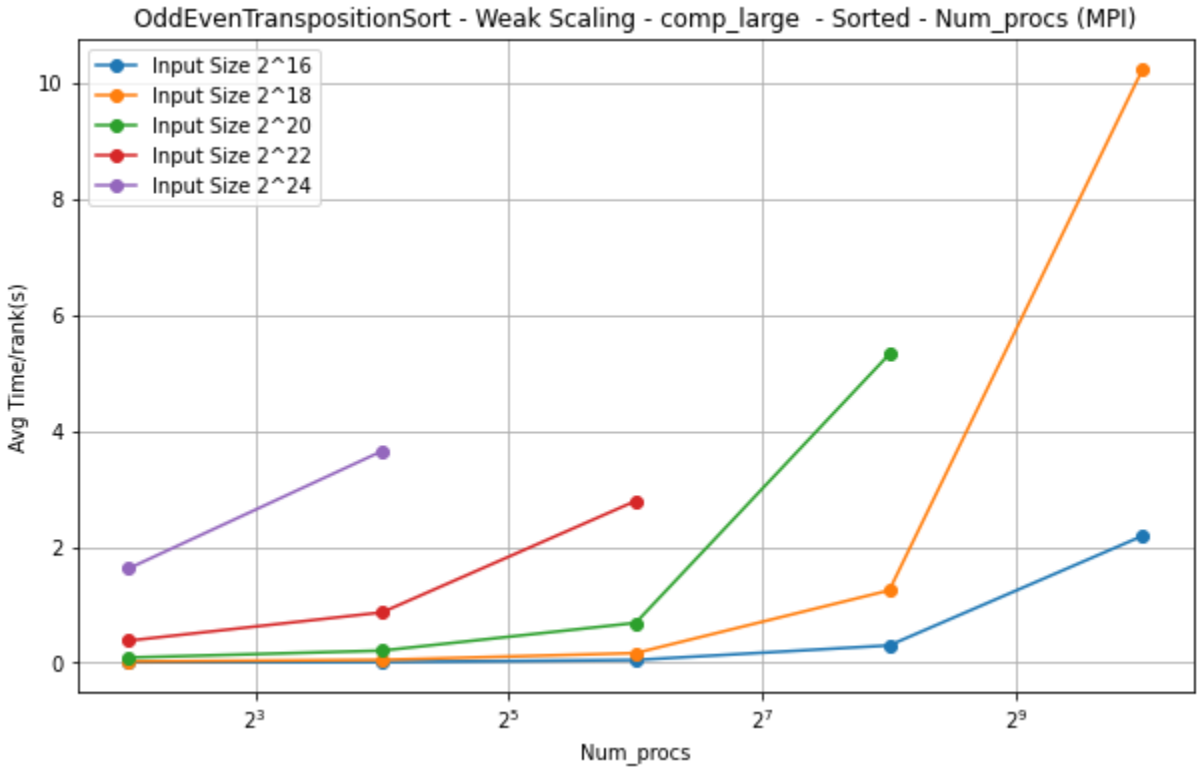

OddEvenTranspositionSort - Strong scaling - comp_large (MPI, Input Size: 16777216)
Sorted
Random
ReverseSorted
Perturbed
4×10^0
3×10^0
2×10^0
Avg Time/rank(s)
2^2
2^3
2^4
Num_procs

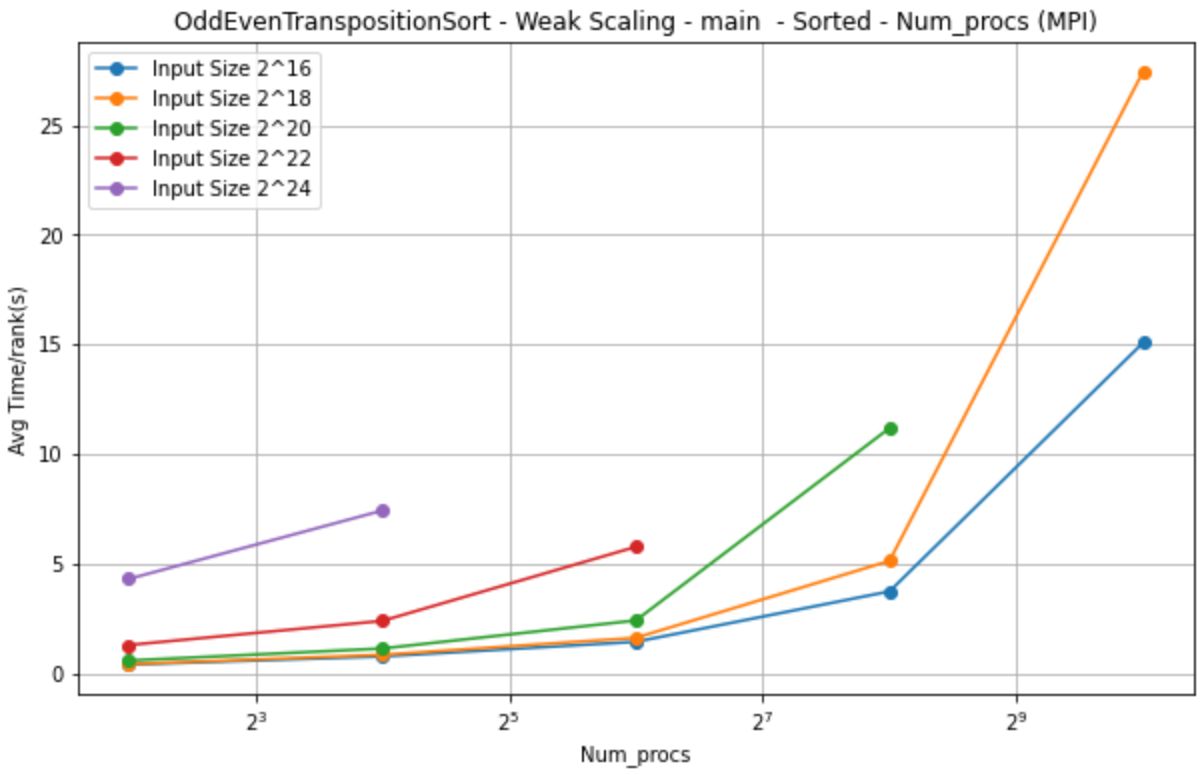

OddEvenTranspositionSort - Strong scaling - comm (MPI, Input Size: 16777216)
Sorted
Random
ReverseSorted
Perturbed
10^1
6×10^0
4×10^0
3×10^0
2×10^0
Avg Time/rank(s)
2^2
2^3
2^4
Num_procs

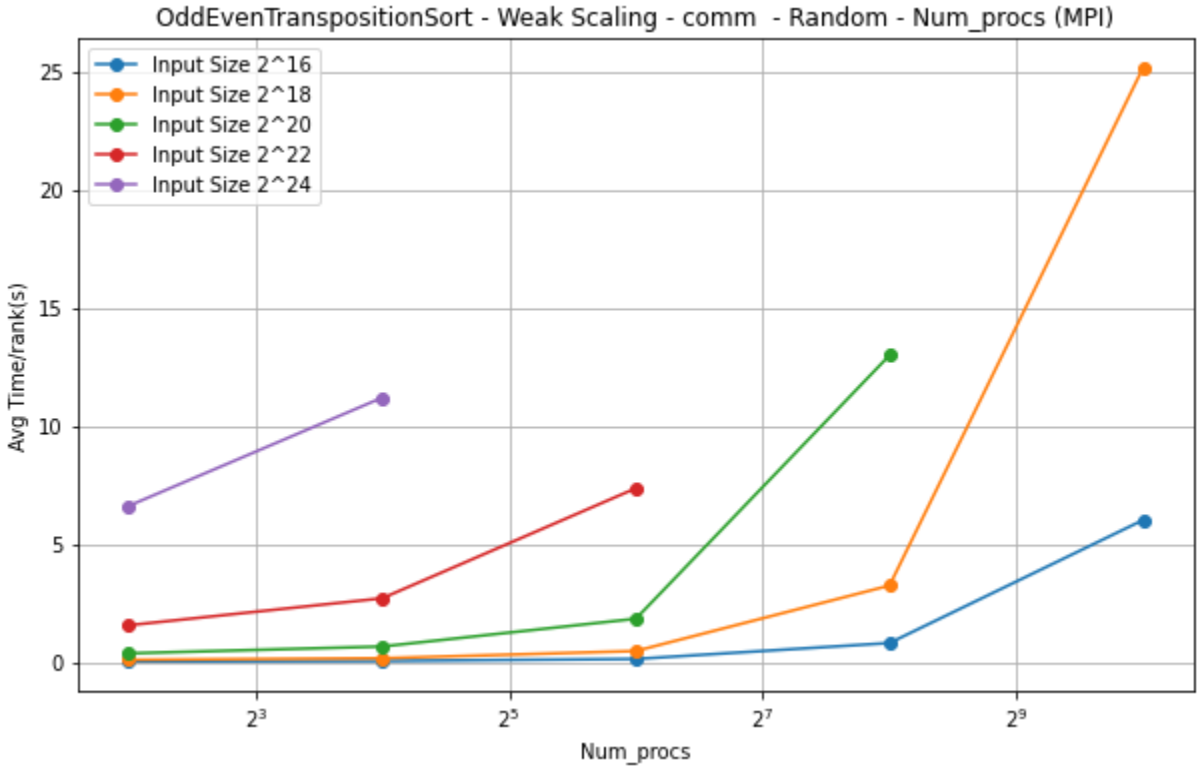


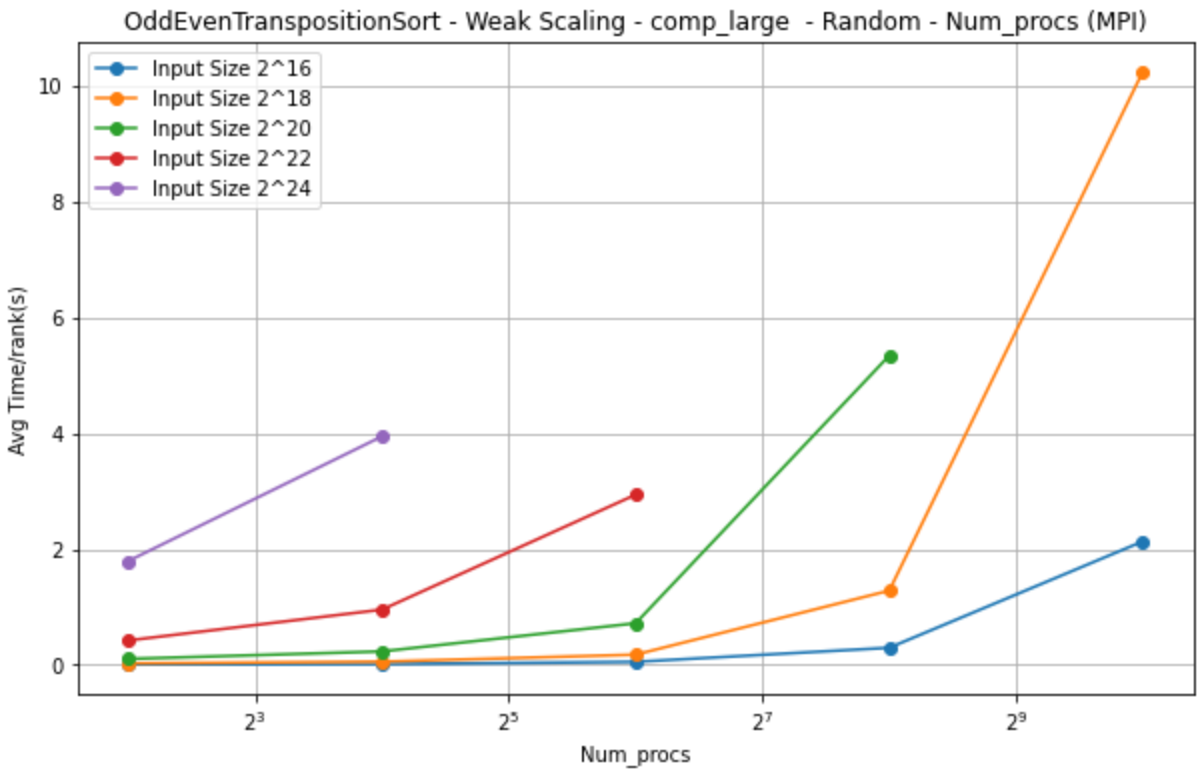
Weak Scaling:

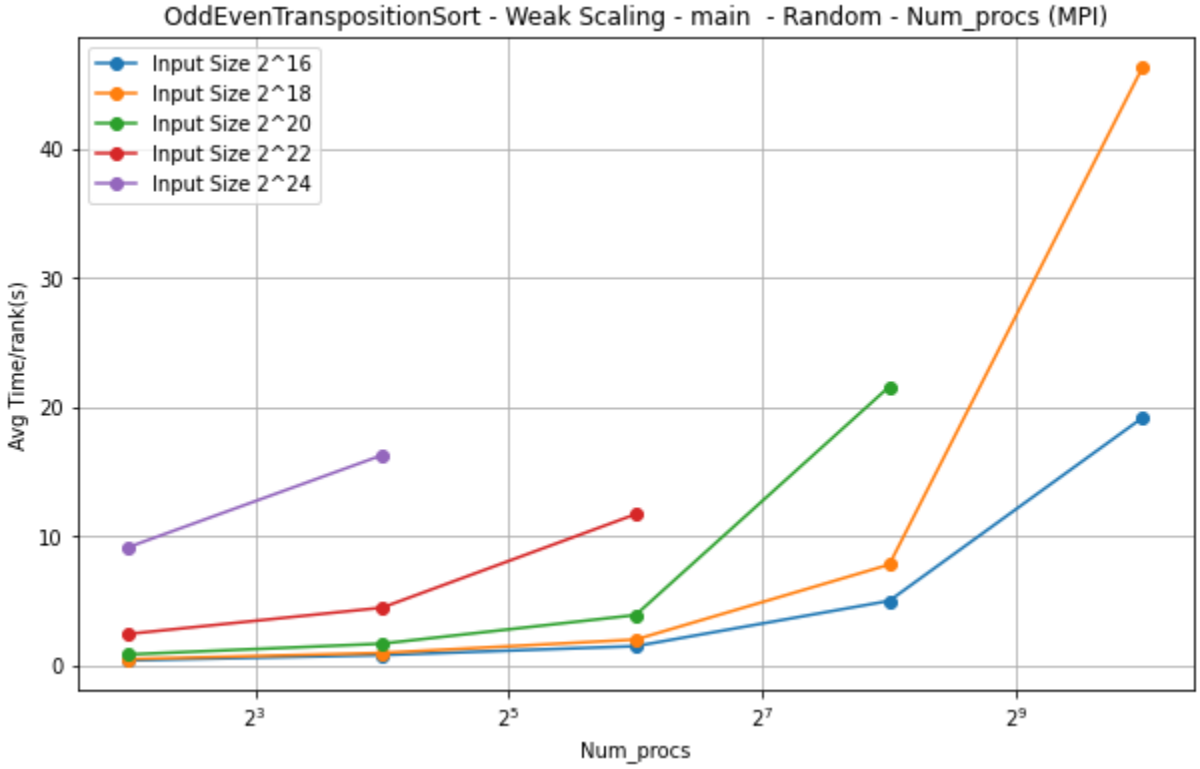

OddEvenTranspositionSort - Weak Scaling - comm - Sorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
0
1
2
3
4
5
2^3
2^5
2^7
2^9
Num_procs

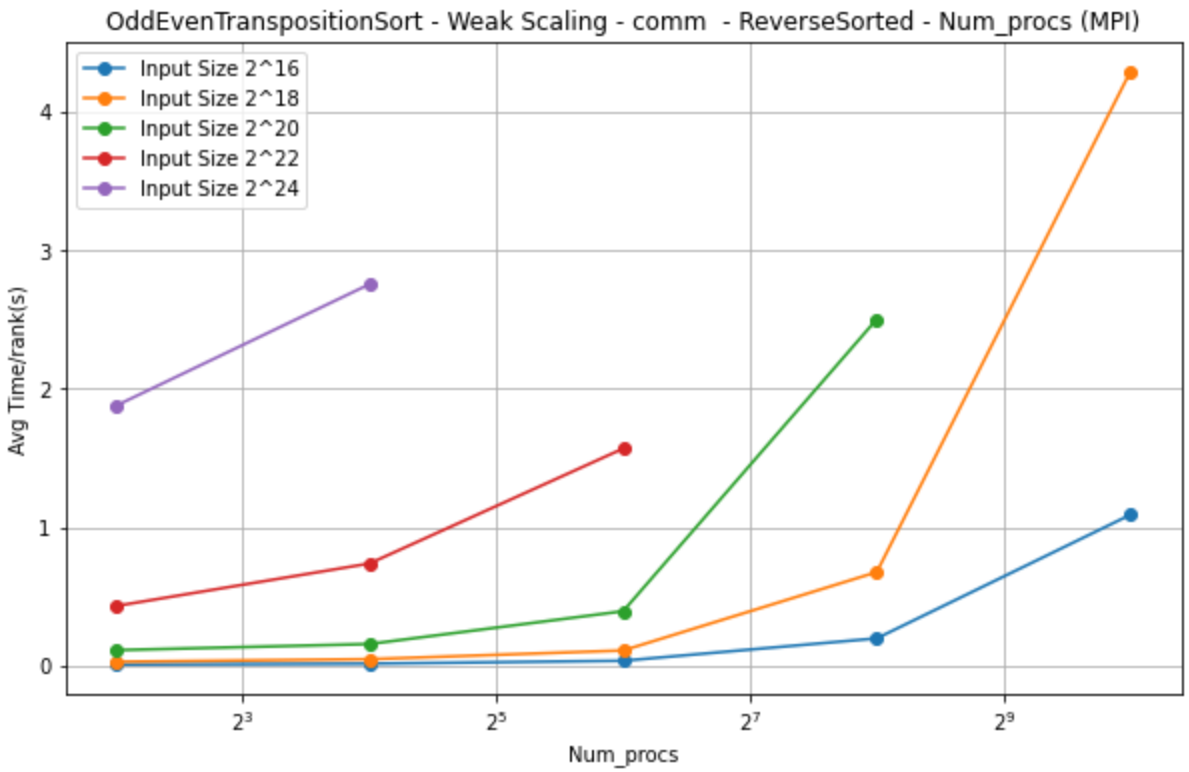

OddEvenTranspositionSort - Weak Scaling - comp_large - Sorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
0
2
4
6
8
10
2^3
2^5
2^7
2^9
Num_procs

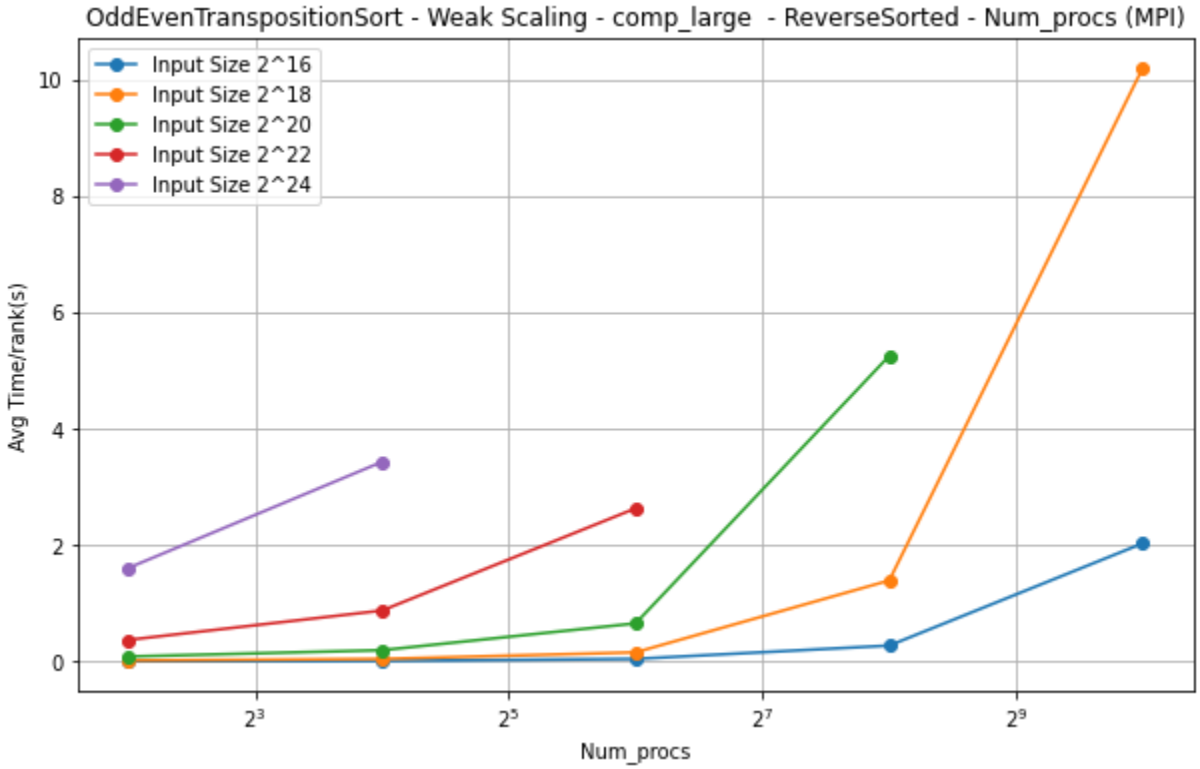

OddEvenTranspositionSort - Weak Scaling - main - Sorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
0
5
10
15
20
25
2^3
2^5
2^7
2^9
Num_procs

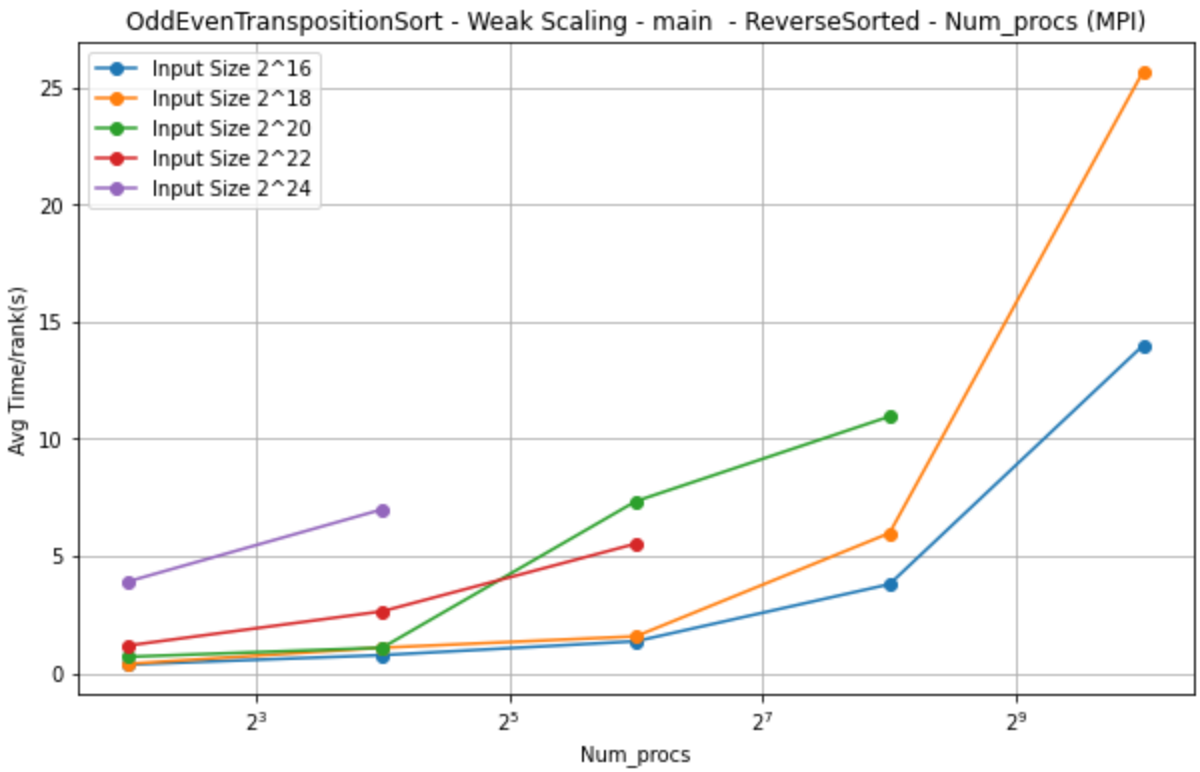

OddEvenTranspositionSort - Weak Scaling - comm - Random - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
0
5
10
15
20
25
2^3
2^5
2^7
2^9
Num_procs

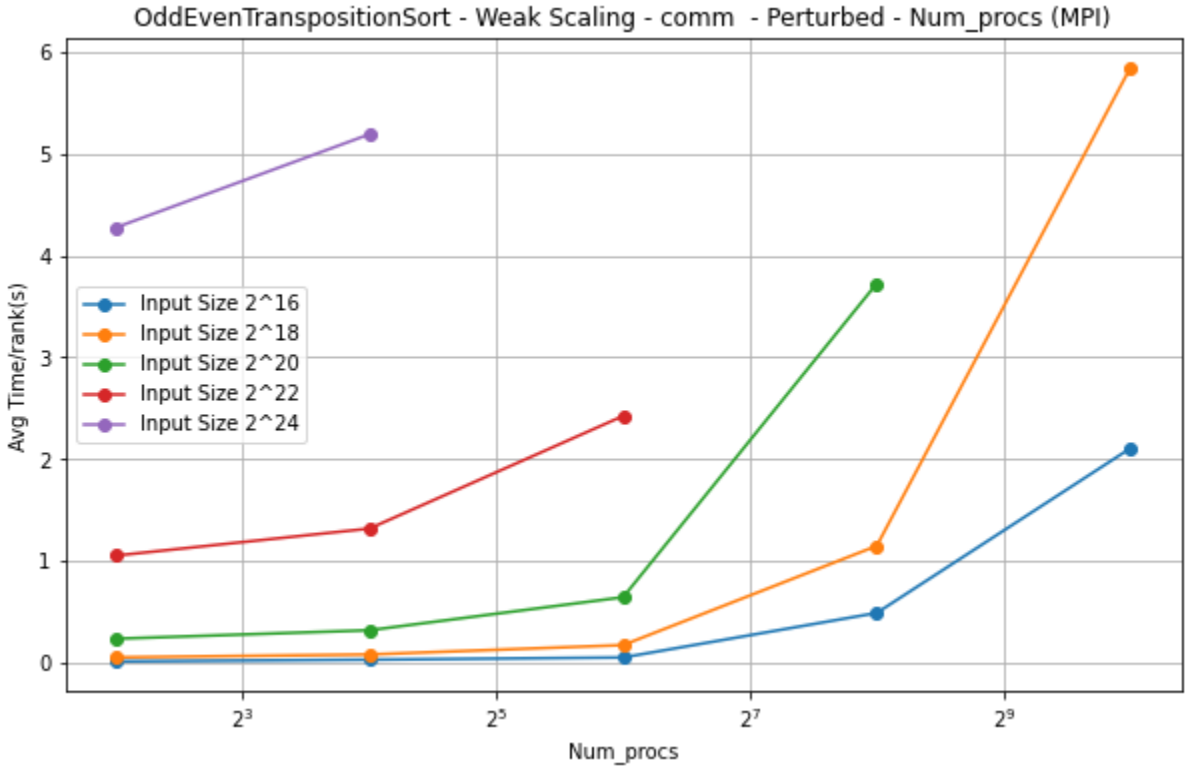

OddEvenTranspositionSort - Weak Scaling - comp_large - Random - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
0
2
4
6
8
10
2^3
2^5
2^7
2^9
Num_procs

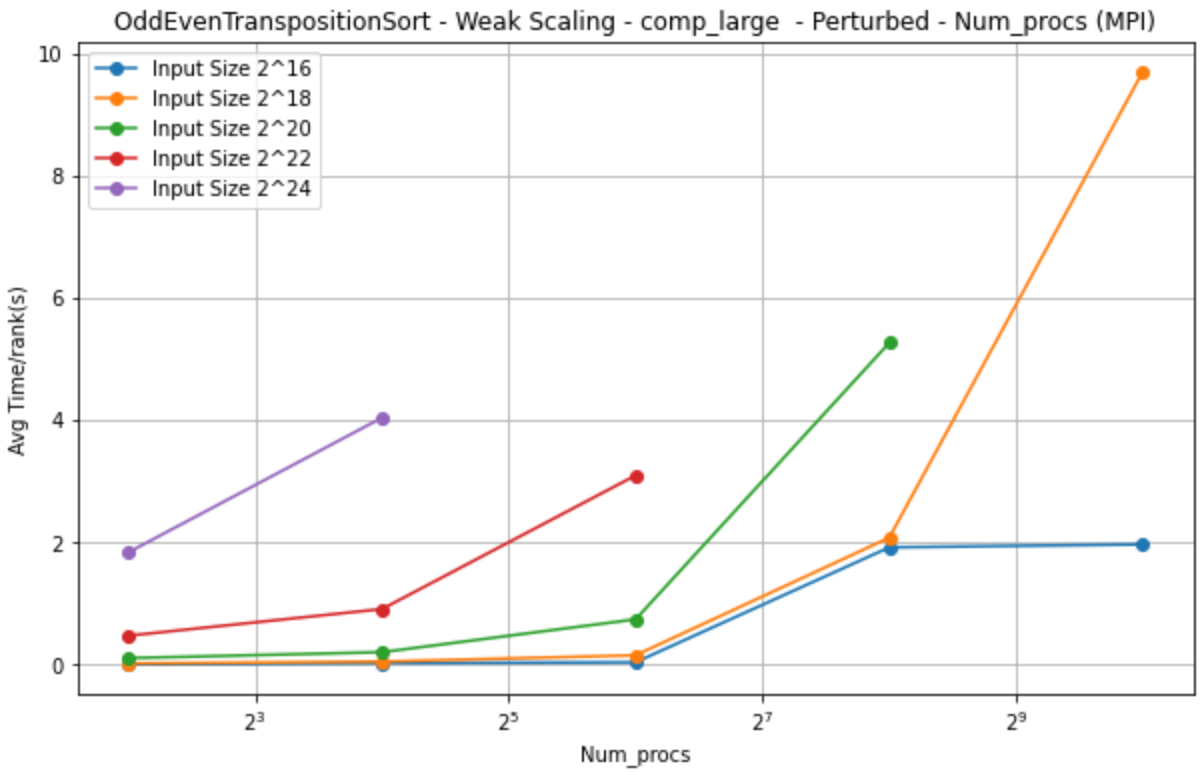

OddEvenTranspositionSort - Weak Scaling - main - Random - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
0
10
20
30
40
2^3
2^5
2^7
2^9
Num_procs

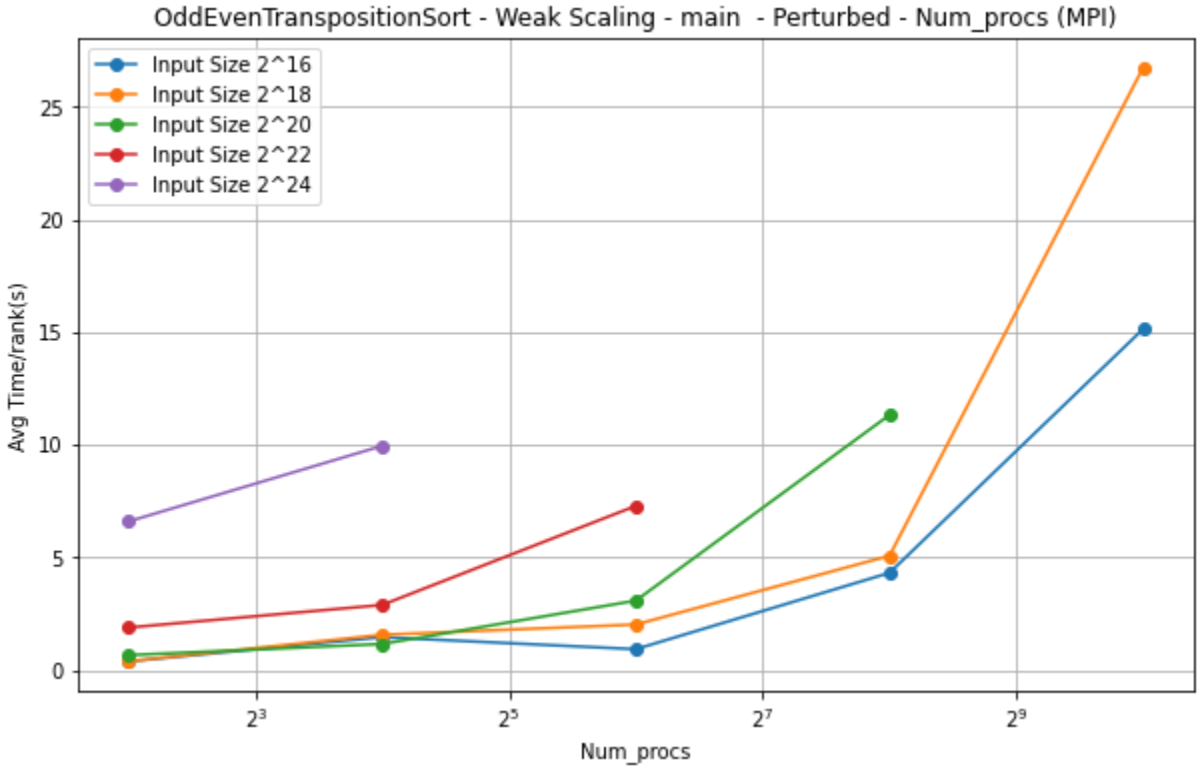

OddEvenTranspositionSort - Weak Scaling - comm - ReverseSorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
Num_procs
0
1
2
3
4
2^3
2^5
2^7
2^9


OddEvenTranspositionSort - Weak Scaling - comp_large - ReverseSorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
Num_procs
0
2
4
6
8
10
2^3
2^5
2^7
2^9

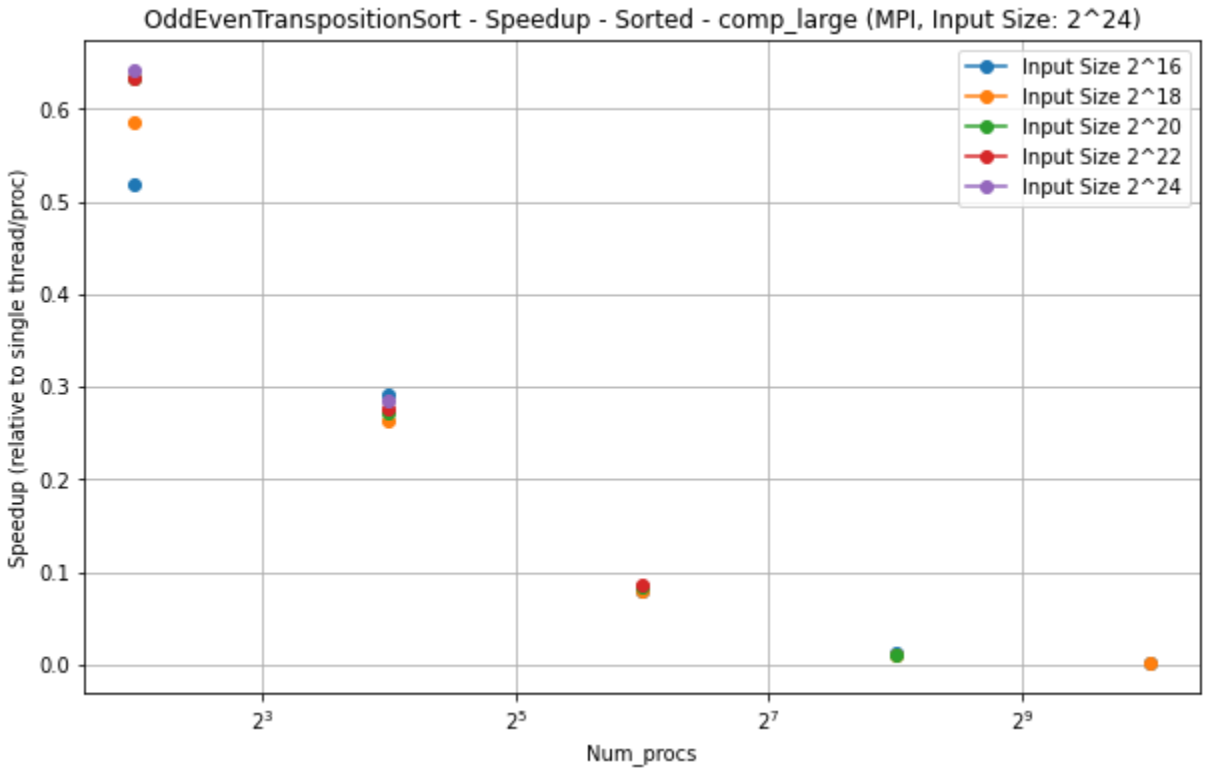

OddEvenTranspositionSort - Weak Scaling - main - ReverseSorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
Num_procs

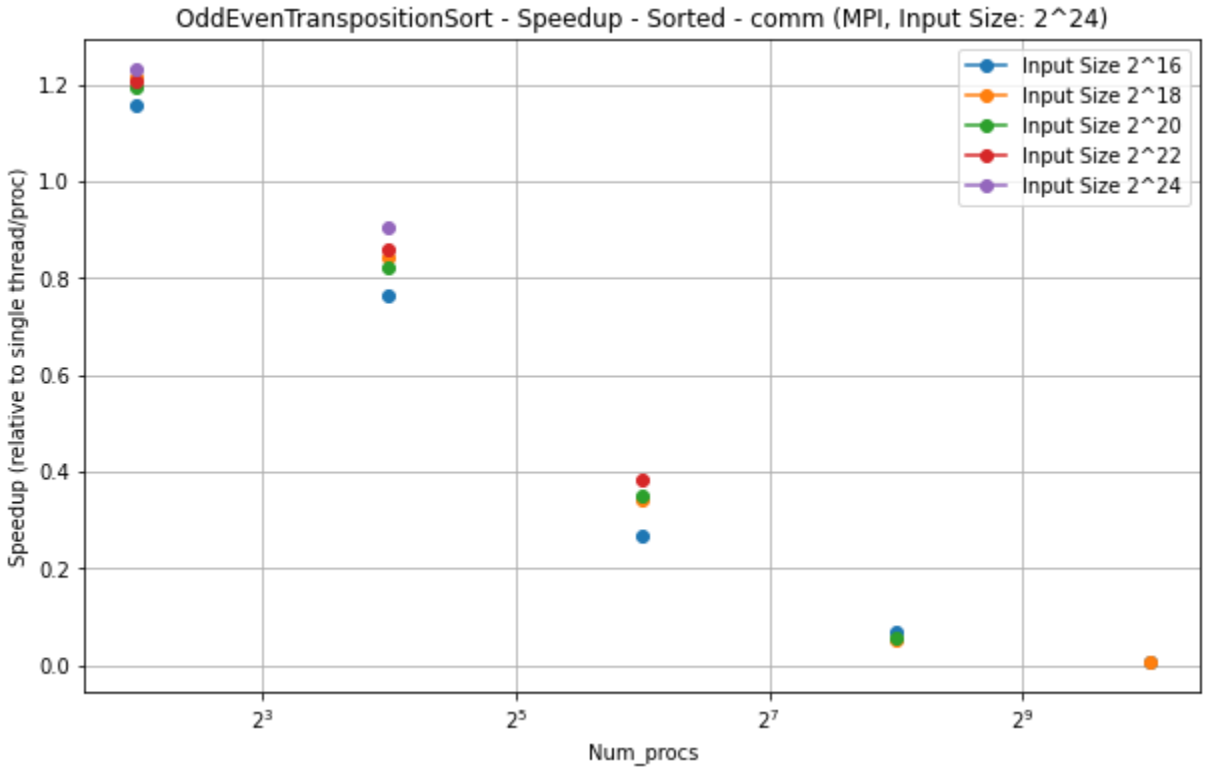

OddEvenTranspositionSort - Weak Scaling - comm - Perturbed - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
Num_procs

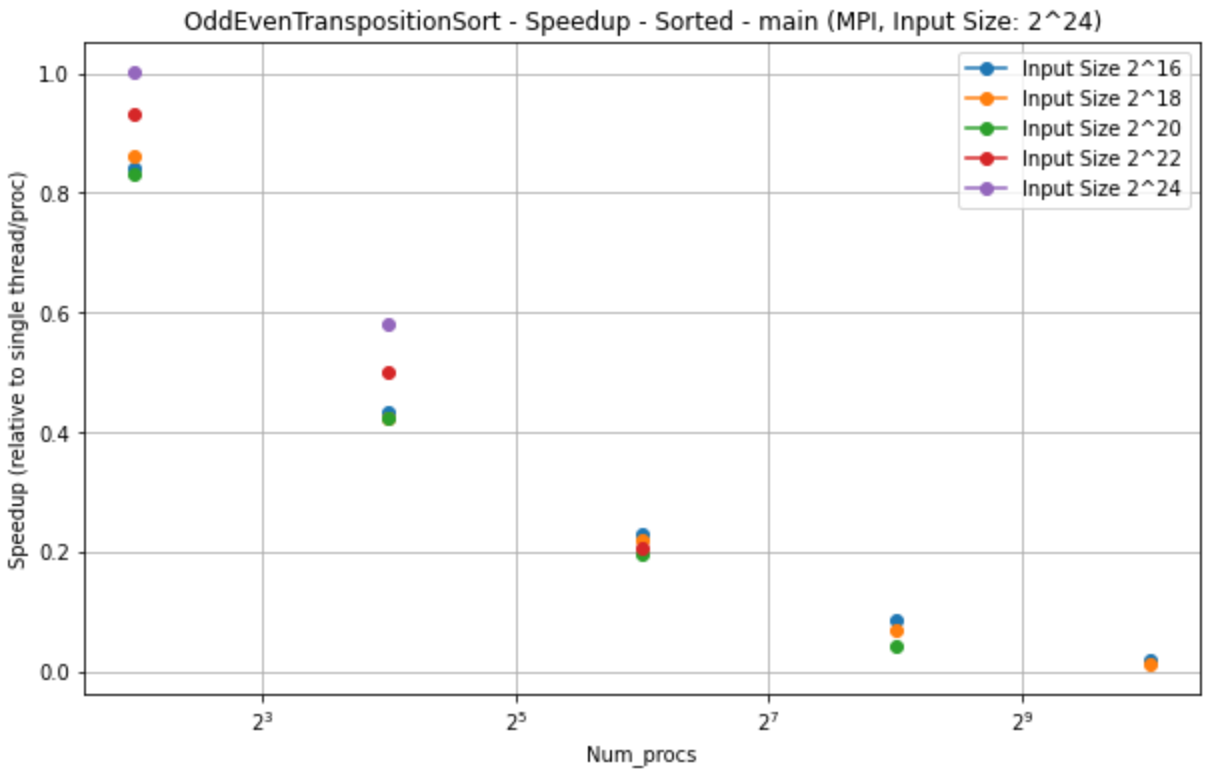

OddEvenTranspositionSort - Weak Scaling - comp_large - Perturbed - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
Num_procs
0
2
4
6
8
10
2^3
2^5
2^7
2^9

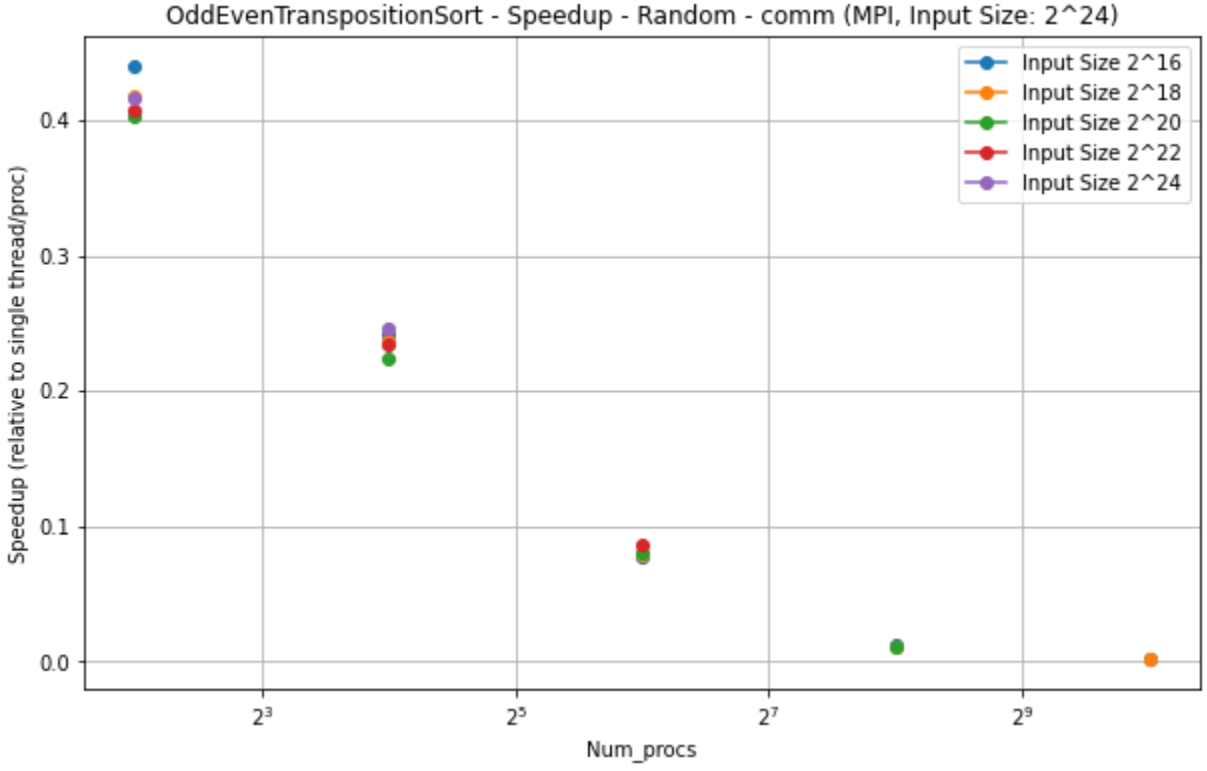

OddEvenTranspositionSort - Weak Scaling - main - Perturbed - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
Num_procs
0
5
10
15
20
25
2^3
2^5
2^7
2^9

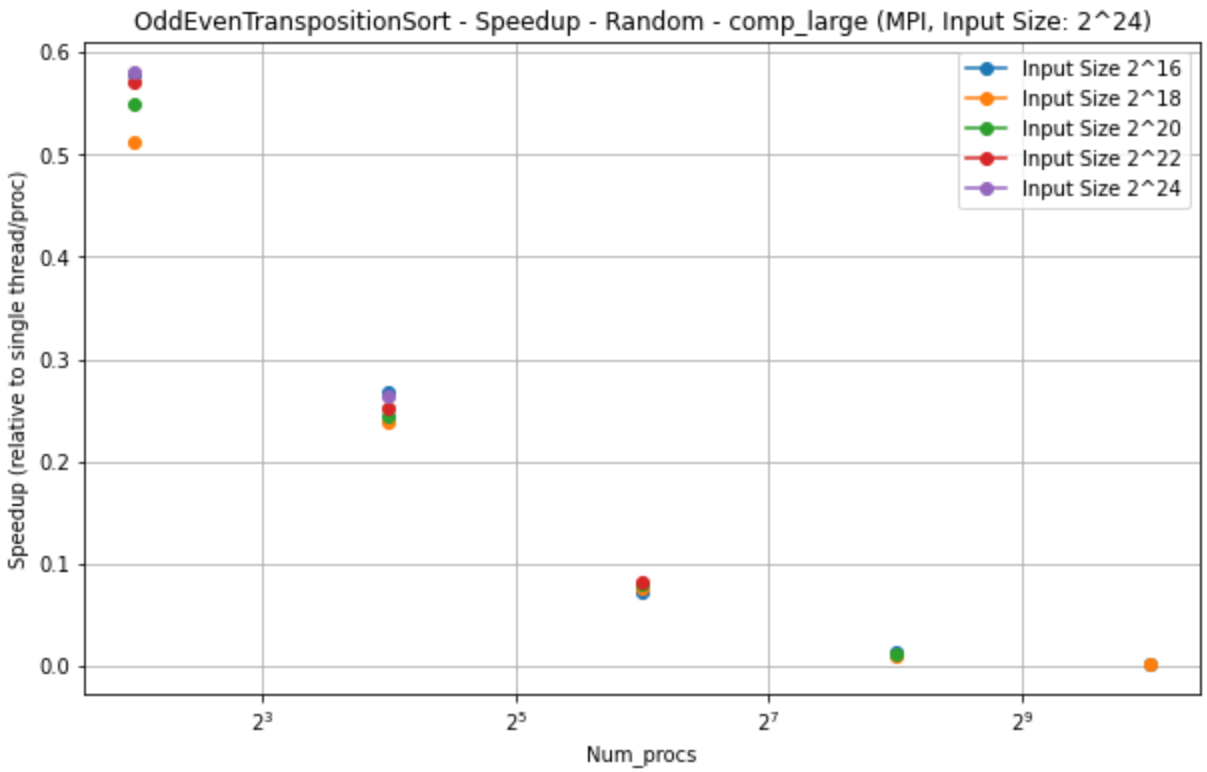
Speedup:

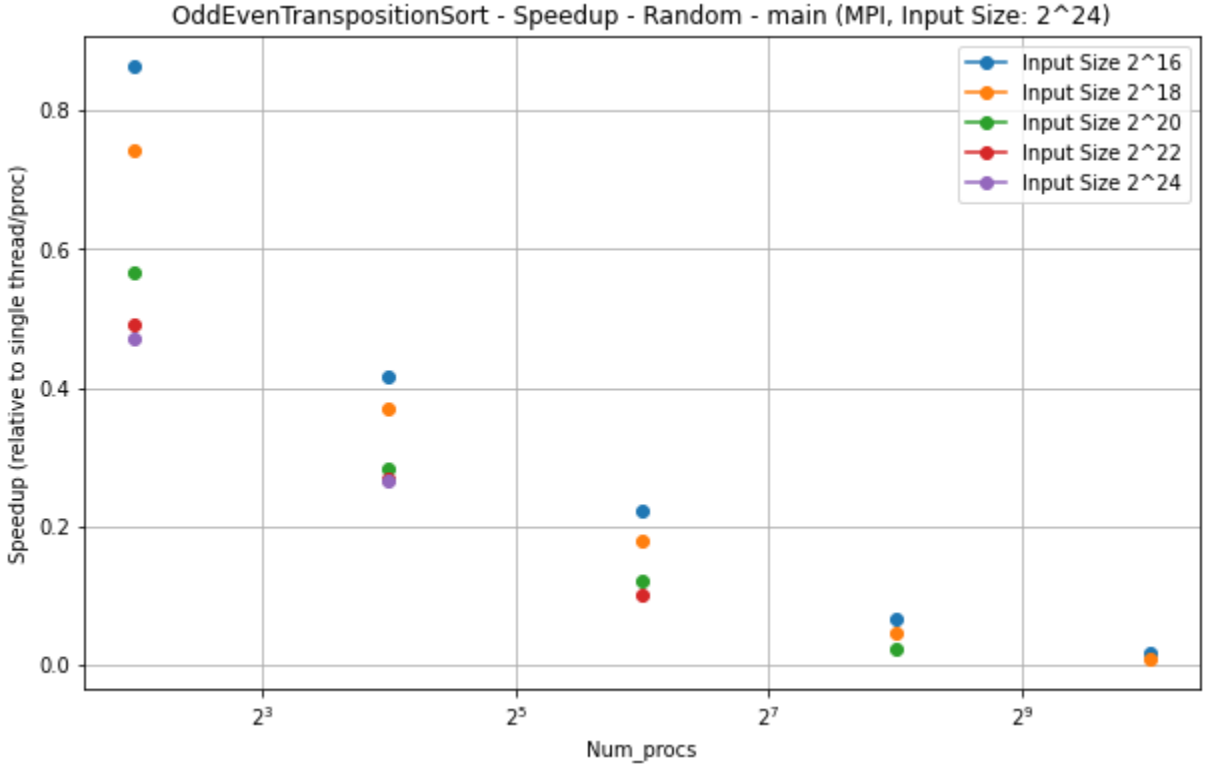

OddEvenTranspositionSort - Speedup - Sorted - comp_large (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2^3
2^5
2^7
2^9
Num_procs

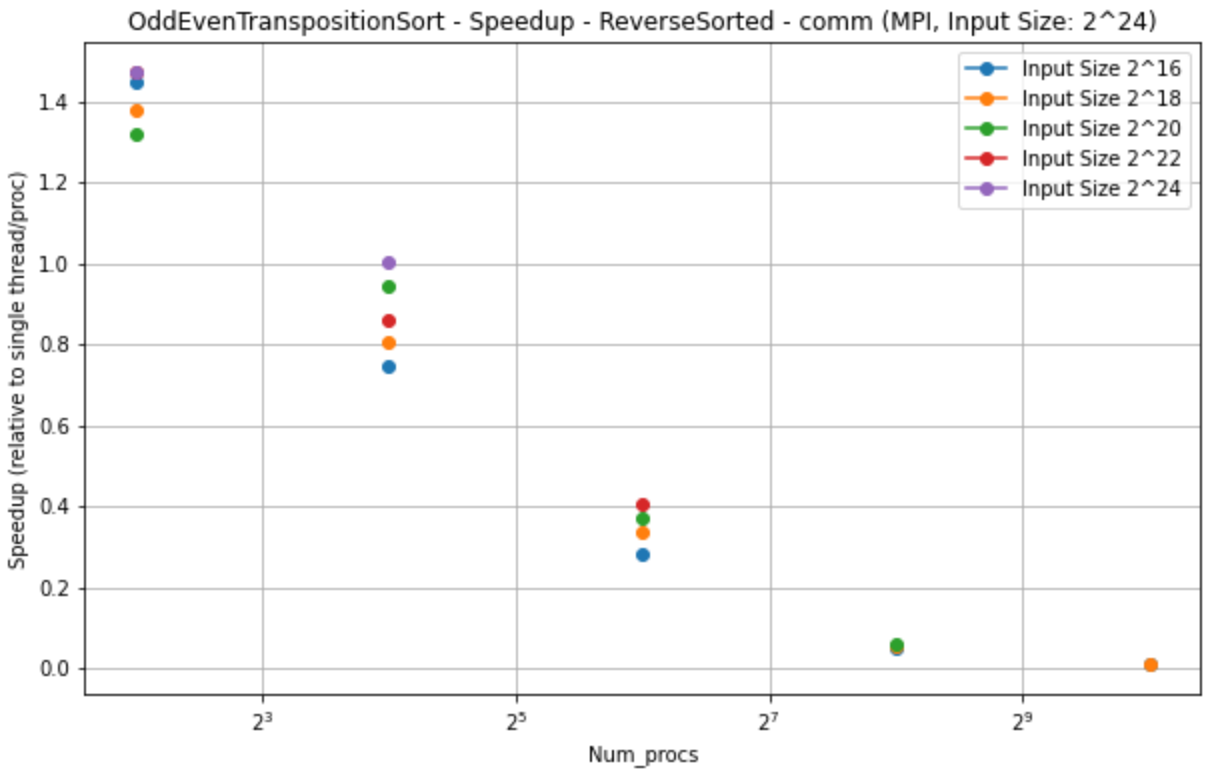

OddEvenTranspositionSort - Speedup - Sorted - comm (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
2^3
2^5
2^7
2^9
Num_procs

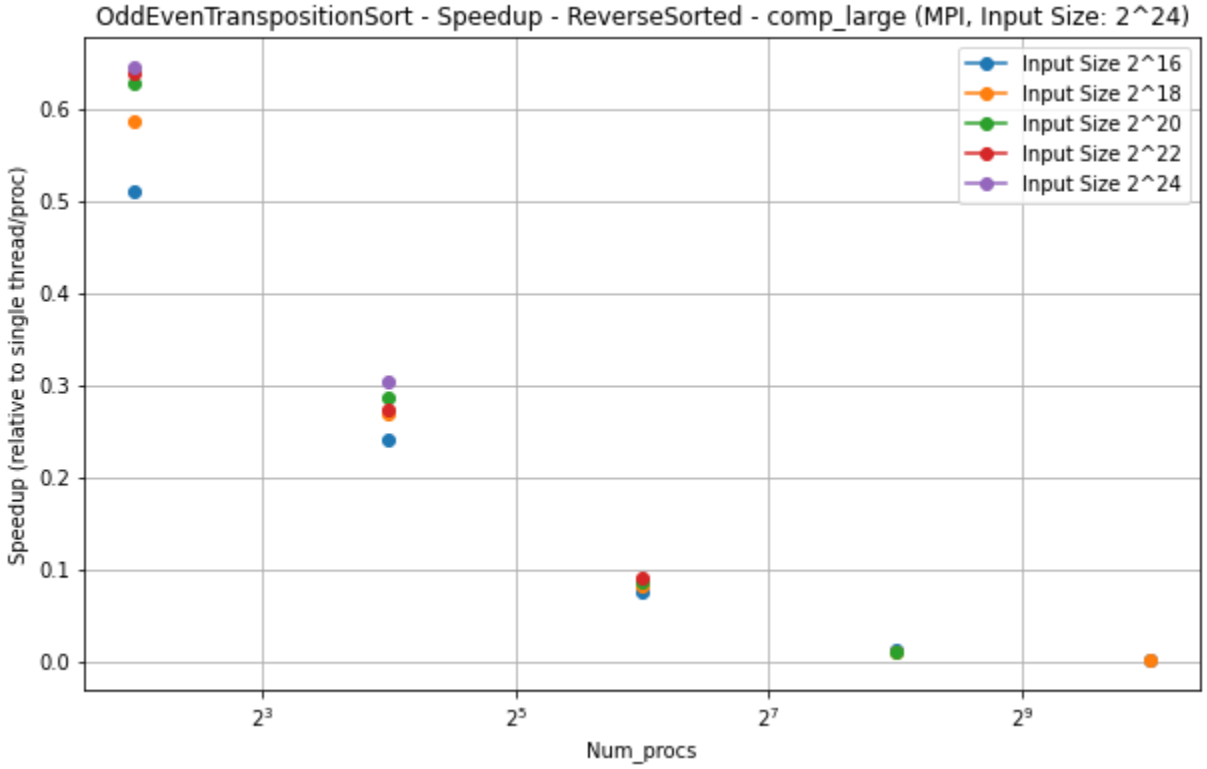

OddEvenTranspositionSort - Speedup - Sorted - main (MPI, Input Size: 2^24)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
1.0
0.8
0.6
0.4
0.2
0.0
2^3
2^5
2^7
2^9

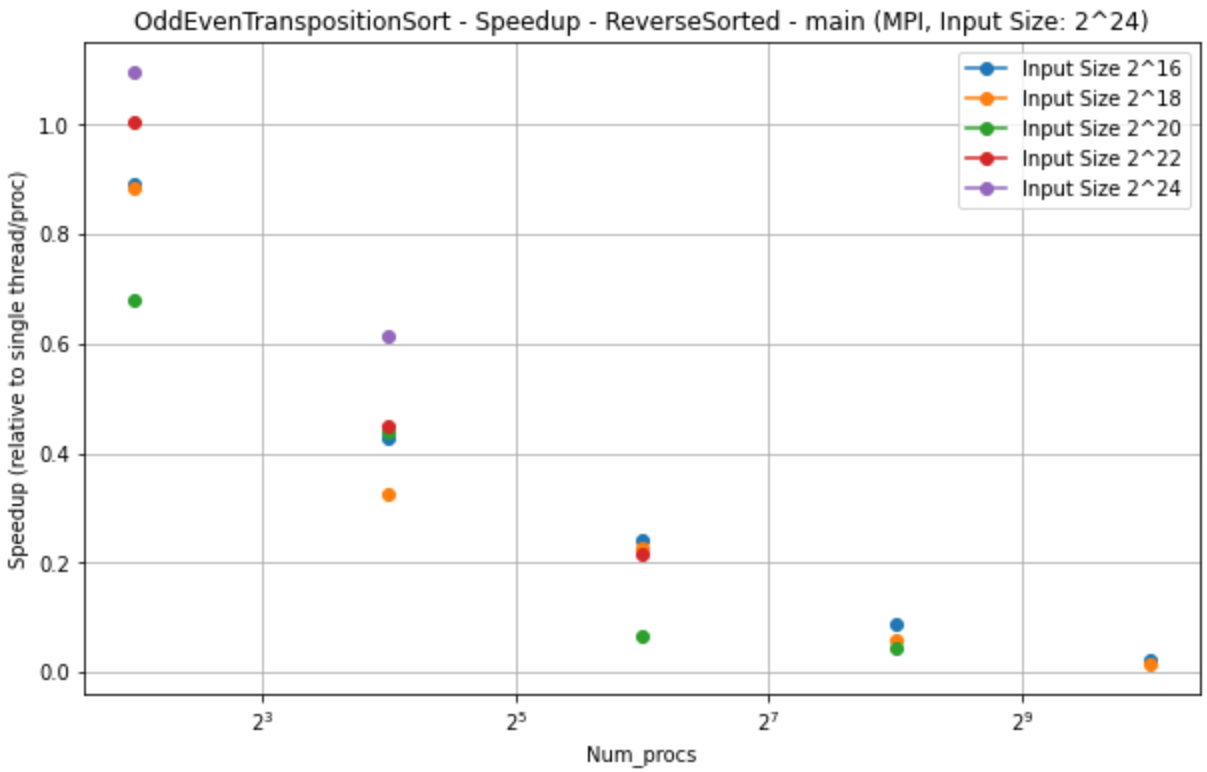

OddEvenTranspositionSort - Speedup - Random - comm (MPI, Input Size: 2^24)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
0.4
0.3
0.2
0.1
0.0
2^3
2^5
2^7
2^9

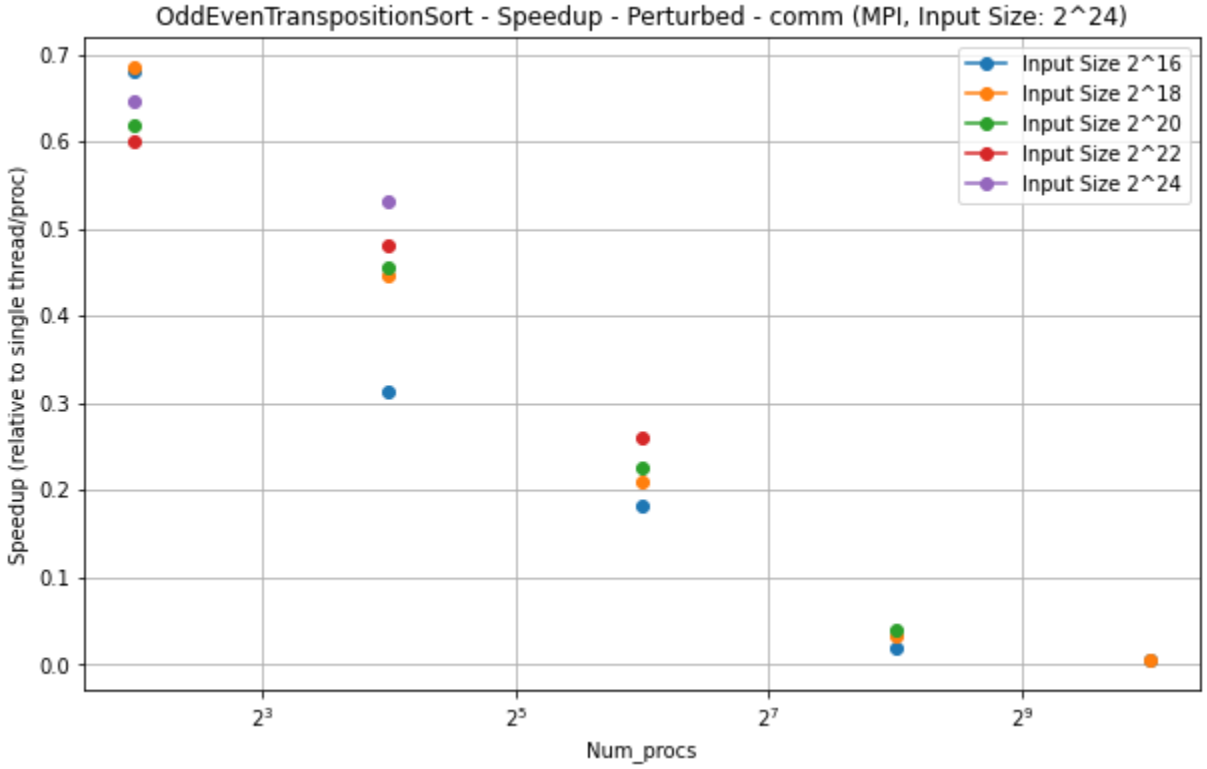

OddEvenTranspositionSort - Speedup - Random - comp_large (MPI, Input Size: 2^24)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2^3
2^5
2^7
2^9

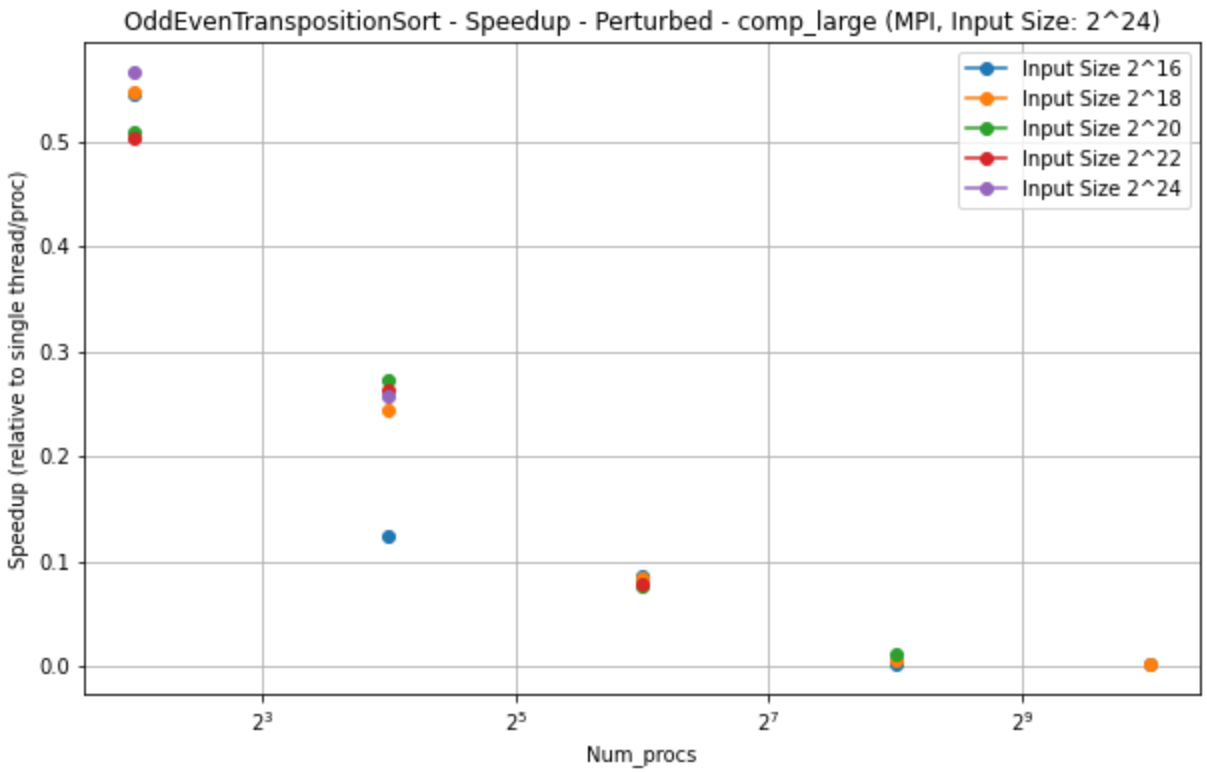

OddEvenTranspositionSort - Speedup - Random - main (MPI, Input Size: 2^24)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
0.0
0.2
0.4
0.6
0.8
2^3
2^5
2^7
2^9

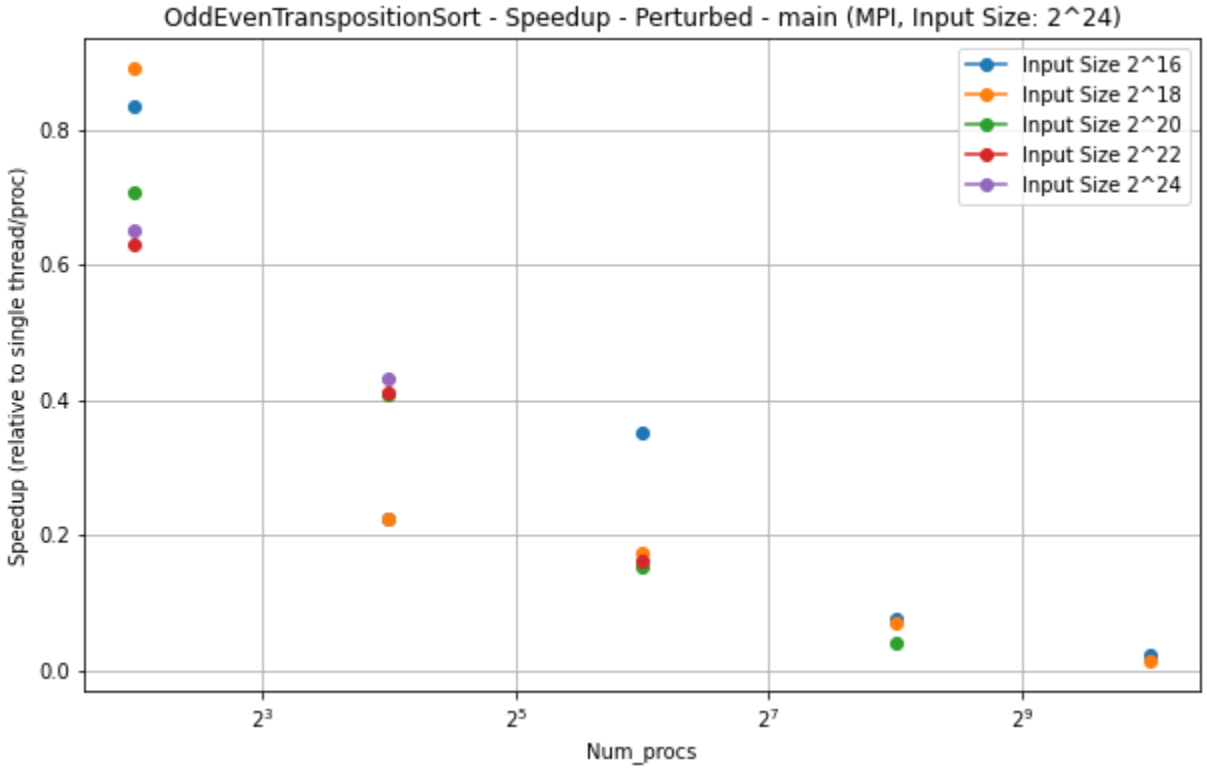

OddEvenTranspositionSort - Speedup - ReverseSorted - comm (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
2^3
2^5
2^7
2^9
Num_procs


OddEvenTranspositionSort - Speedup - ReverseSorted - comp_large (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2^3
2^5
2^7
2^9
Num_procs

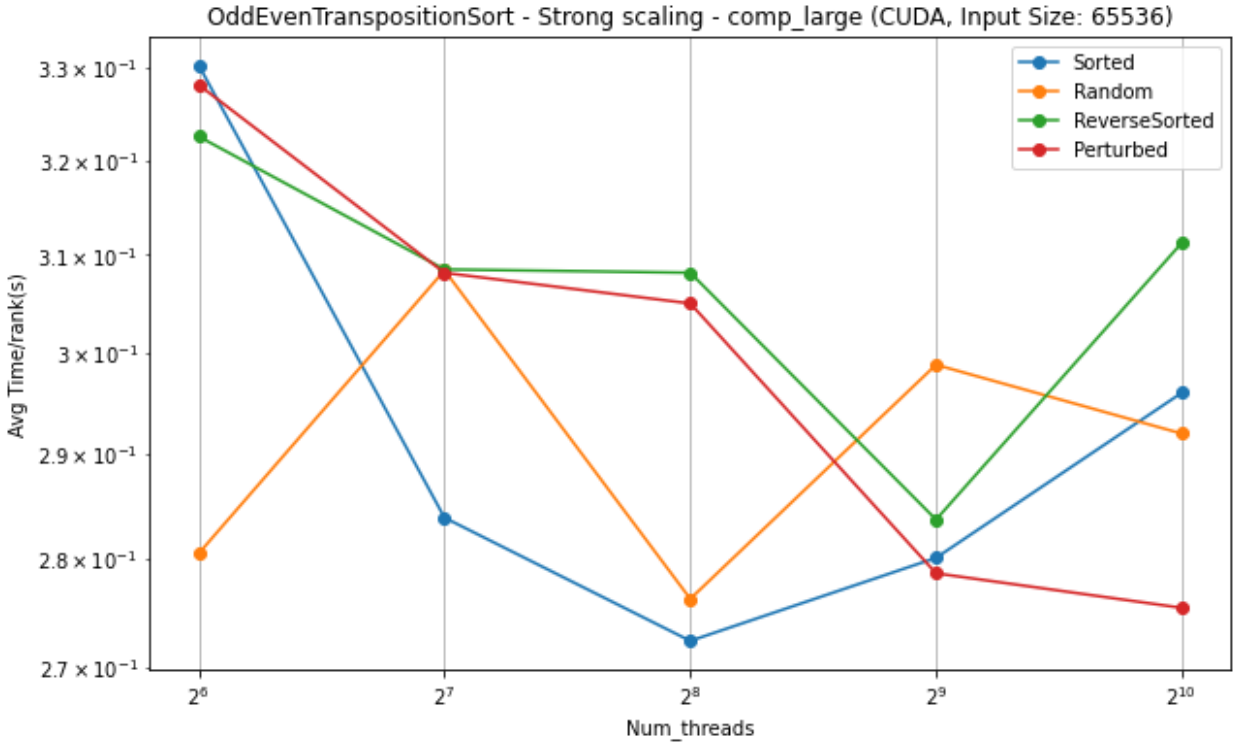

OddEvenTranspositionSort - Speedup - ReverseSorted - main (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
0.0
0.2
0.4
0.6
0.8
1.0
2^3
2^5
2^7
2^9
Num_procs

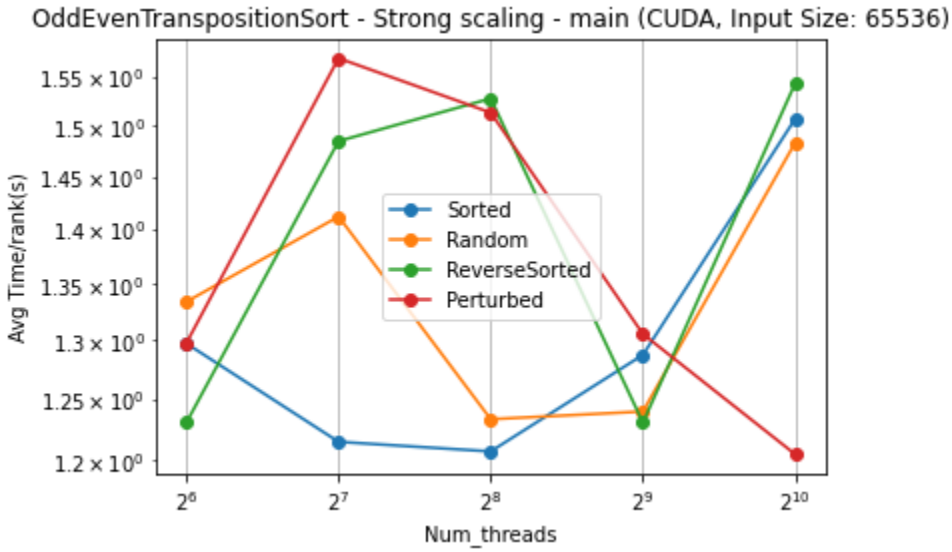

OddEvenTranspositionSort - Speedup - Perturbed - comm (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
2^3
2^5
2^7
2^9
Num_procs

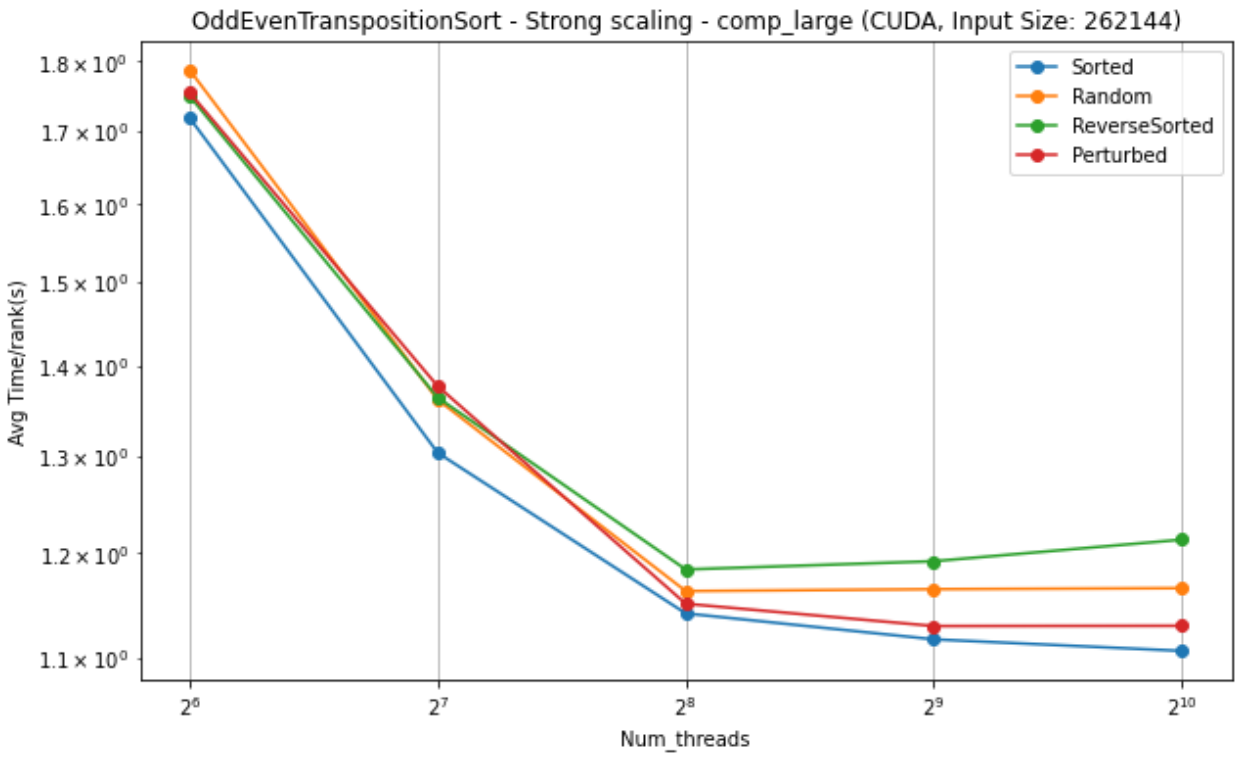

OddEvenTranspositionSort - Speedup - Perturbed - comp_large (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
0.0
0.1
0.2
0.3
0.4
0.5
2^3
2^5
2^7
2^9
Num_procs

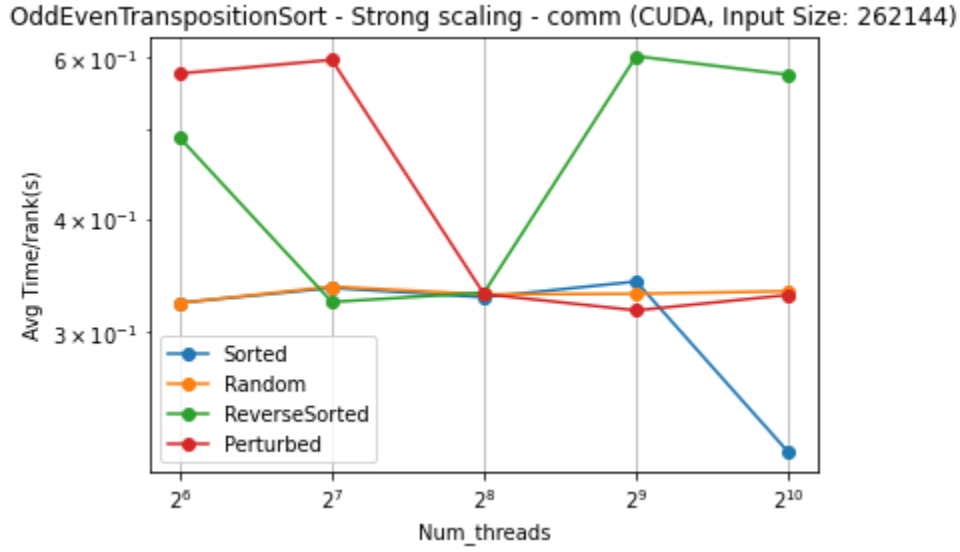

OddEvenTranspositionSort - Speedup - Perturbed - main (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
0.0
0.2
0.4
0.6
0.8
2^3
2^5
2^7
2^9
Num_procs

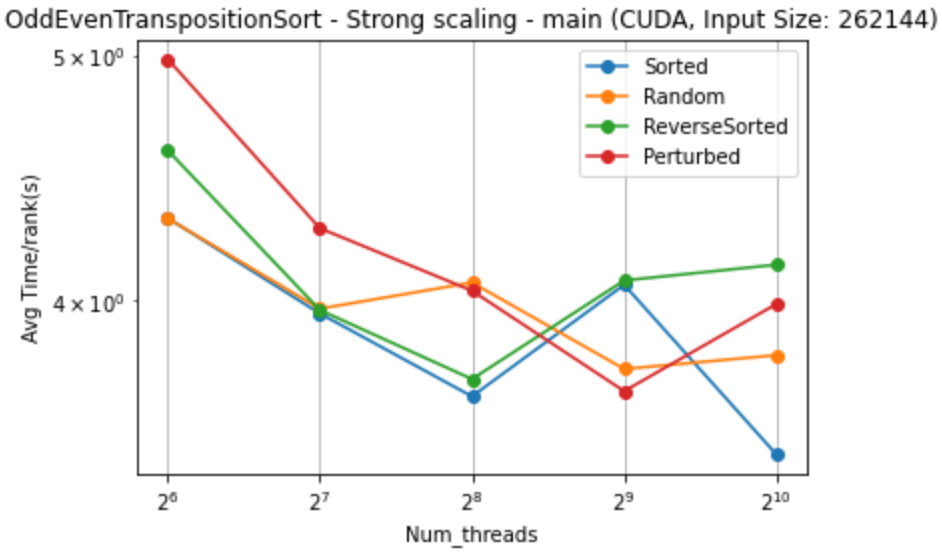
CUDA:
Strong Scaling:

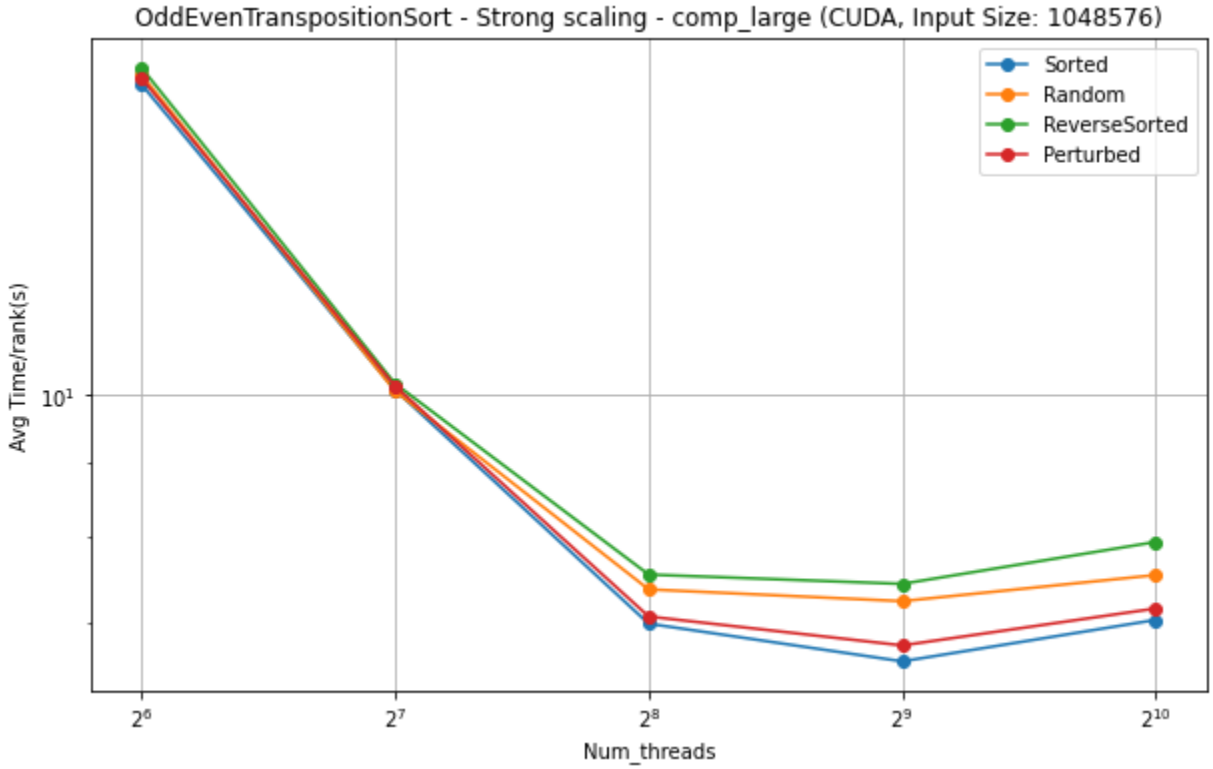




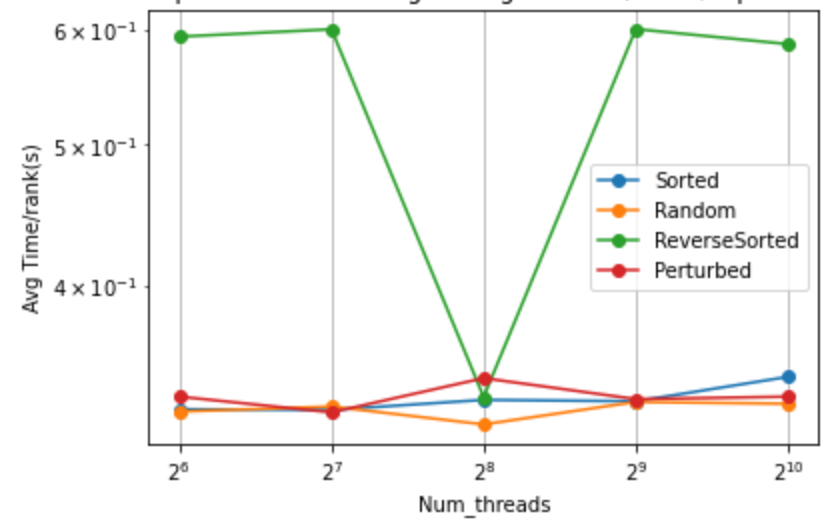

OddEvenTranspositionSort - Strong scaling - comp_large (CUDA, Input Size: 262144)
Avg Time/rank(s)
Num_threads
Sorted
Random
ReverseSorted
Perturbed


OddEvenTranspositionSort - Strong scaling - comm (CUDA, Input Size: 262144)
Avg Time/rank(s)
Num_threads
Sorted
Random
ReverseSorted
Perturbed

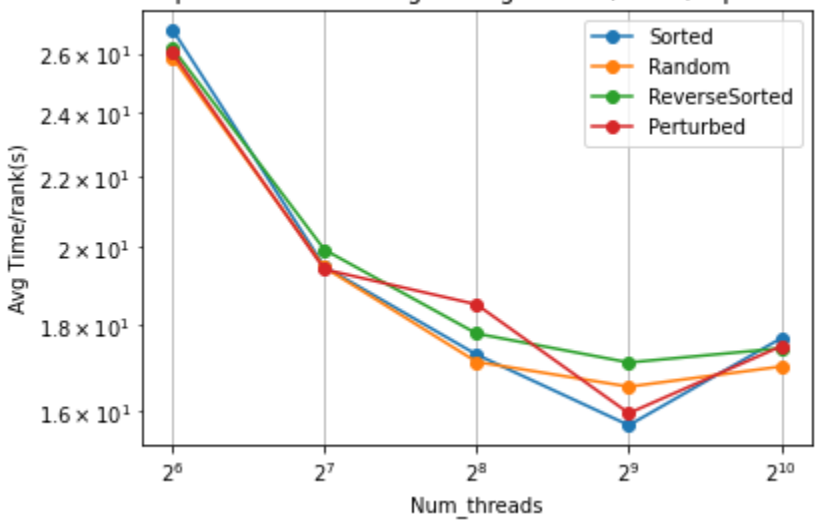

OddEvenTranspositionSort - Strong scaling - main (CUDA, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_threads

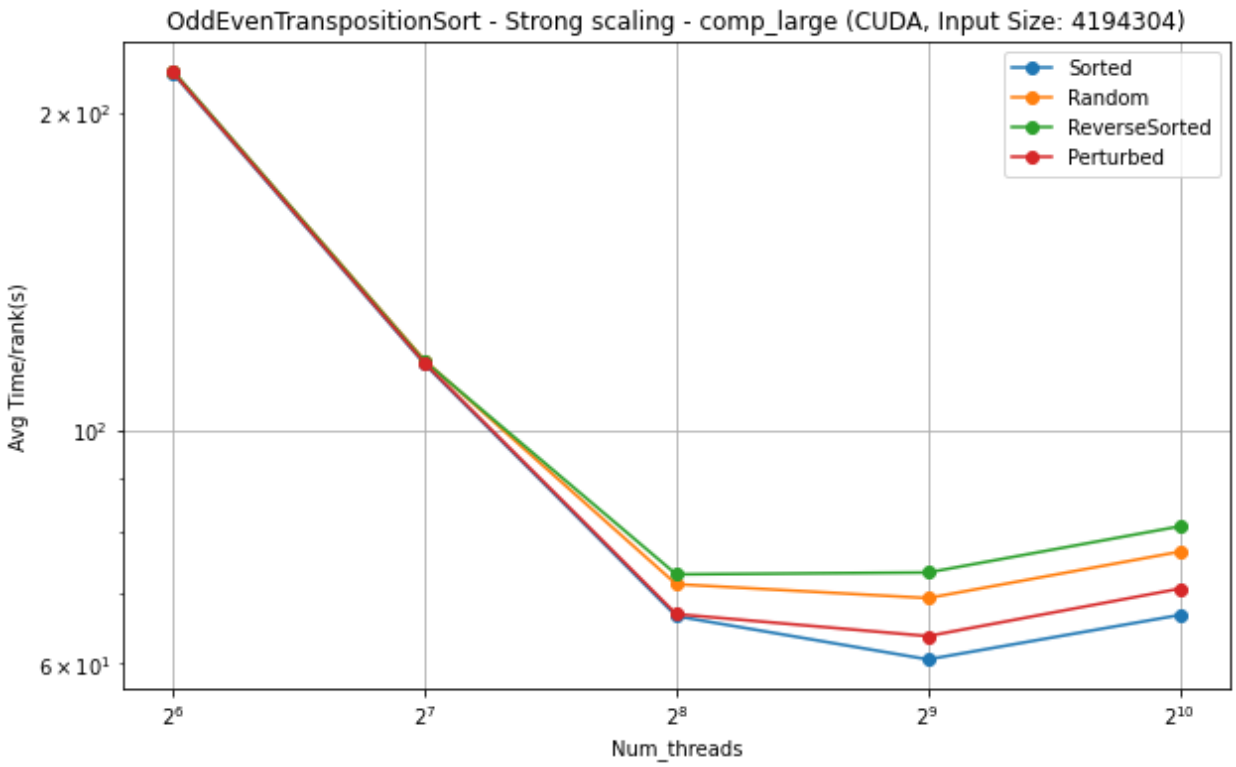

OddEvenTranspositionSort - Strong scaling - comp_large (CUDA, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_threads

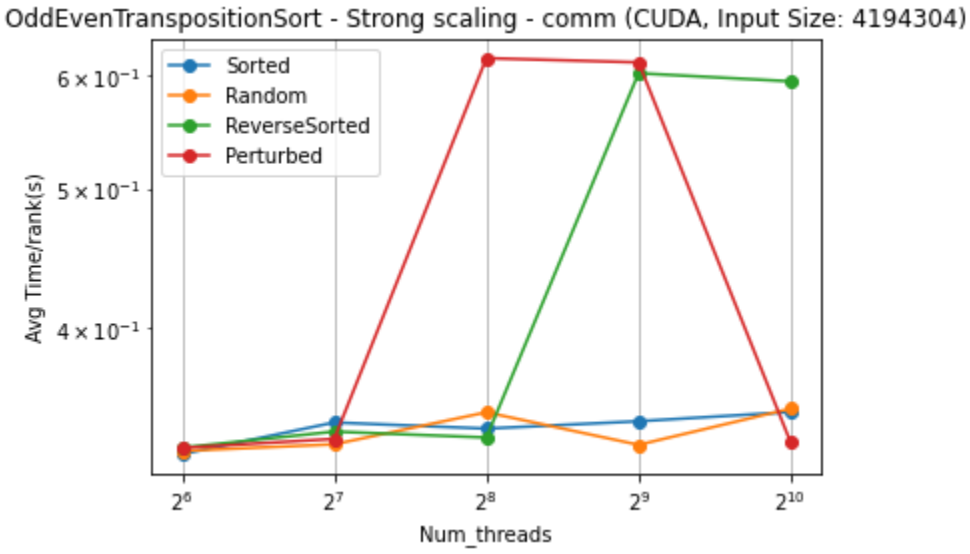
OddEvenTranspositionSort - Strong scaling - comm (CUDA, Input Size: 1048576)

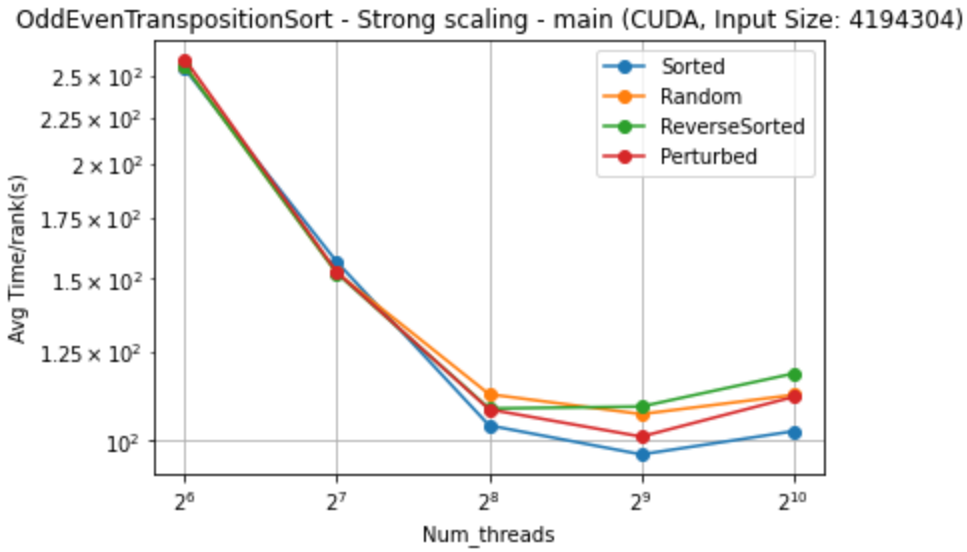


OddEvenTranspositionSort - Strong scaling - main (CUDA, Input Size: 1048576)

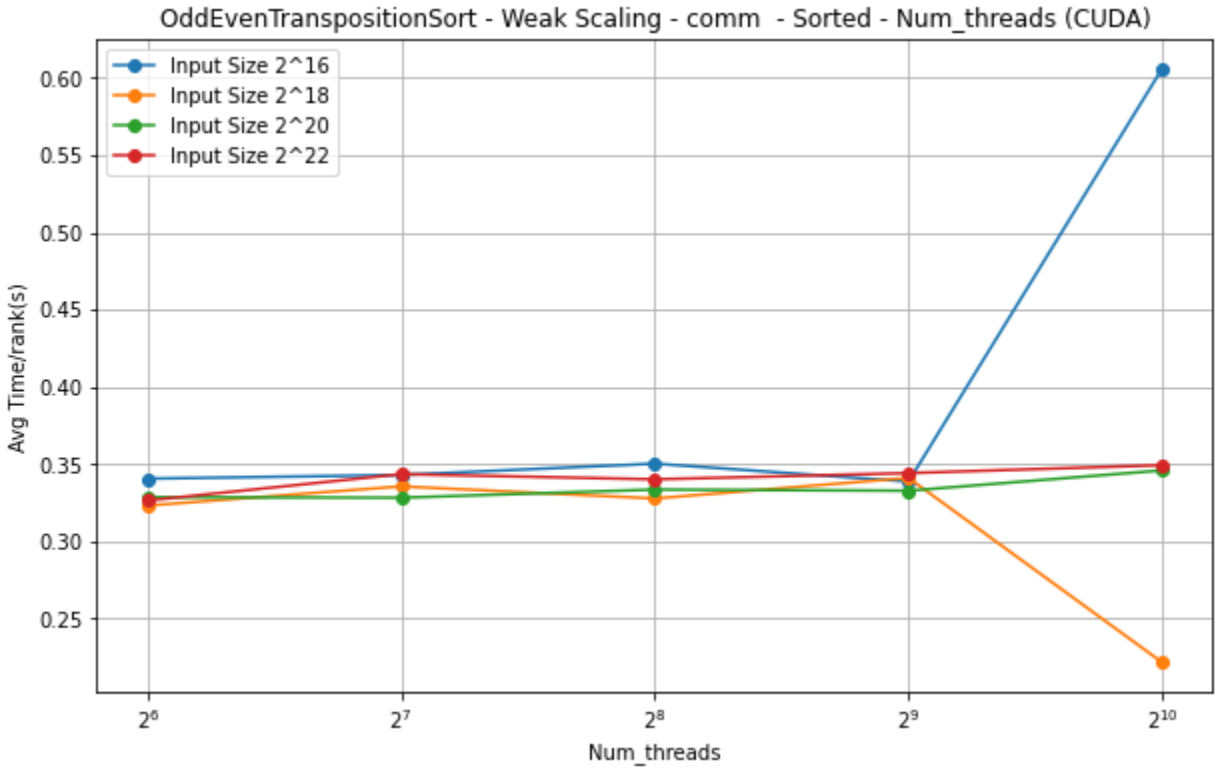


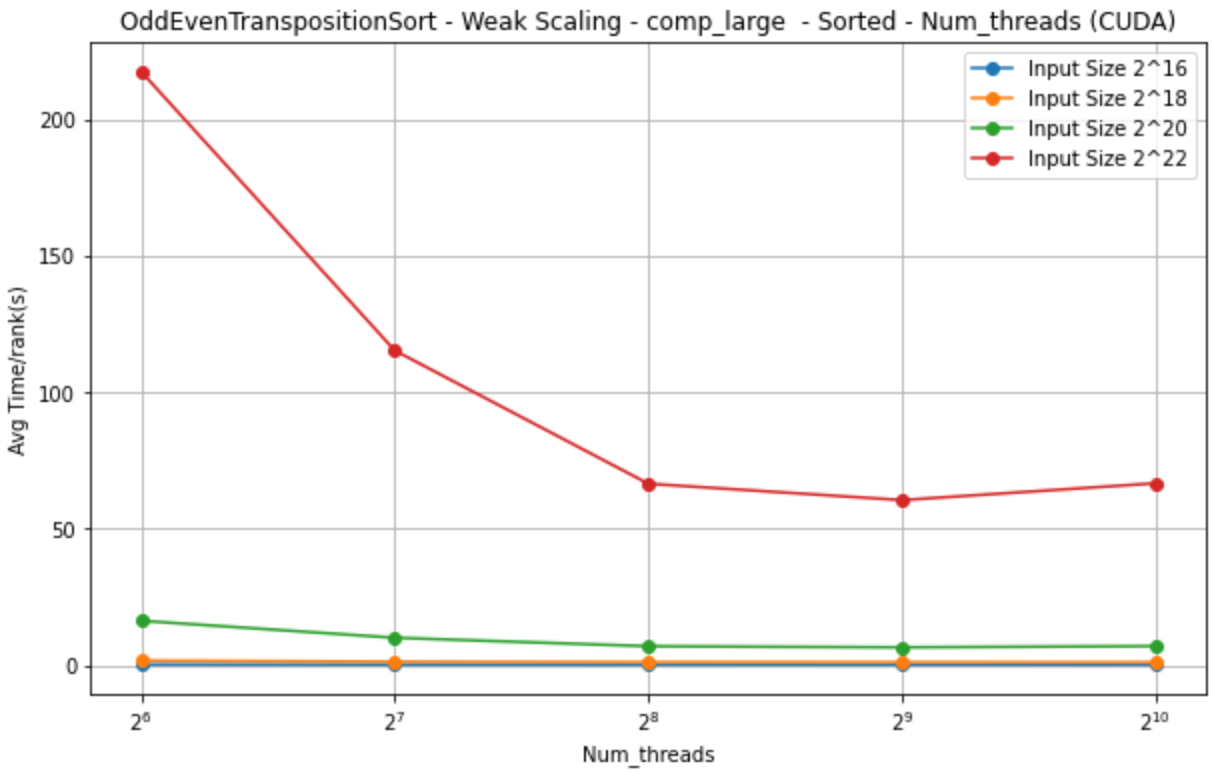

OddEvenTranspositionSort - Strong scaling - comp_large (CUDA, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
2×10^2
10^2
6×10^1
Avg Time/rank(s)
2^6
2^7
2^8
2^9
2^{10}
Num_threads

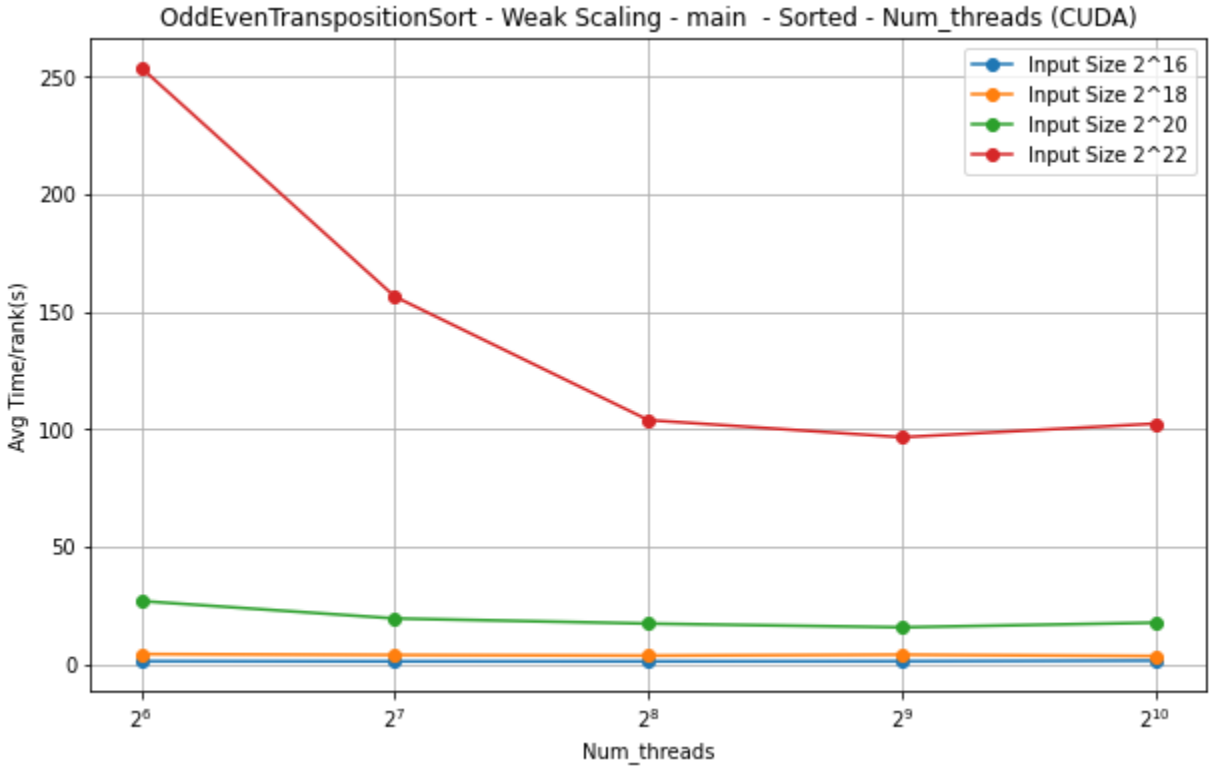

OddEvenTranspositionSort - Strong scaling - comm (CUDA, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
6×10^{-1}
5×10^{-1}
4×10^{-1}
Avg Time/rank(s)
2^6
2^7
2^8
2^9
2^{10}
Num_threads

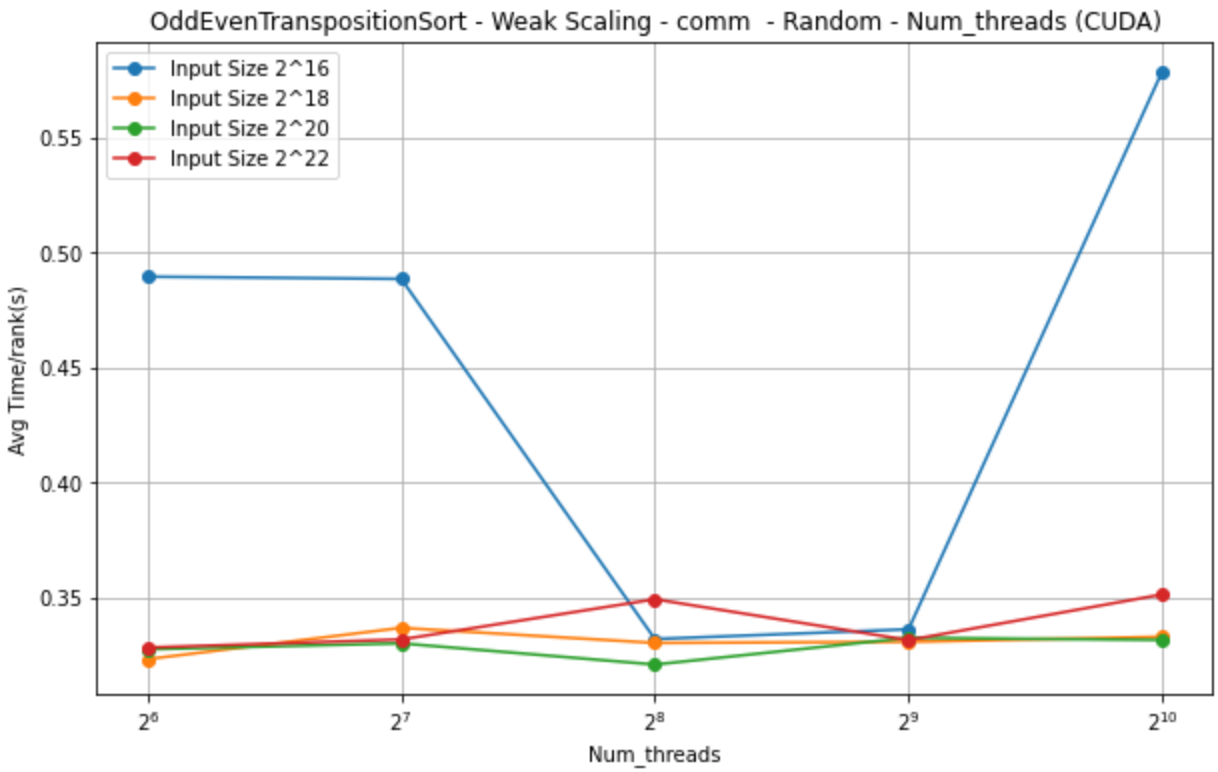


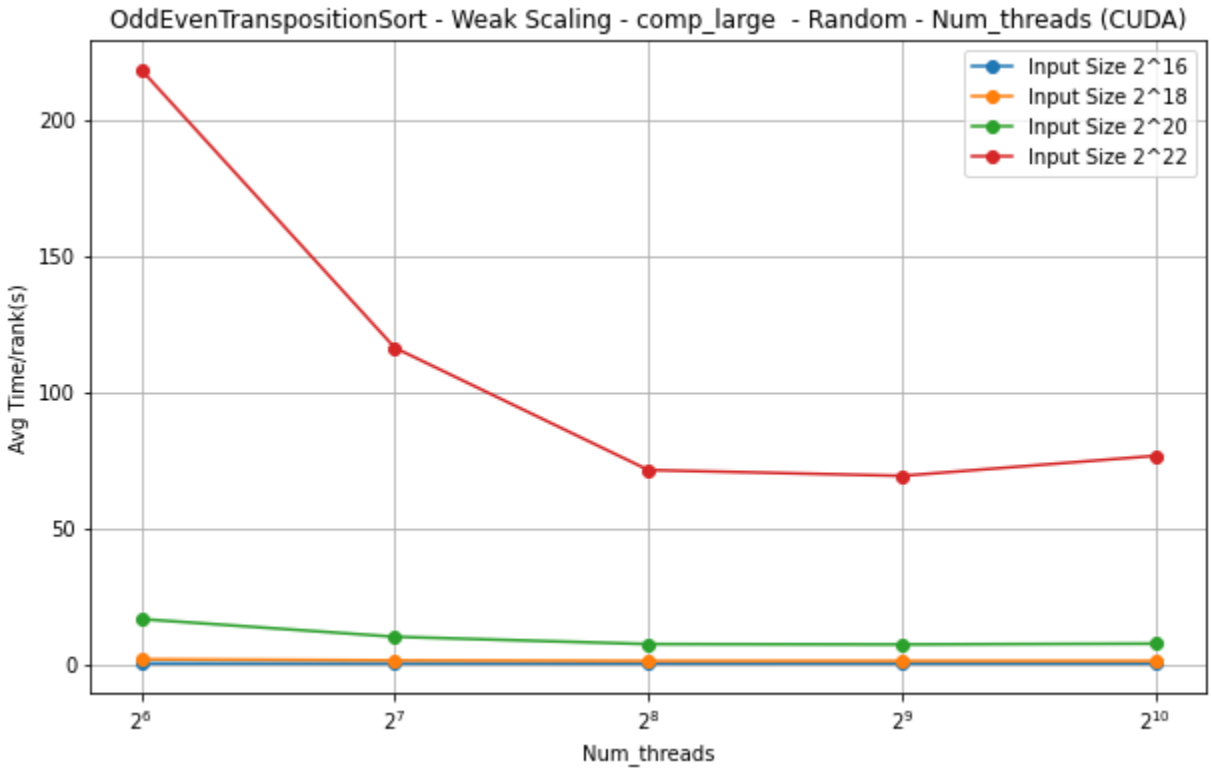
Weak Scaling:

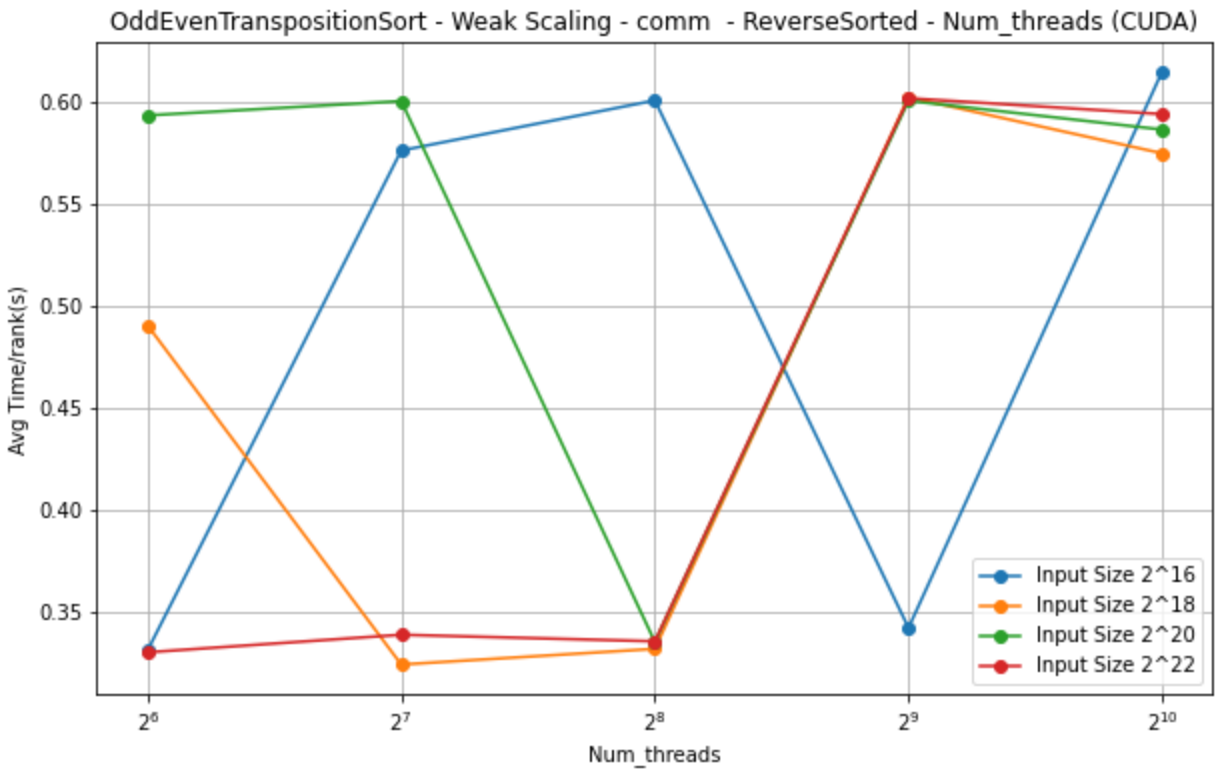


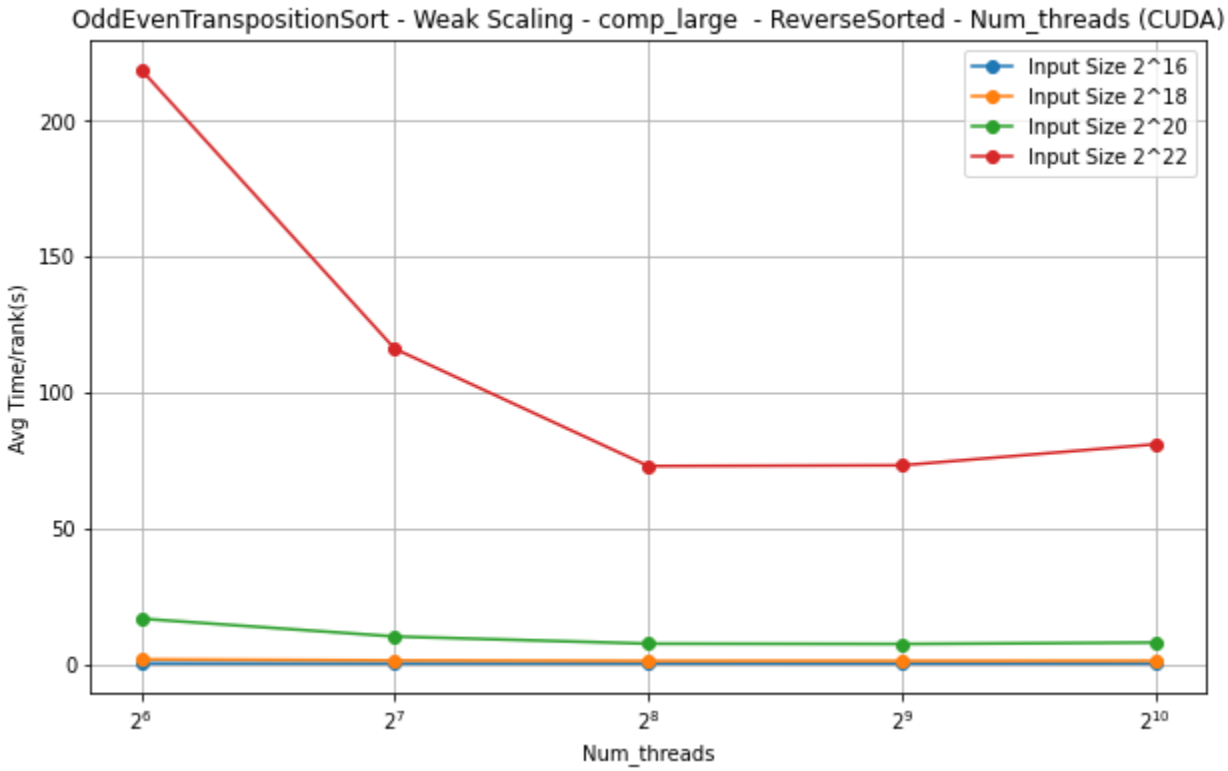

OddEvenTranspositionSort - Weak Scaling - comp_large - Sorted - Num_threads (CUDA)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
0
50
100
150
200
2^6
2^7
2^8
2^9
2^{10}
Num_threads

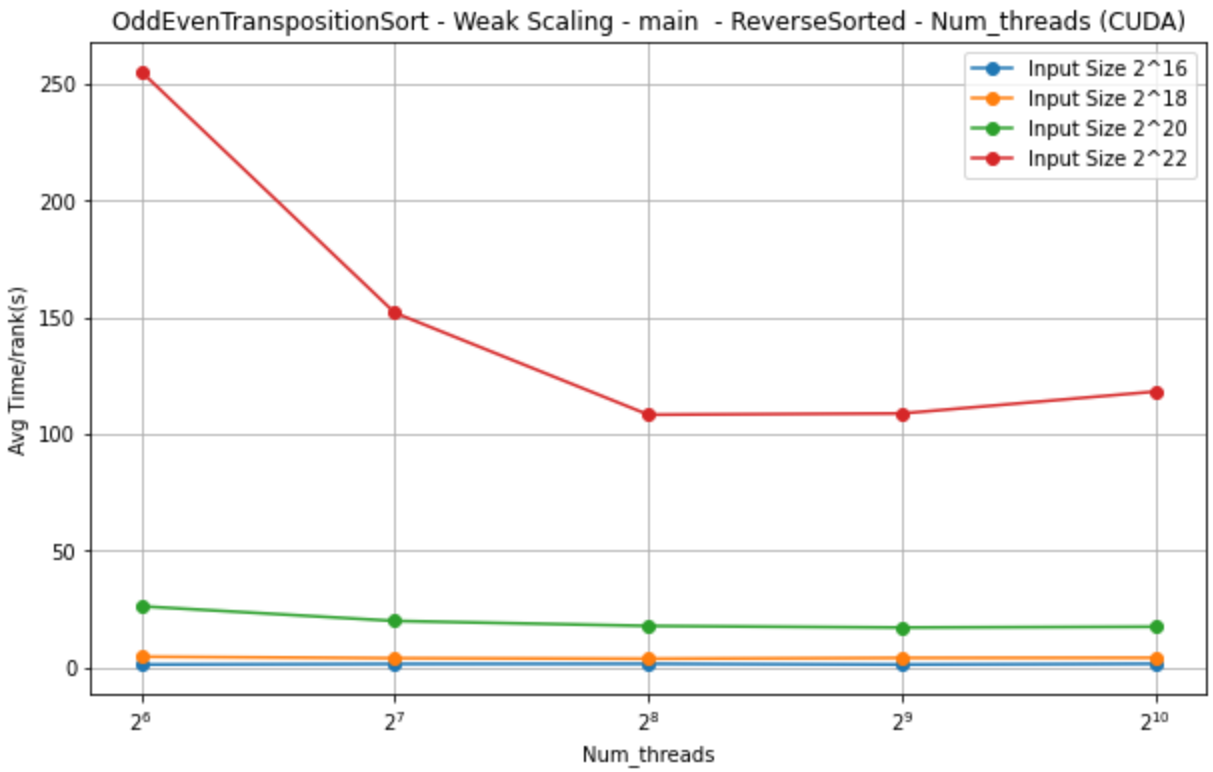

OddEvenTranspositionSort - Weak Scaling - main - Sorted - Num_threads (CUDA)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
0
50
100
150
200
250
2^6
2^7
2^8
2^9
2^{10}
Num_threads

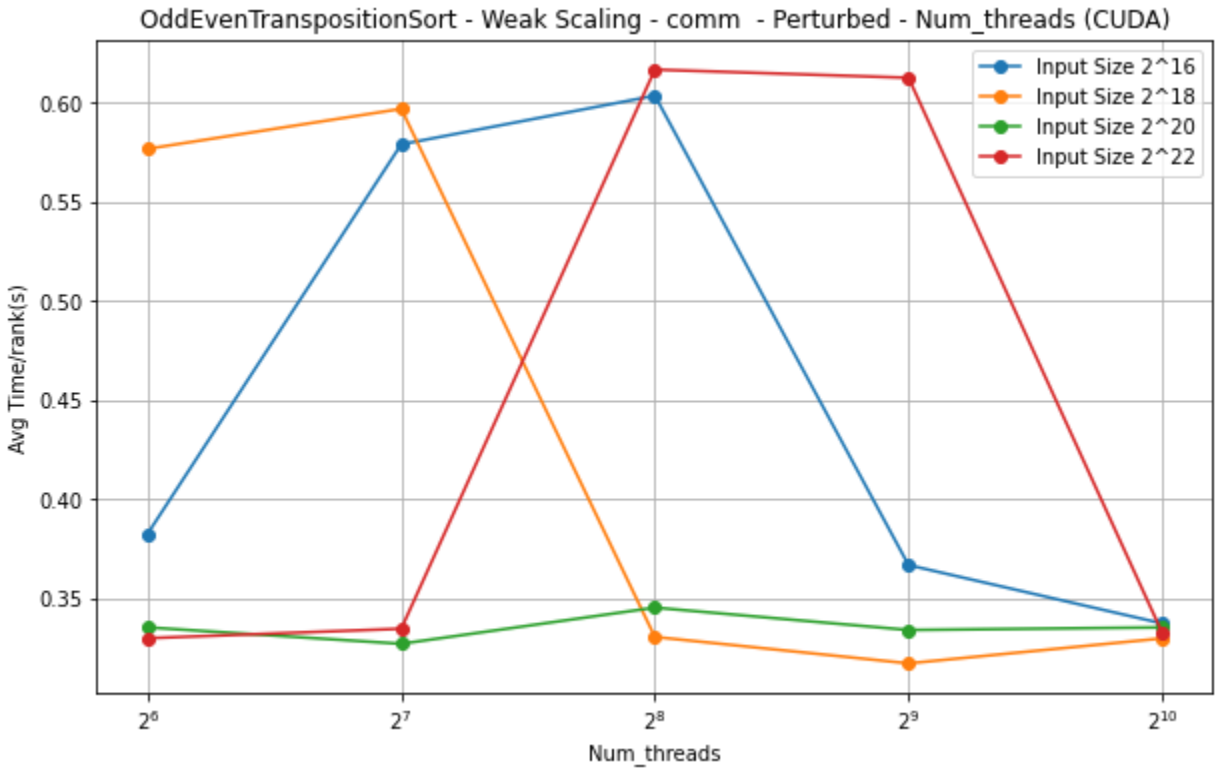

OddEvenTranspositionSort - Weak Scaling - comm - Random - Num_threads (CUDA)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
0.55
0.50
0.45
0.40
0.35
2^6
2^7
2^8
2^9
2^{10}
Num_threads

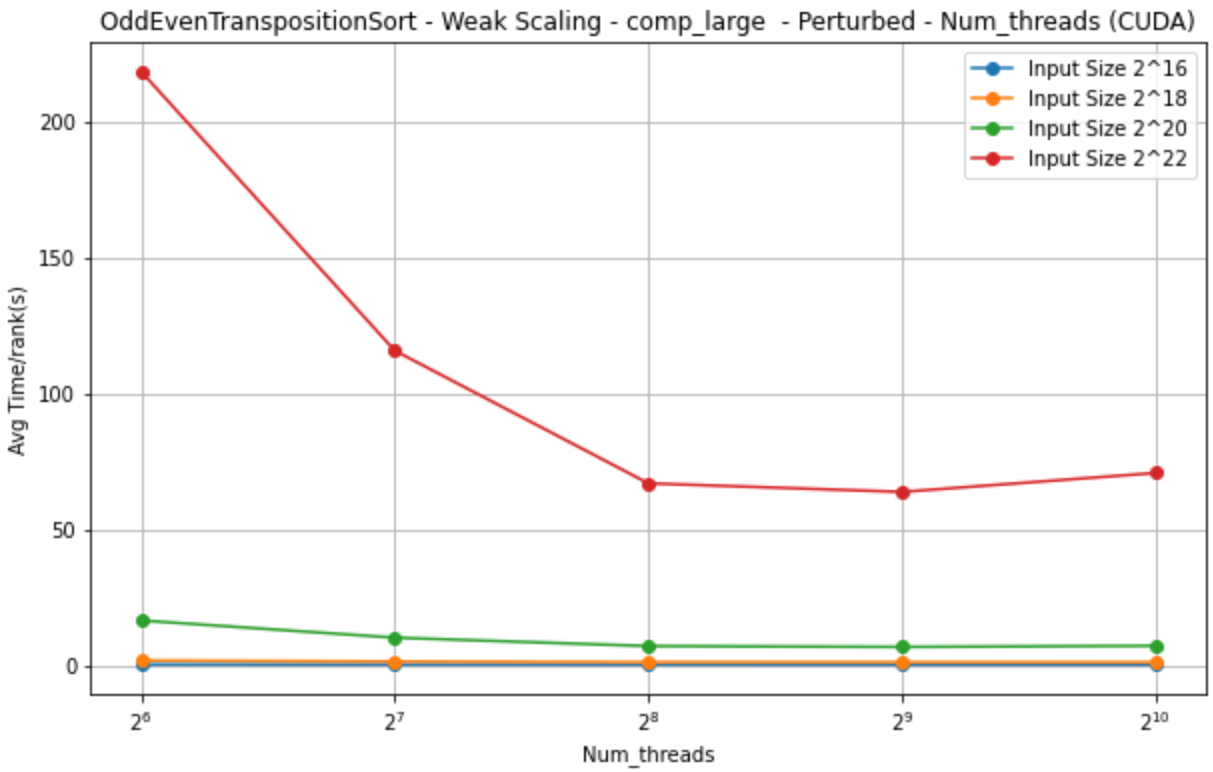

OddEvenTranspositionSort - Weak Scaling - comp_large - Random - Num_threads (CUDA)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
200
150
100
50
0
2^6
2^7
2^8
2^9
2^{10}
Num_threads

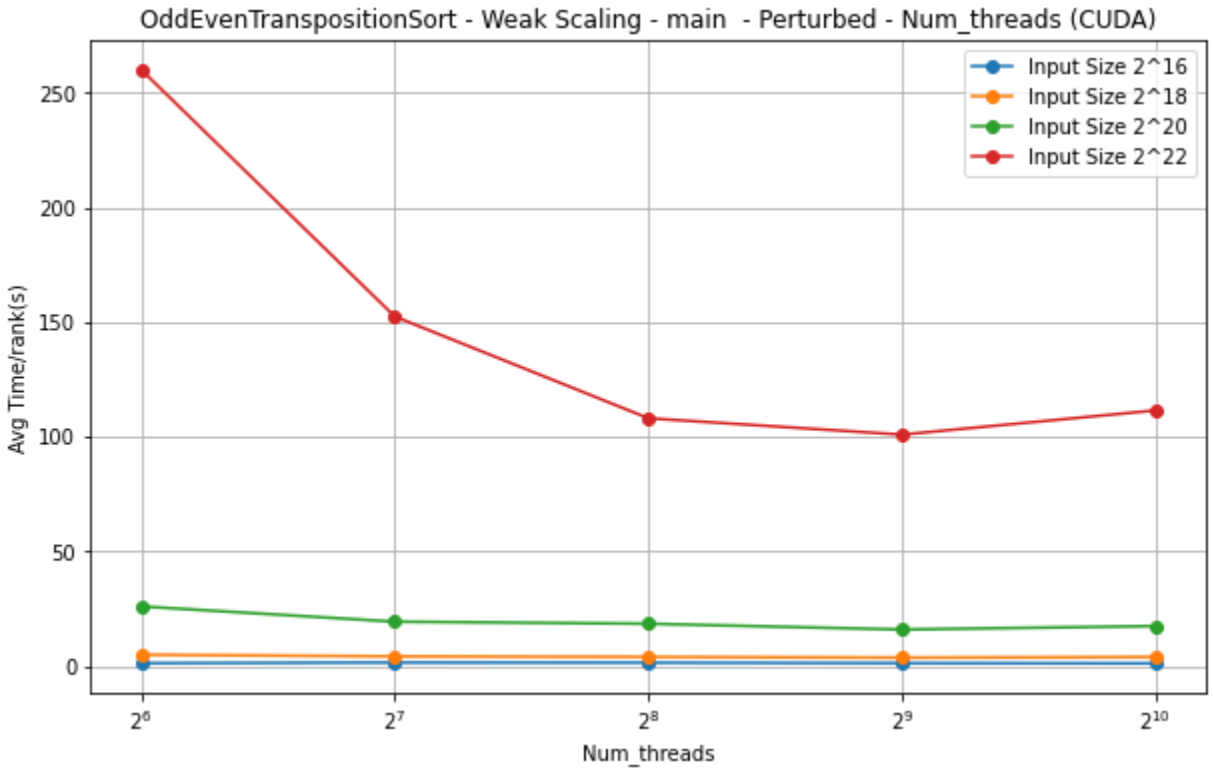

OddEvenTranspositionSort - Weak Scaling - comm - ReverseSorted - Num_threads (CUDA)
Avg Time/rank(s)
0.60
0.55
0.50
0.45
0.40
0.35
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
2^6
2^7
2^8
2^9
2^{10}
Num_threads


OddEvenTranspositionSort - Weak Scaling - comp_large - ReverseSorted - Num_threads (CUDA)
Avg Time/rank(s)
200
150
100
50
0
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
2^6
2^7
2^8
2^9
2^{10}
Num_threads

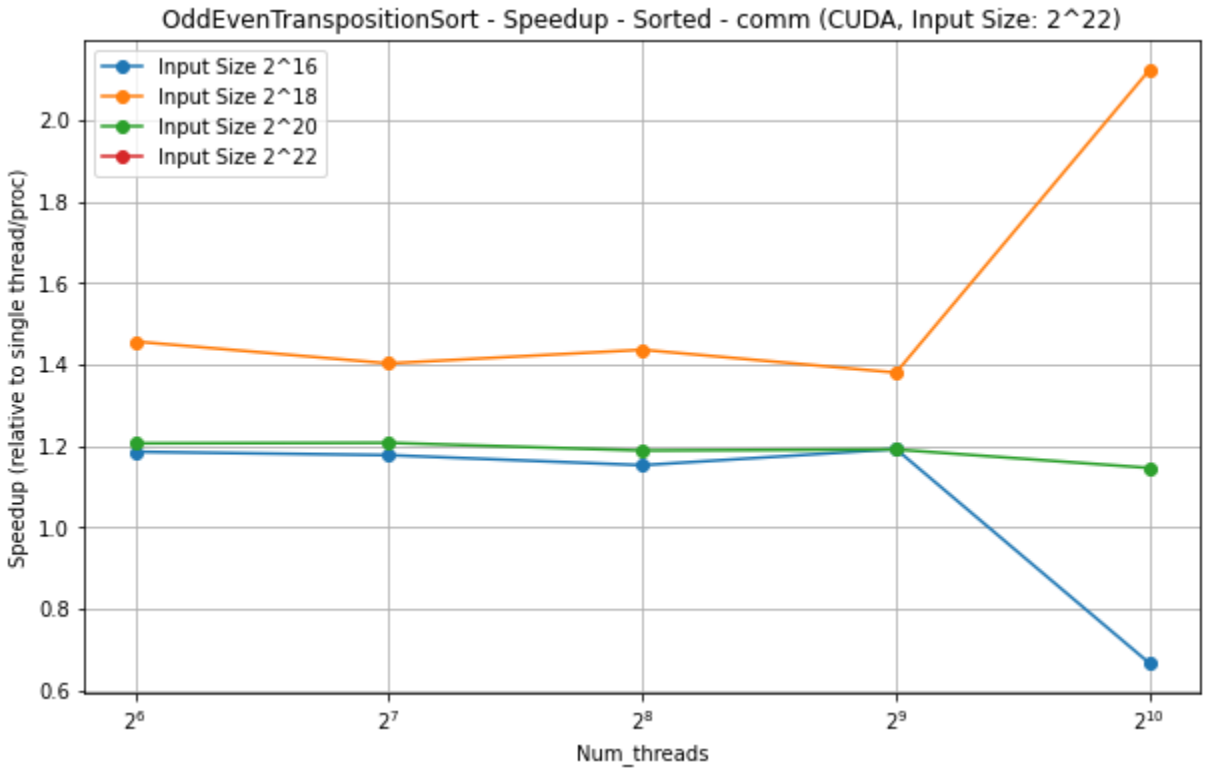

OddEvenTranspositionSort - Weak Scaling - main - ReverseSorted - Num_threads (CUDA)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
Num_threads
0
50
100
150
200
250
2^6
2^7
2^8
2^9
2^{10}

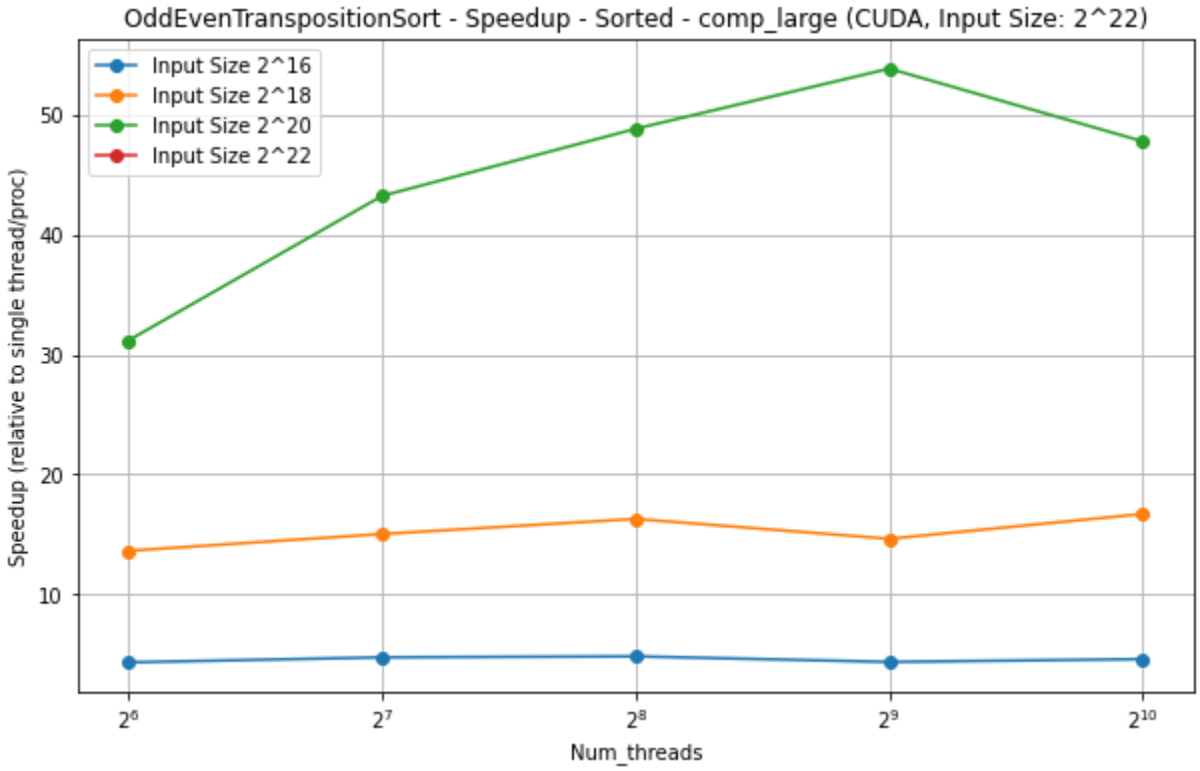

OddEvenTranspositionSort - Weak Scaling - comm - Perturbed - Num_threads (CUDA)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
Num_threads
0.35
0.40
0.45
0.50
0.55
0.60
2^6
2^7
2^8
2^9
2^{10}

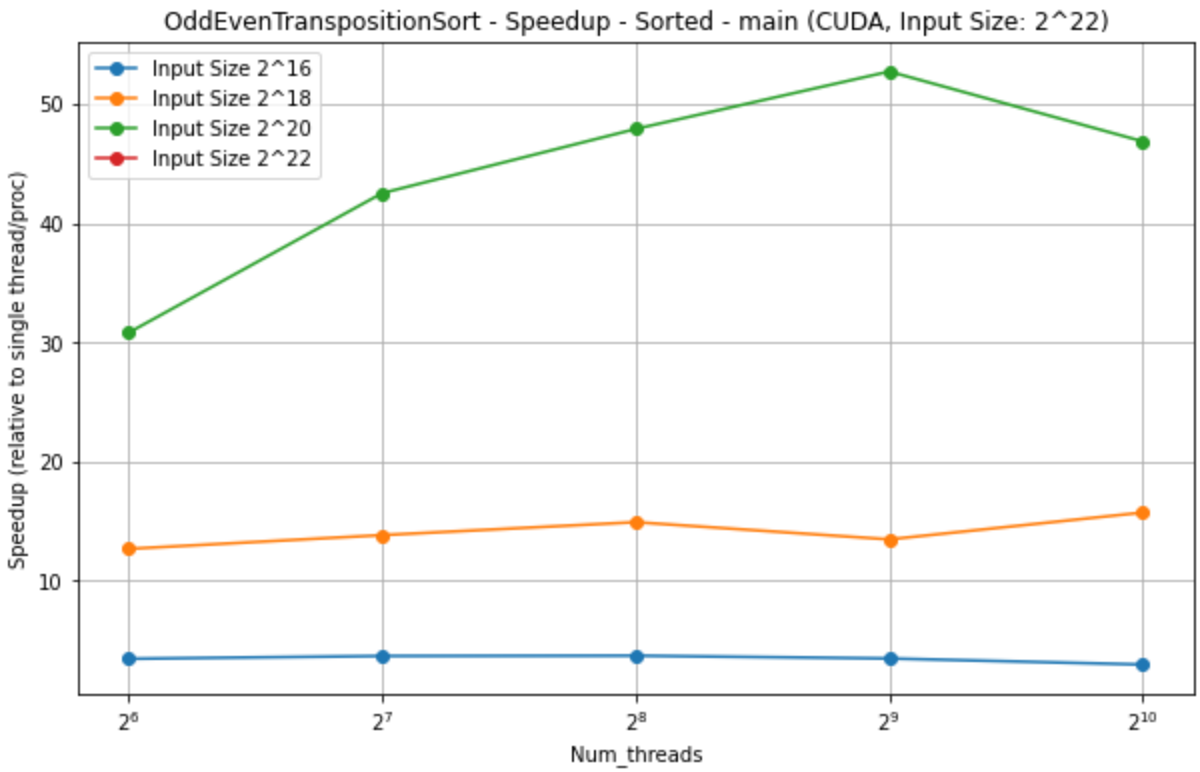

OddEvenTranspositionSort - Weak Scaling - comp_large - Perturbed - Num_threads (CUDA)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
Num_threads
0
50
100
150
200
2^6
2^7
2^8
2^9
2^{10}

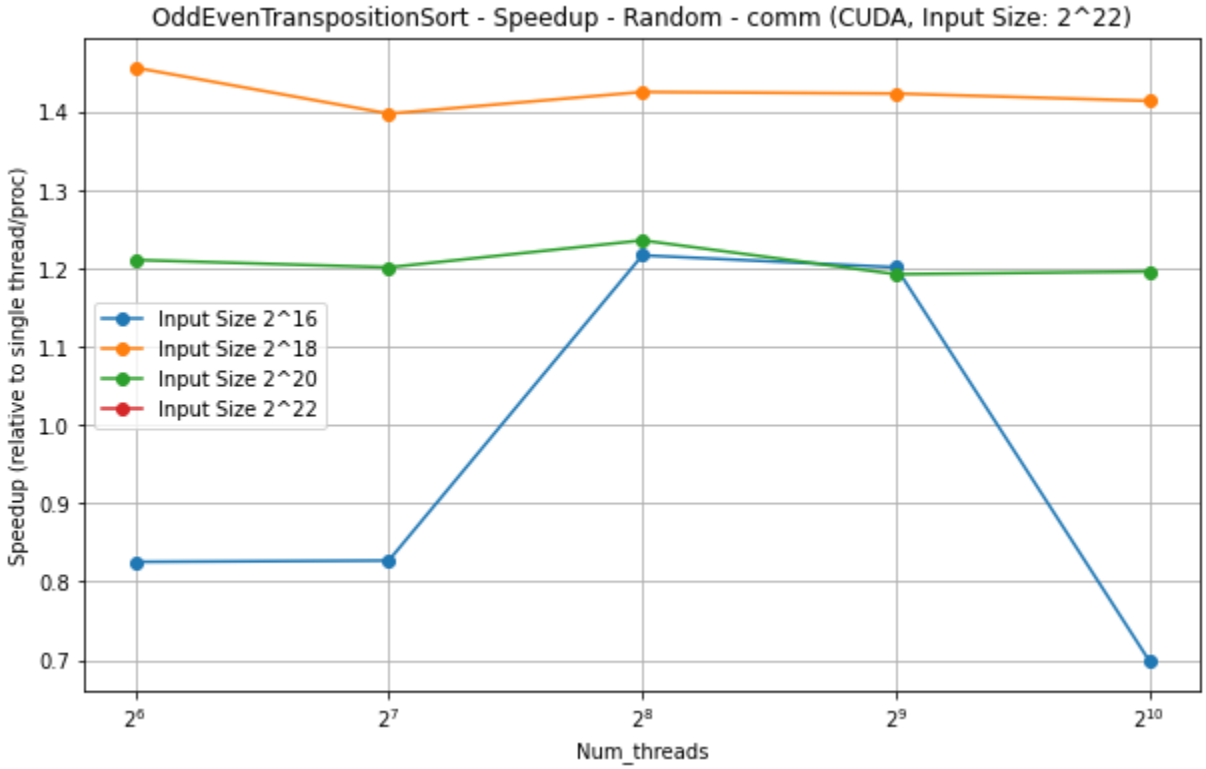

OddEvenTranspositionSort - Weak Scaling - main - Perturbed - Num_threads (CUDA)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Avg Time/rank(s)
Num_threads
0
50
100
150
200
250
2^6
2^7
2^8
2^9
2^{10}

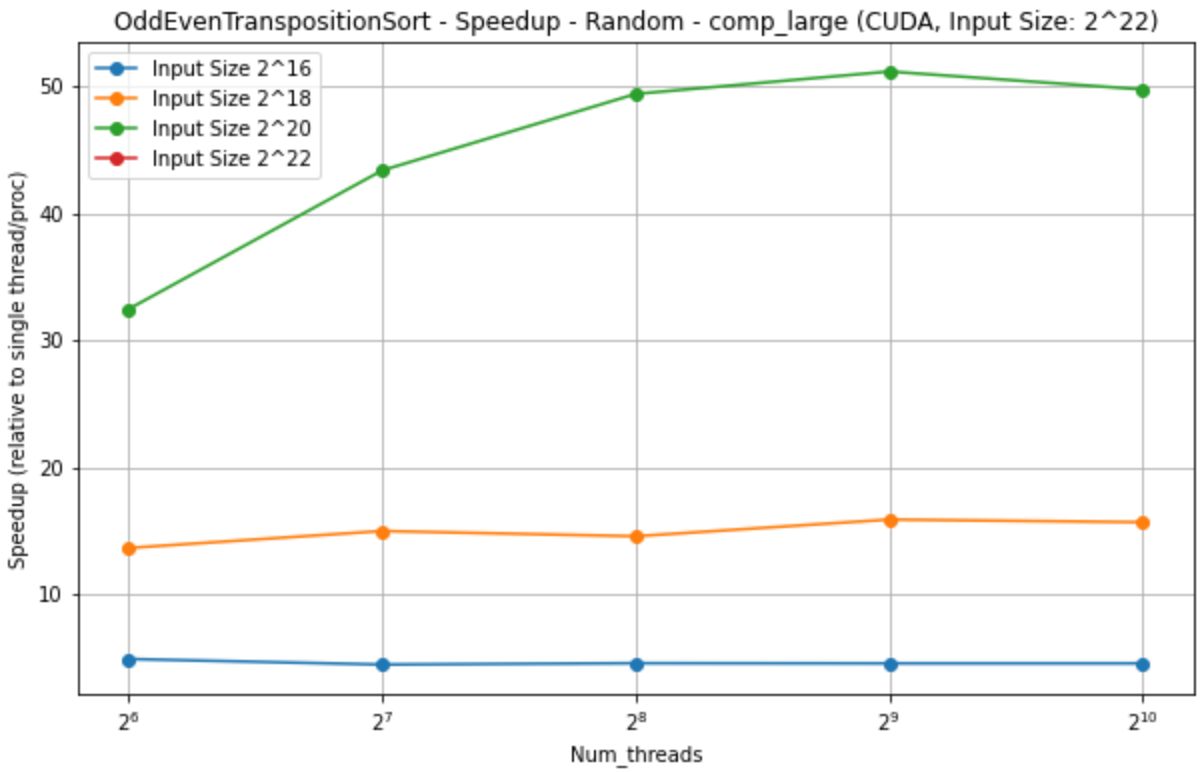
Speedup:

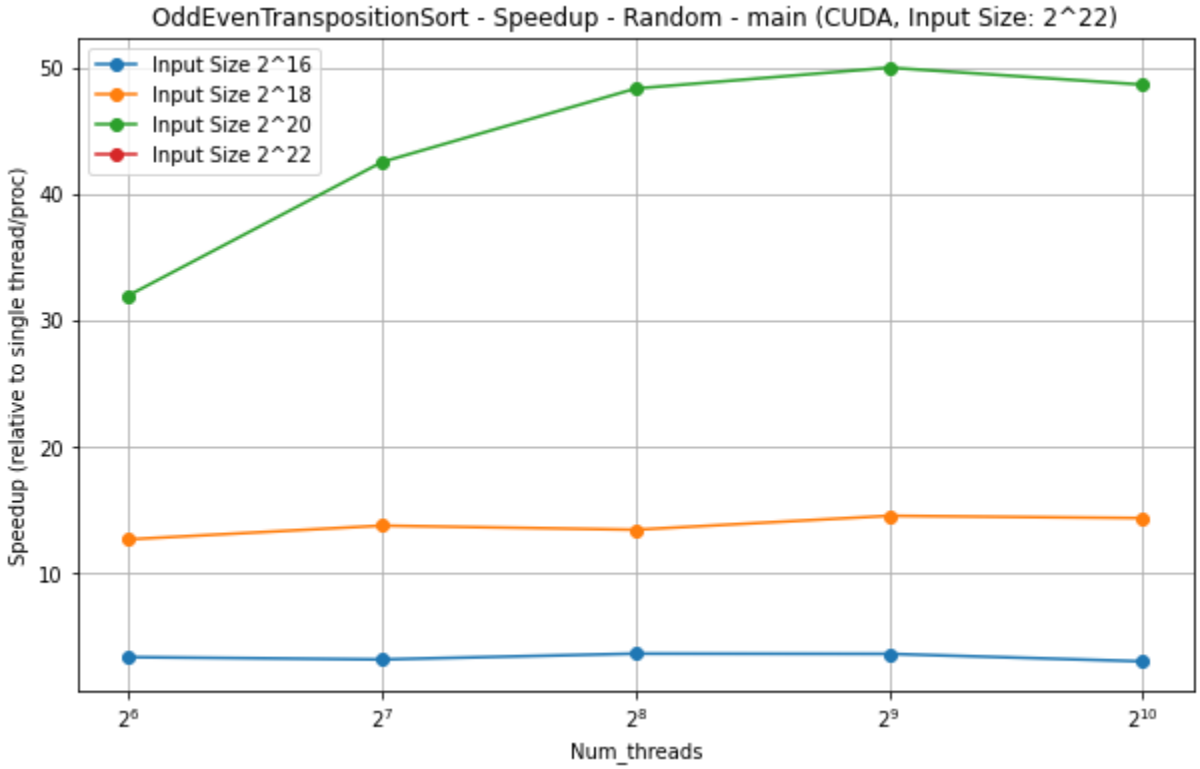

OddEvenTranspositionSort - Speedup - Sorted - comm (CUDA, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
2^6
2^7
2^8
2^9
2^{10}
Num_threads

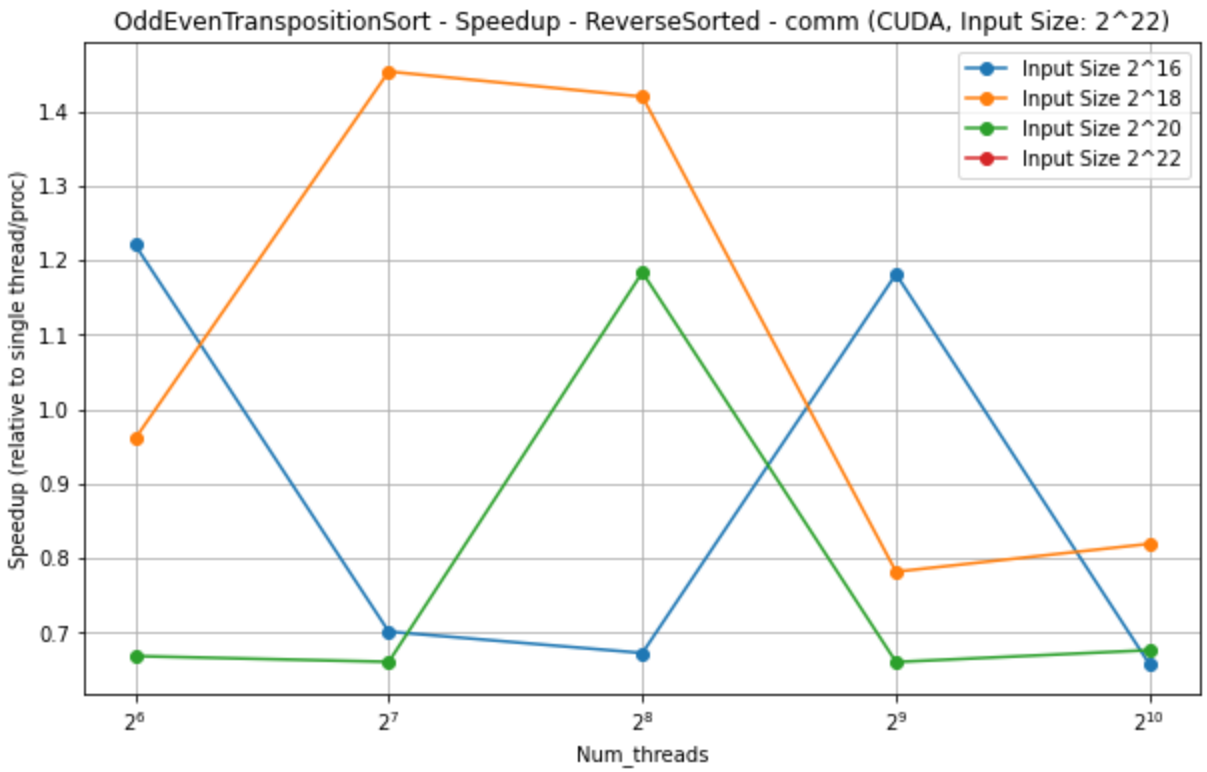

OddEvenTranspositionSort - Speedup - Sorted - comp_large (CUDA, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
50
40
30
20
10
2^6
2^7
2^8
2^9
2^{10}
Num_threads

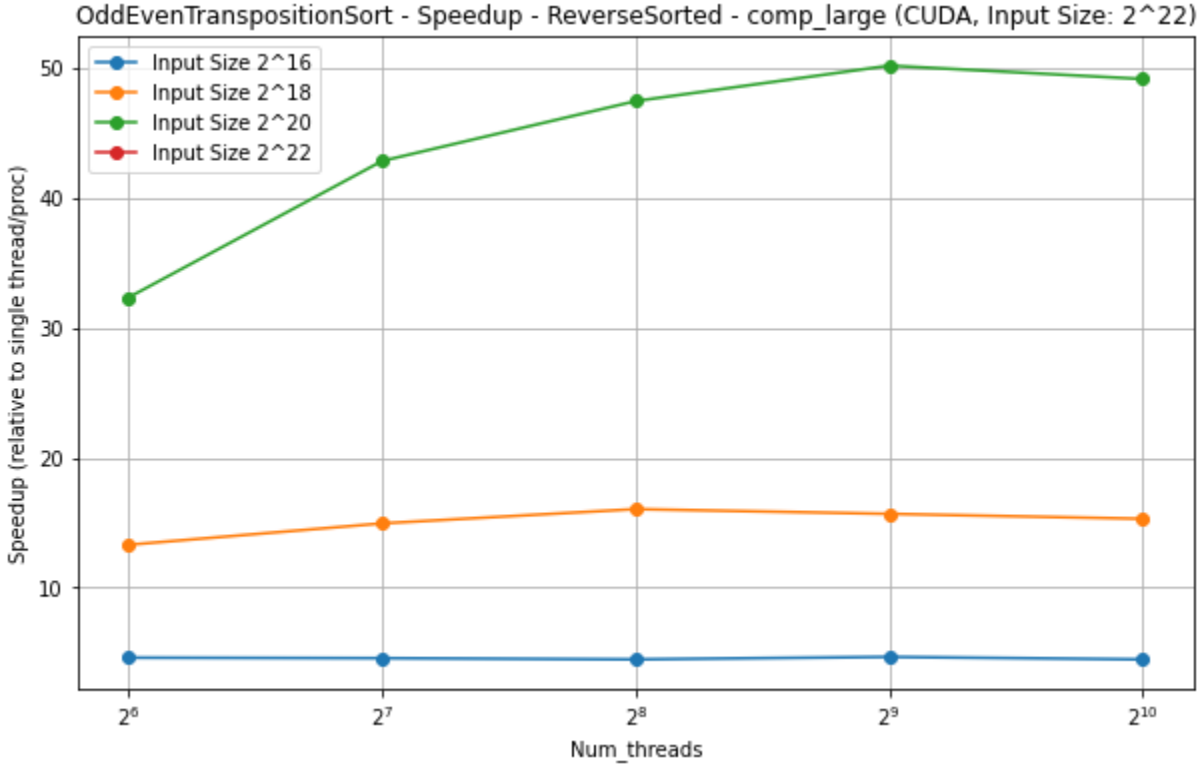

OddEvenTranspositionSort - Speedup - Sorted - main (CUDA, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
50
40
30
20
10
2^6
2^7
2^8
2^9
2^{10}
Num_threads

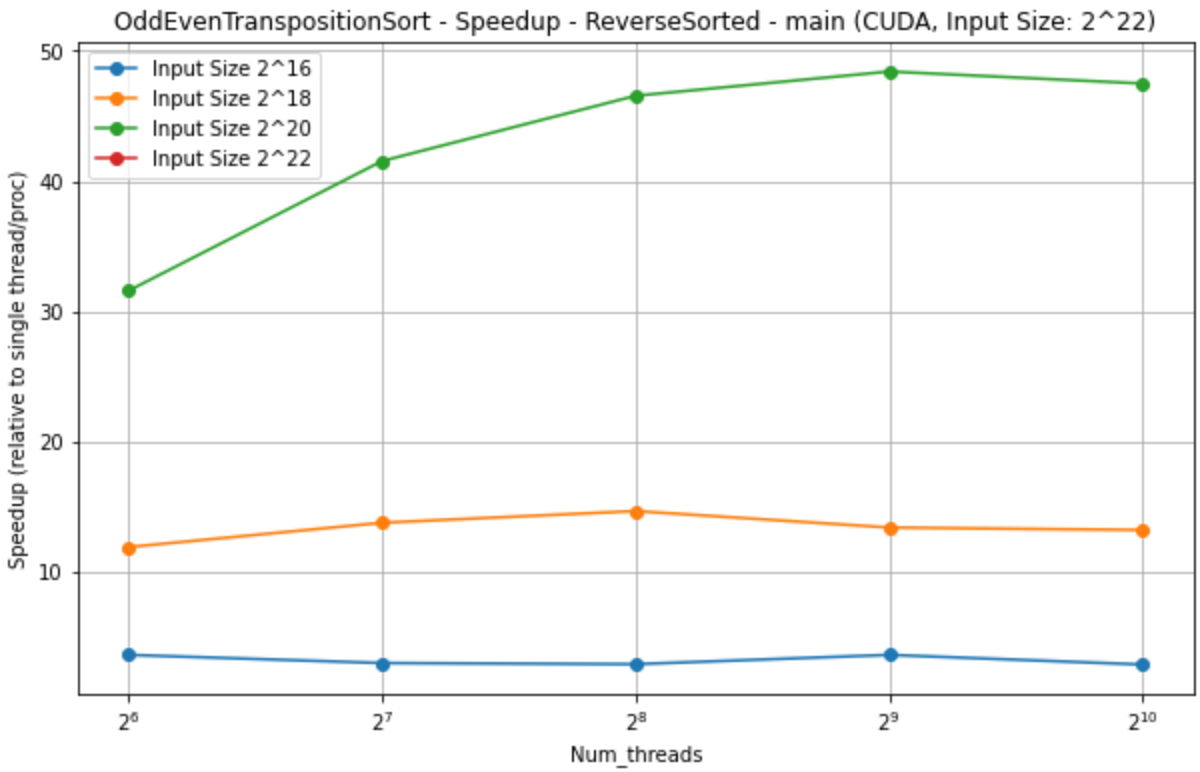

OddEvenTranspositionSort - Speedup - Random - comm (CUDA, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
2^6
2^7
2^8
2^9
2^{10}
Num_threads

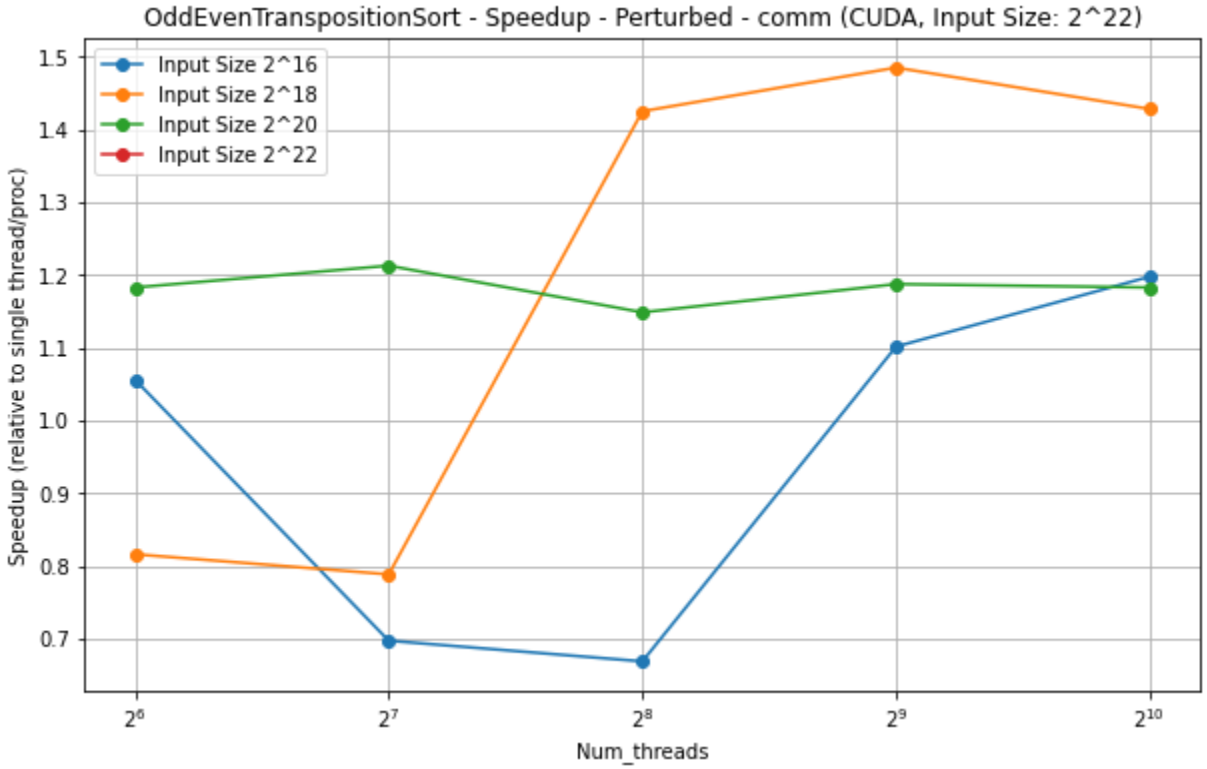

OddEvenTranspositionSort - Speedup - Random - comp_large (CUDA, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
50
40
30
20
10
2^6
2^7
2^8
2^9
2^{10}
Num_threads

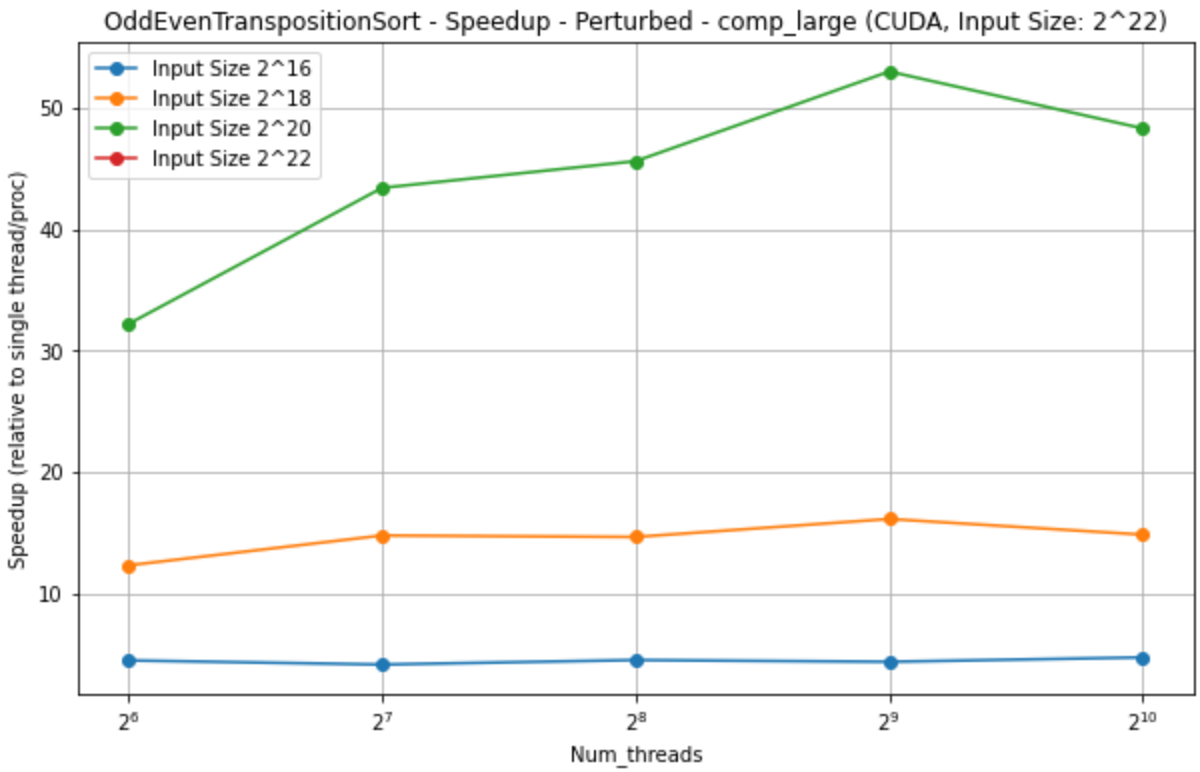

OddEvenTranspositionSort - Speedup - Random - main (CUDA, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
50
40
30
20
10
2^6
2^7
2^8
2^9
2^{10}
Num_threads

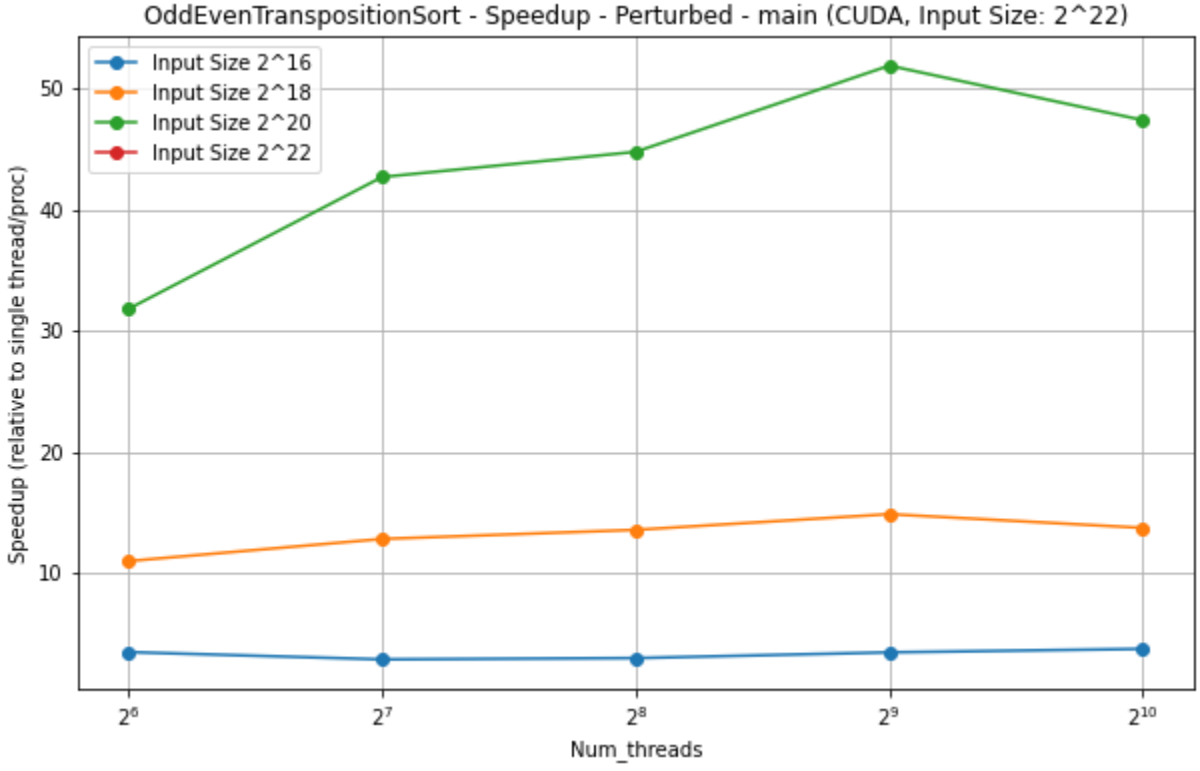

OddEvenTranspositionSort - Speedup - ReverseSorted - comm (CUDA, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
2^6
2^7
2^8
2^9
2^{10}
Num_threads


OddEvenTranspositionSort - Speedup - ReverseSorted - comp_large (CUDA, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
50
40
30
20
10
2^6
2^7
2^8
2^9
2^{10}
Num_threads


OddEvenTranspositionSort - Speedup - ReverseSorted - main (CUDA, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
50
40
30
20
10
2^6
2^7
2^8
2^9
2^{10}
Num_threads


OddEvenTranspositionSort - Speedup - Perturbed - comm (CUDA, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
2^6
2^7
2^8
2^9
2^{10}
Num_threads


OddEvenTranspositionSort - Speedup - Perturbed - comp_large (CUDA, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
50
40
30
20
10
2^6
2^7
2^8
2^9
2^{10}
Num_threads


OddEvenTranspositionSort - Speedup - Perturbed - main (CUDA, Input Size: 2^22)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Speedup (relative to single thread/proc)
50
40
30
20
10
2^6
2^7
2^8
2^9
2^{10}
Num_threads

Merge Sort:

Considerations:

Merge sort is a sorting algorithm that is parallelized by the array being divided and sorted among multiple processes and threads to allow for greater efficiency. After the divided arrays are sorted concurrently they are synchronized together. My CUDA implementation has a bug that essentially makes it no better than a sequential version of merge sort which is why you will see that it does not scale very well. Aside from that I was able to generate all the required files as needed.

Performance Analysis:

I will be analyzing the plots below in the following section:

Strong scaling plot analysis:

For the MPI implementation, I found a few interesting observations. Firstly that for the computation regions sorted is nearly always faster than perturbed, which is nearly always faster than Reverse Sorted, which is nearly always faster than Random Input Types. Secondly that communication only scaled for larger inputs, but otherwise it did not scale at all. In this scenario I would say that communication, which consists of multiple MPI_Gather and Scatter calls, was the bottleneck.

For the CUDA implementations I found the same general trend for the different input types but only on large input sizes. Generally as the number of threads increased both the communication and computation regions increased. This is most likely due to the fact that my algorithm did not really parralelize at all.

Weak scaling plot analysis:
For the MPI implementation the plots show that larger input sizes tend to scale better for both comp and comm, for all input types. Comp usually also scales all the time for all input types. The input types in this case didn't seem to make much difference.

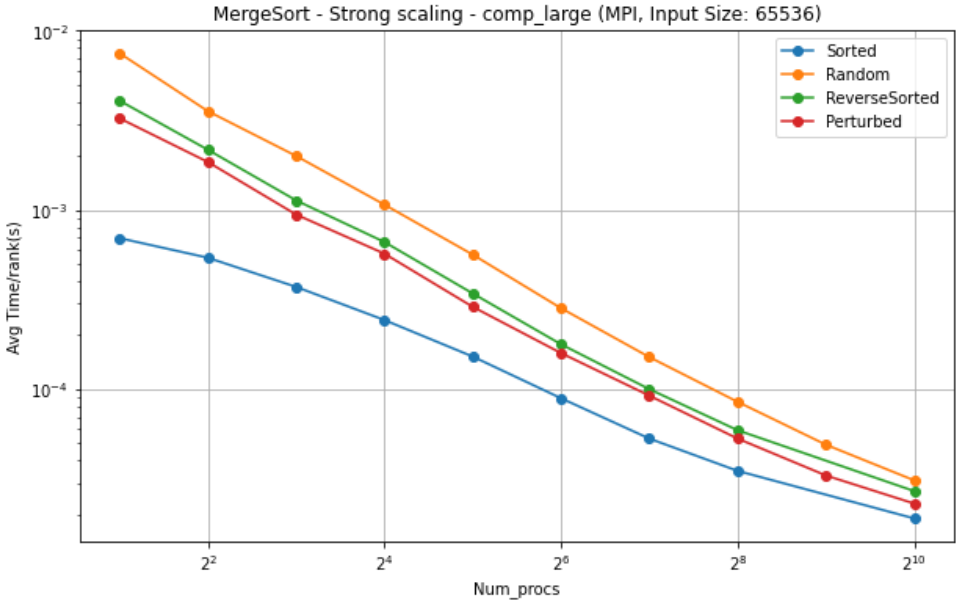
For the CUDA implementation the weak scaling plots didn't reveal much more than the strong scaling plots already have.

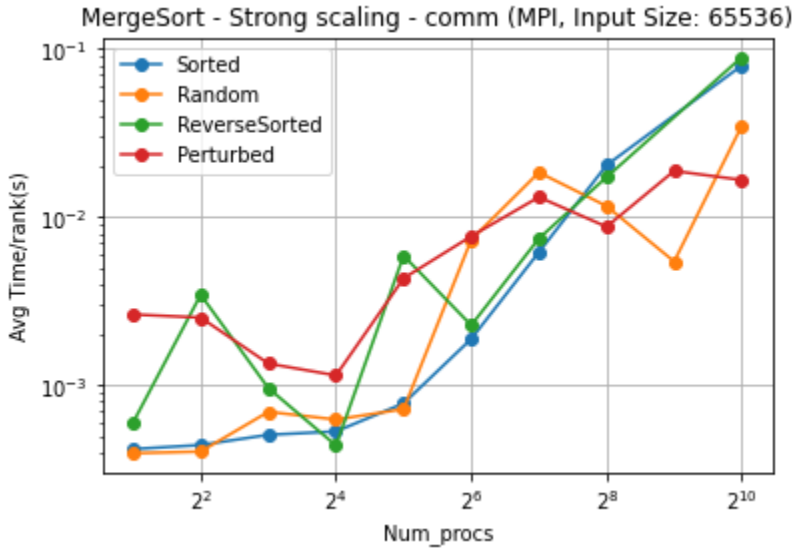
Speedup plot analysis:
For the MPI implementation my main takeaway is that the speedup for computation continuously increases with increasing processor count, while speedup for comm peaks at around 2^5.
For CUDA my main takeaway is that there is no real trend to be observed as I did not see any based on observing the plots.

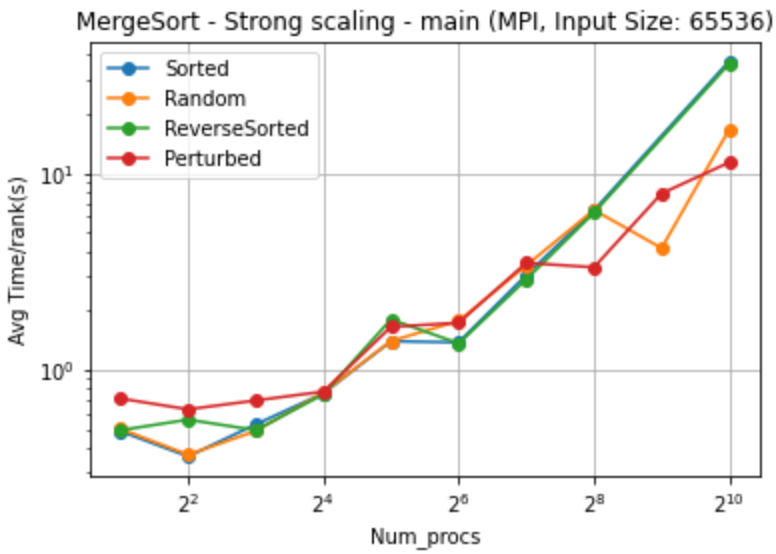
Merge Sort Plots:

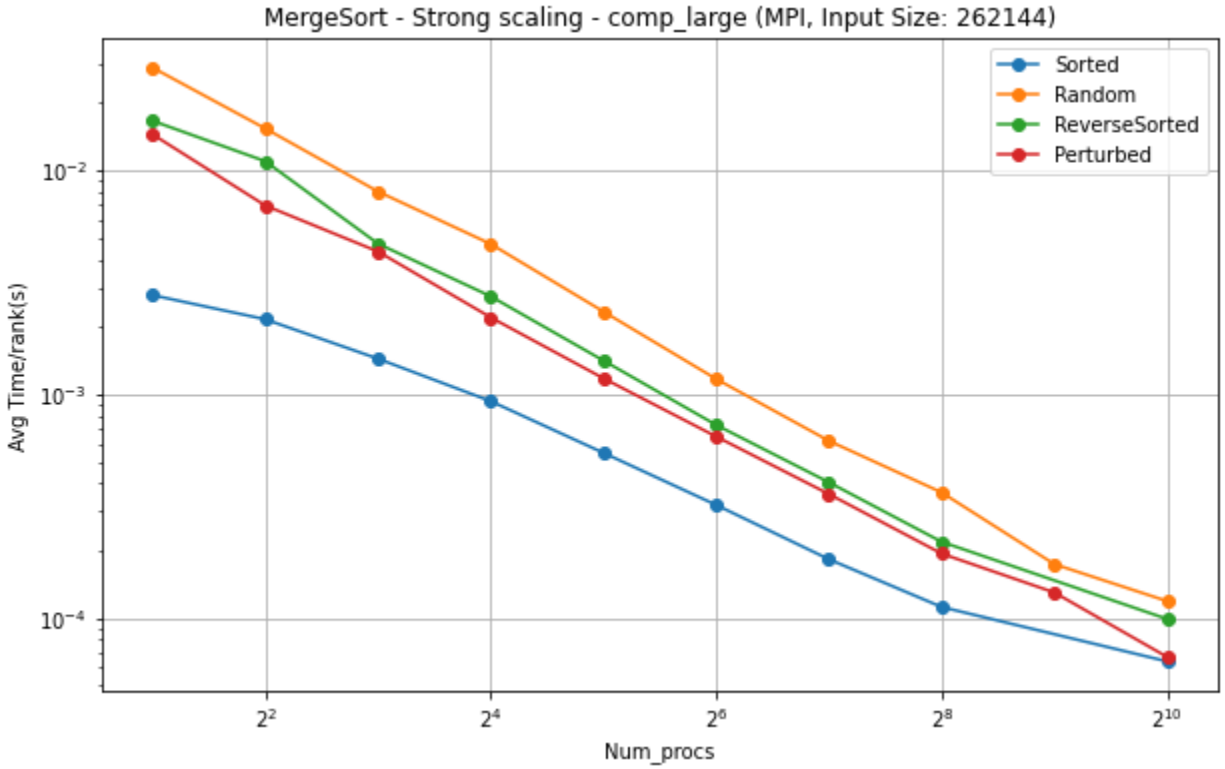
MPI:

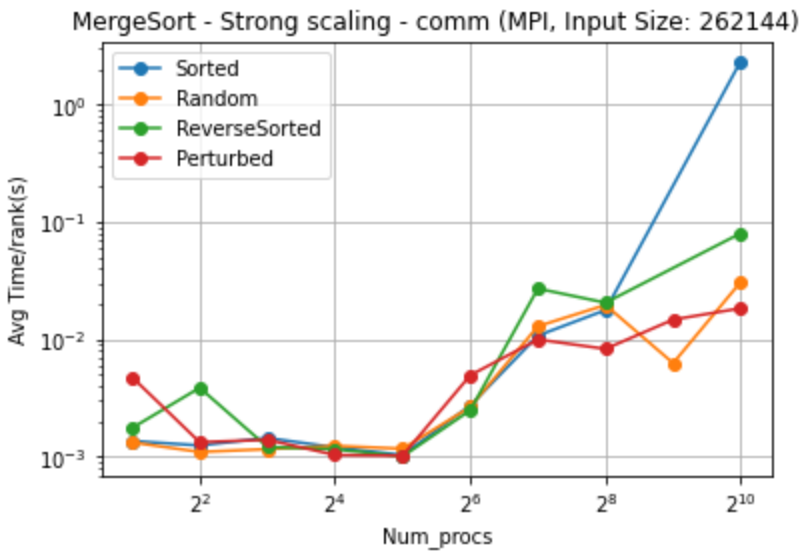
Strong Scaling:

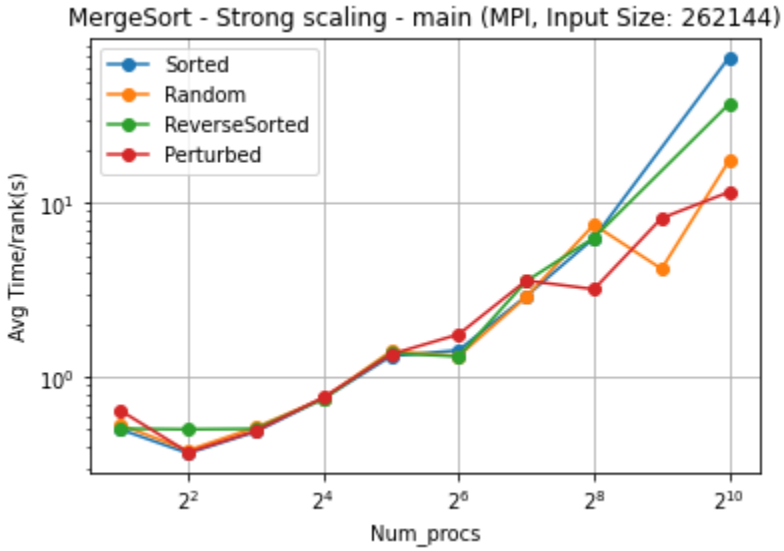


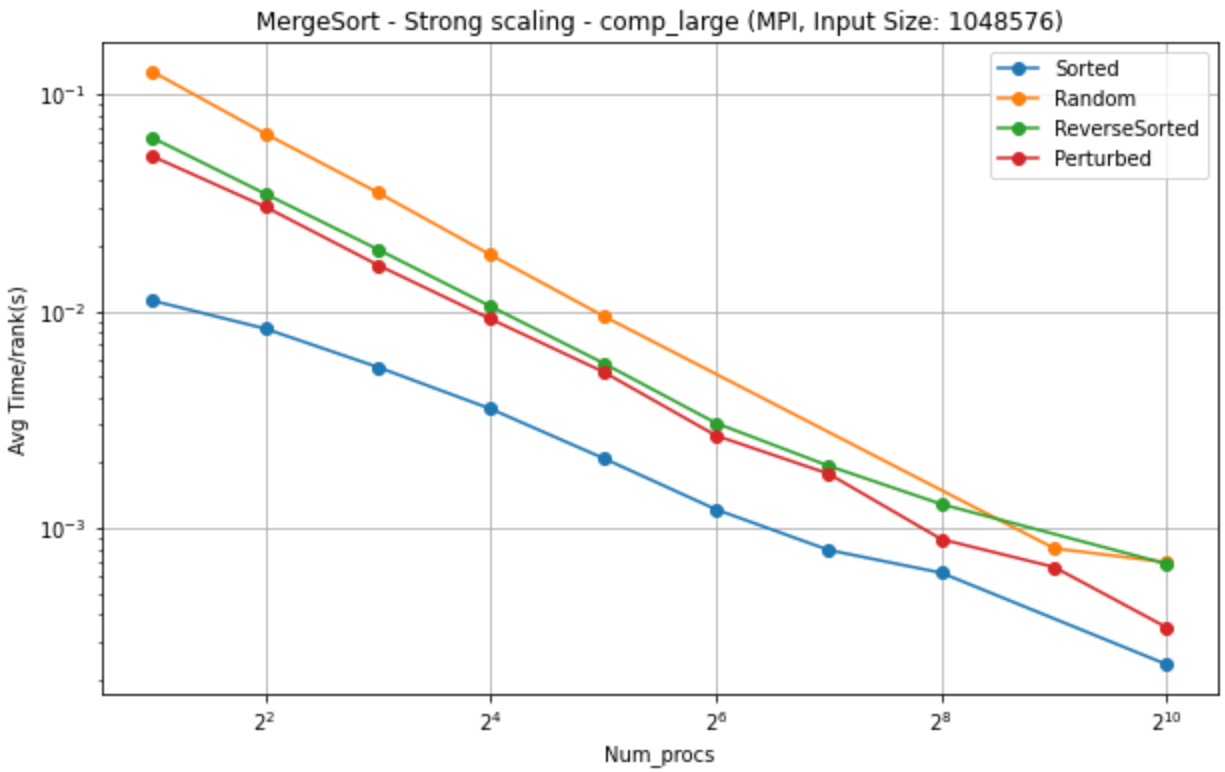


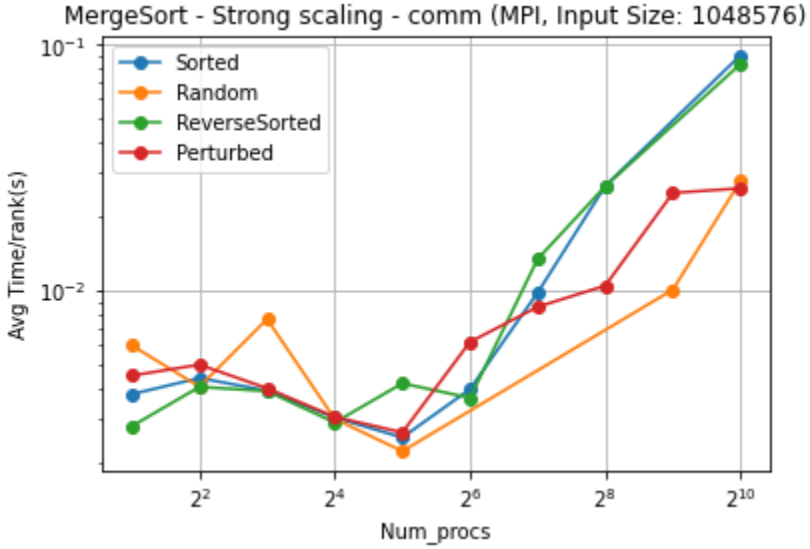

MergeSort - Strong scaling - main (MPI, Input Size: 65536)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
10^0
2^2
2^4
2^6
2^8
2^{10}
Num_procs

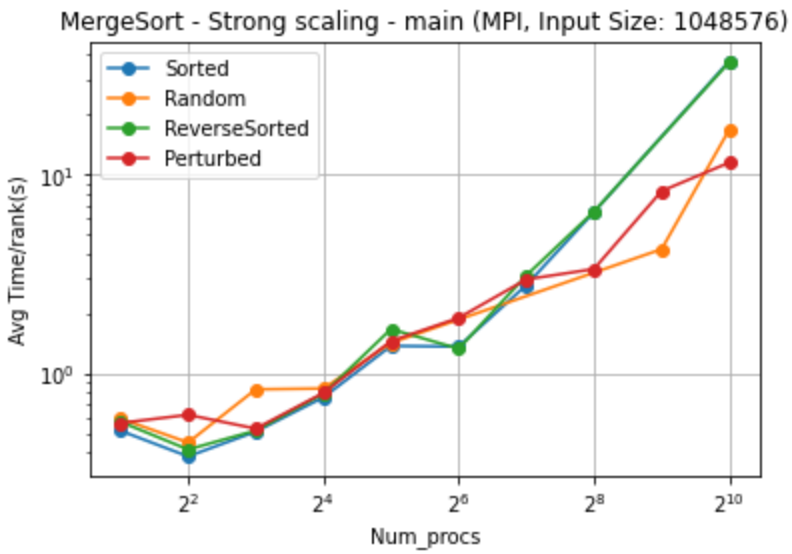

MergeSort - Strong scaling - comp_large (MPI, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{-2}
10^{-3}
10^{-4}
2^2
2^4
2^6
2^8
2^{10}
Num_procs

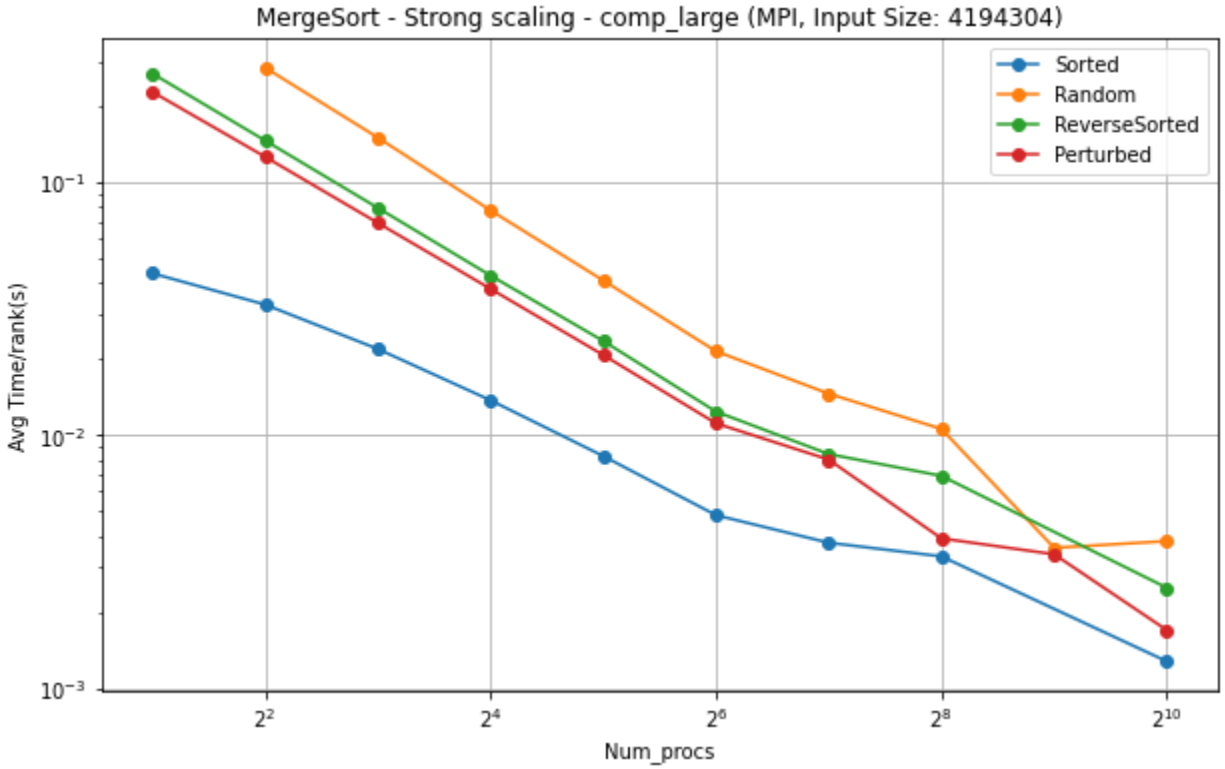

MergeSort - Strong scaling - comm (MPI, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{0}
10^{-1}
10^{-2}
10^{-3}
2^{2}
2^{4}
2^{6}
2^{8}
2^{10}
Num_procs

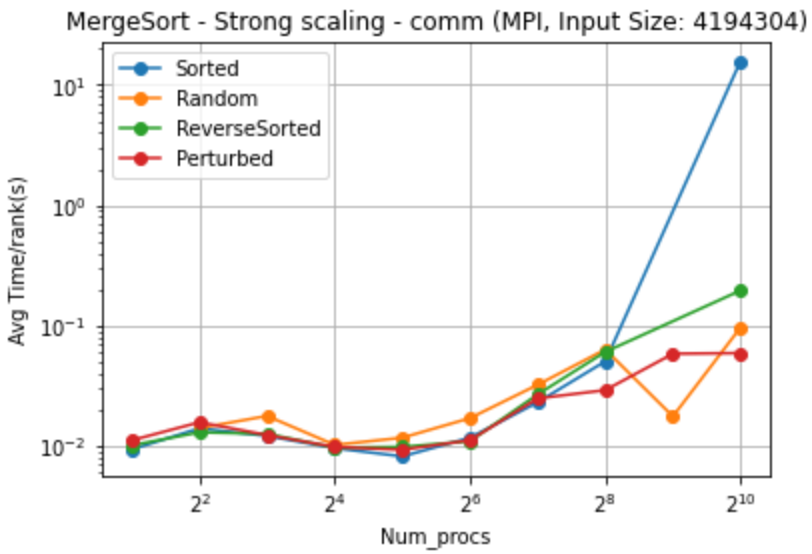

MergeSort - Strong scaling - main (MPI, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{1}
10^{0}
2^{2}
2^{4}
2^{6}
2^{8}
2^{10}
Num_procs

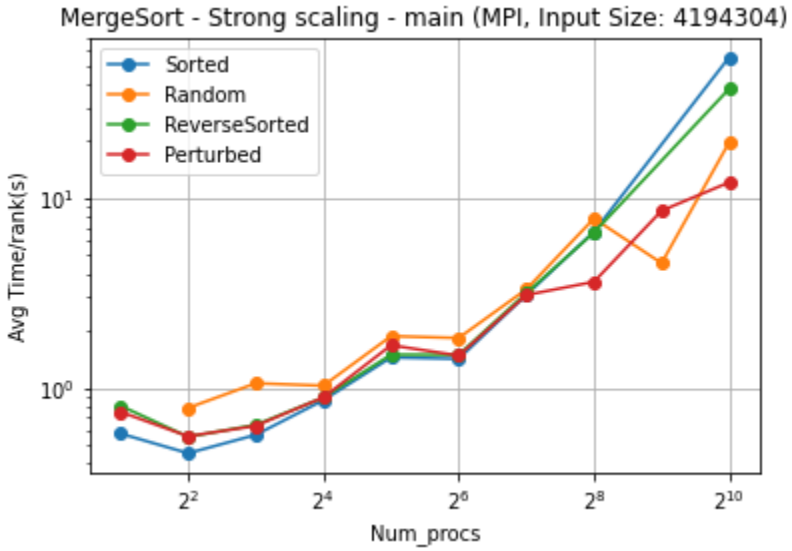

MergeSort - Strong scaling - comp_large (MPI, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{-1}
10^{-2}
10^{-3}
2^2
2^4
2^6
2^8
2^{10}
Num_procs

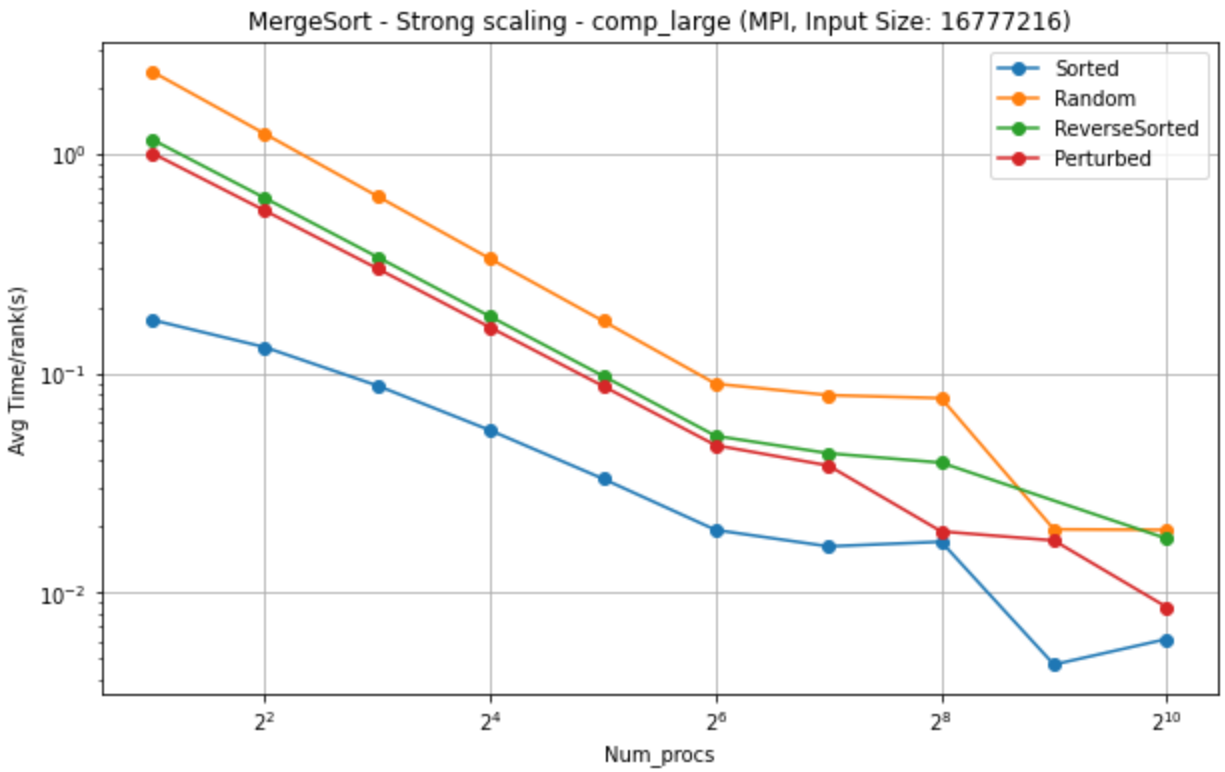

MergeSort - Strong scaling - comm (MPI, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{-1}
10^{-2}
2^2
2^4
2^6
2^8
2^{10}
Num_procs

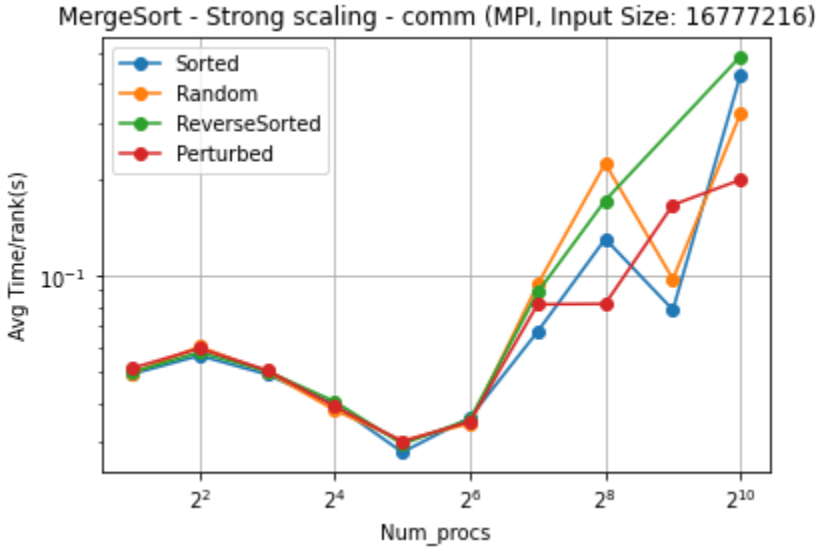

MergeSort - Strong scaling - main (MPI, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
10^0
2^2
2^4
2^6
2^8
2^{10}
Num_procs

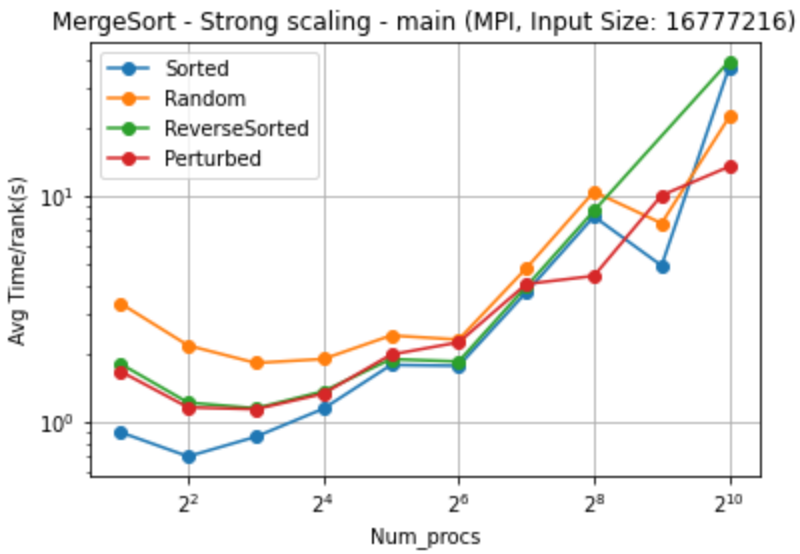

MergeSort - Strong scaling - comp_large (MPI, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{-1}
10^{-2}
10^{-3}
2^2
2^4
2^6
2^8
2^{10}
Num_procs

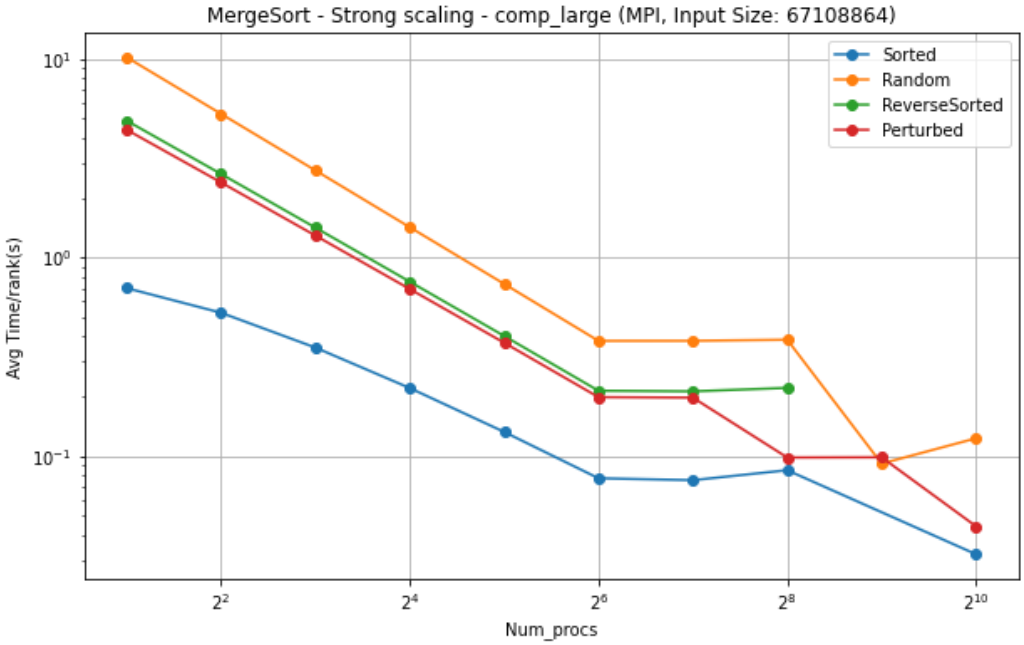

MergeSort - Strong scaling - comm (MPI, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{1}
10^{0}
10^{-1}
10^{-2}
2^{2}
2^{4}
2^{6}
2^{8}
2^{10}
Num_procs

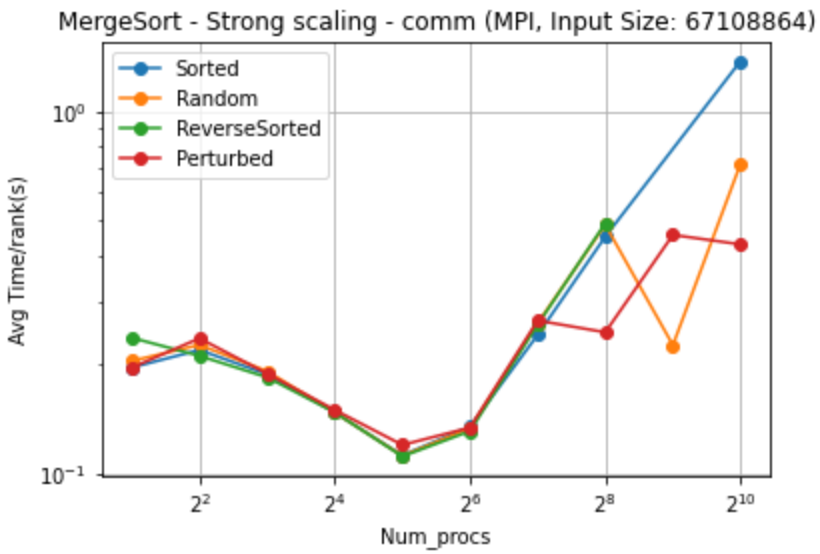

MergeSort - Strong scaling - main (MPI, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{1}
10^{0}
2^{2}
2^{4}
2^{6}
2^{8}
2^{10}
Num_procs

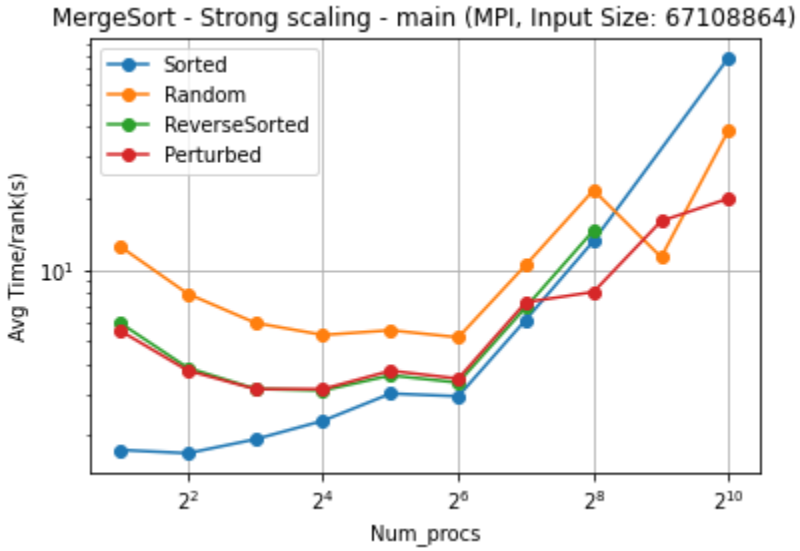

MergeSort - Strong scaling - comp_large (MPI, Input Size: 16777216)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{0}
10^{-1}
10^{-2}
2^{2}
2^{4}
2^{6}
2^{8}
2^{10}
Num_procs

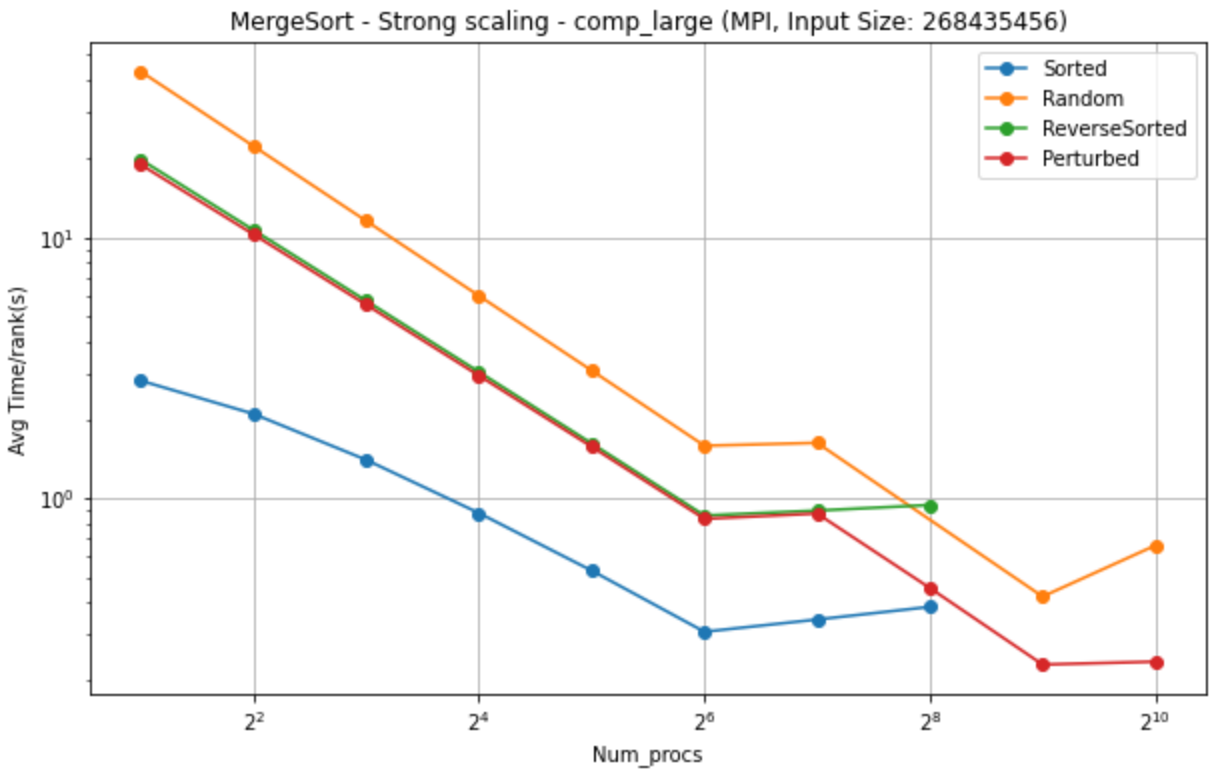

MergeSort - Strong scaling - comm (MPI, Input Size: 16777216)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{-1}
2^{2}
2^{4}
2^{6}
2^{8}
2^{10}
Num_procs

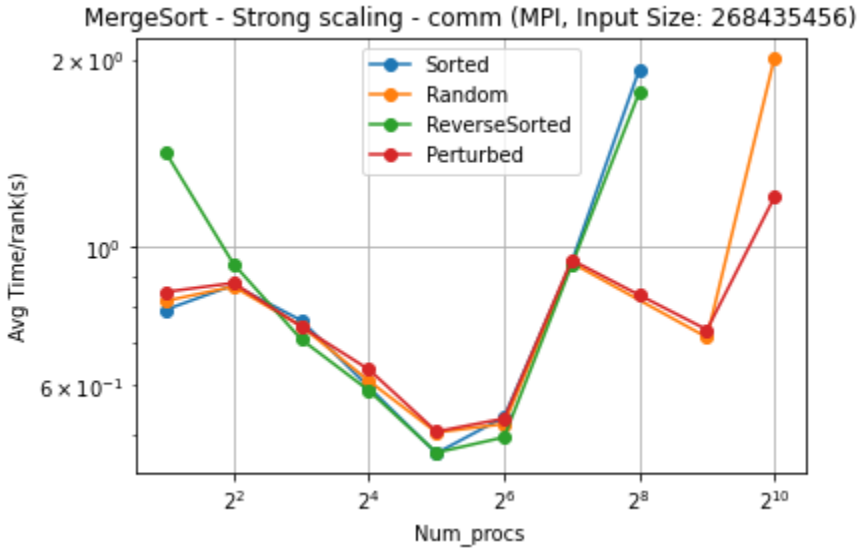

MergeSort - Strong scaling - main (MPI, Input Size: 16777216)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
10^0
2^2
2^4
2^6
2^8
2^{10}
Num_procs

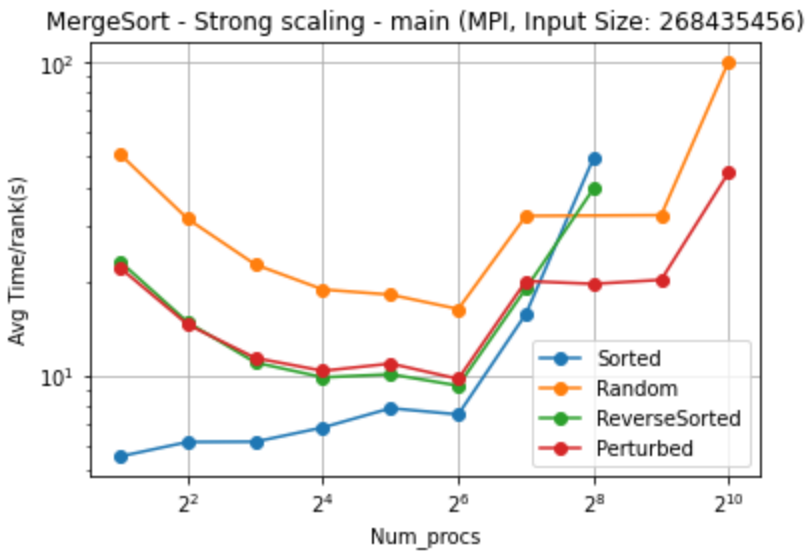

MergeSort - Strong scaling - comp_large (MPI, Input Size: 67108864)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
10^0
10^{-1}
2^2
2^4
2^6
2^8
2^{10}
Num_procs


MergeSort - Strong scaling - comm (MPI, Input Size: 67108864)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^0
10^{-1}
2^2
2^4
2^6
2^8
2^{10}
Num_procs

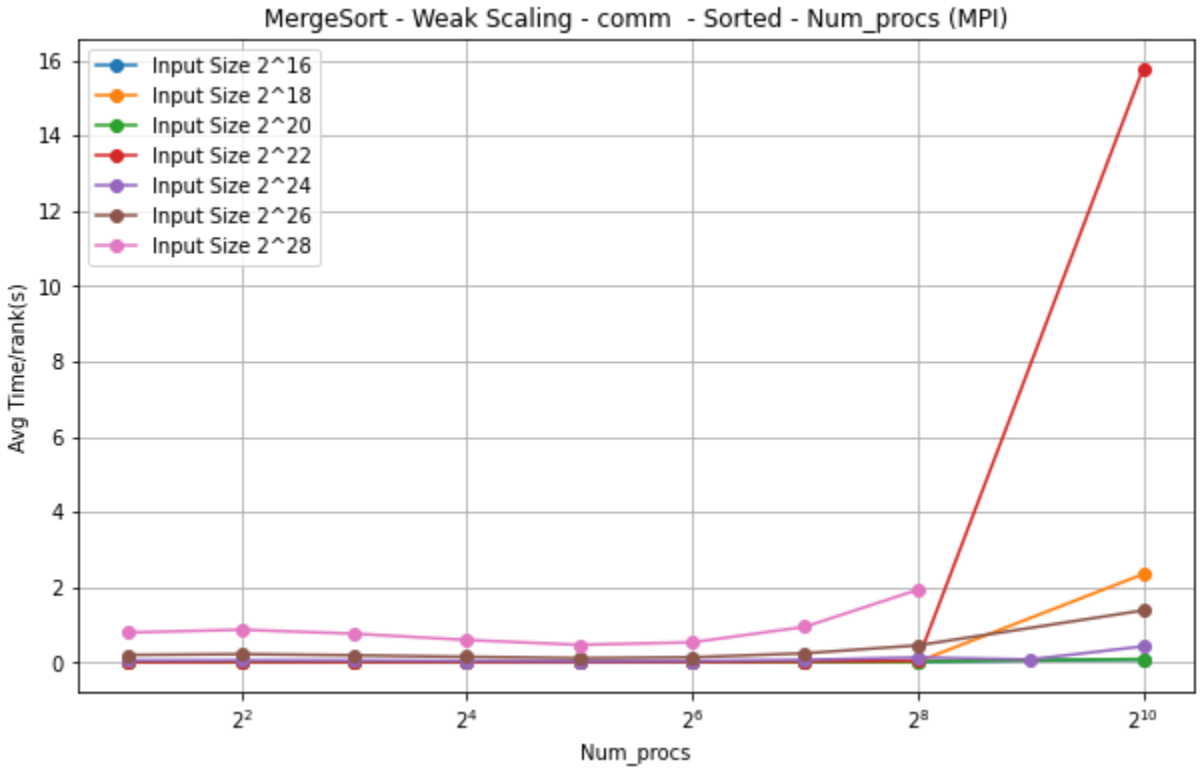

MergeSort - Strong scaling - main (MPI, Input Size: 67108864)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
2^2
2^4
2^6
2^8
2^{10}
Num_procs

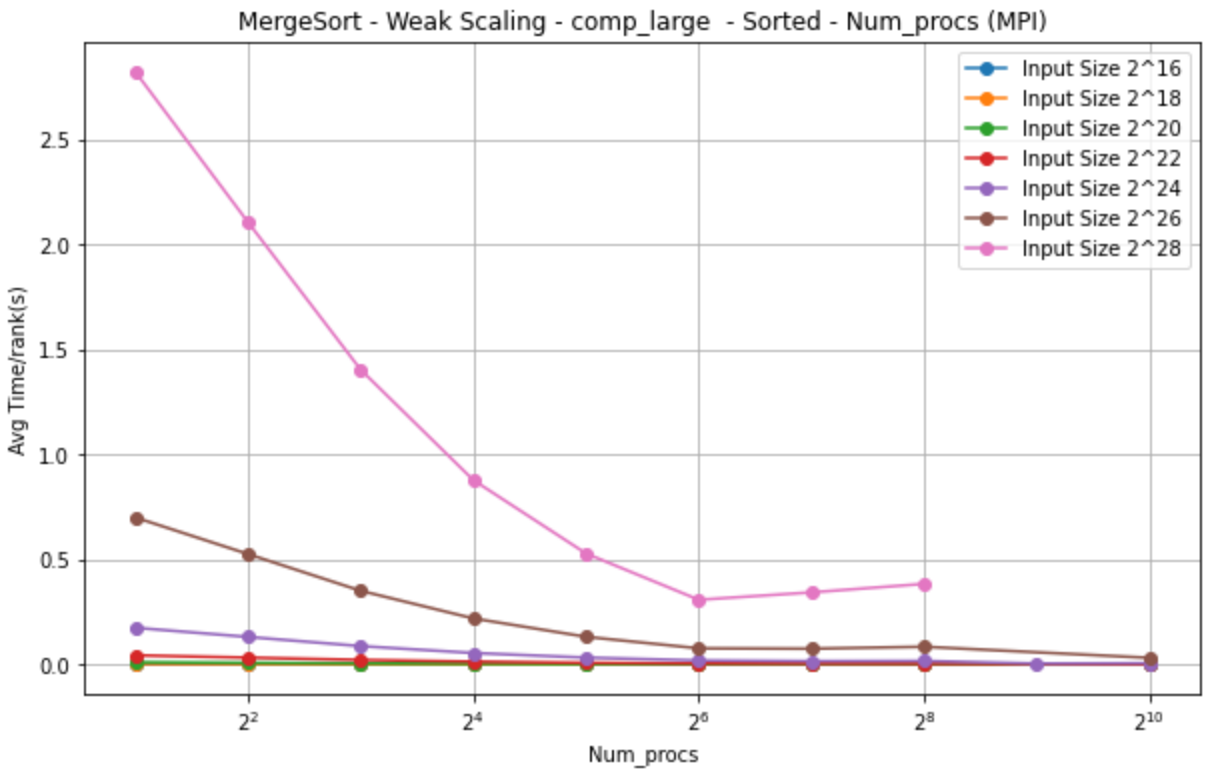

MergeSort - Strong scaling - comp_large (MPI, Input Size: 268435456)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
10^0
2^2
2^4
2^6
2^8
2^{10}
Num_procs

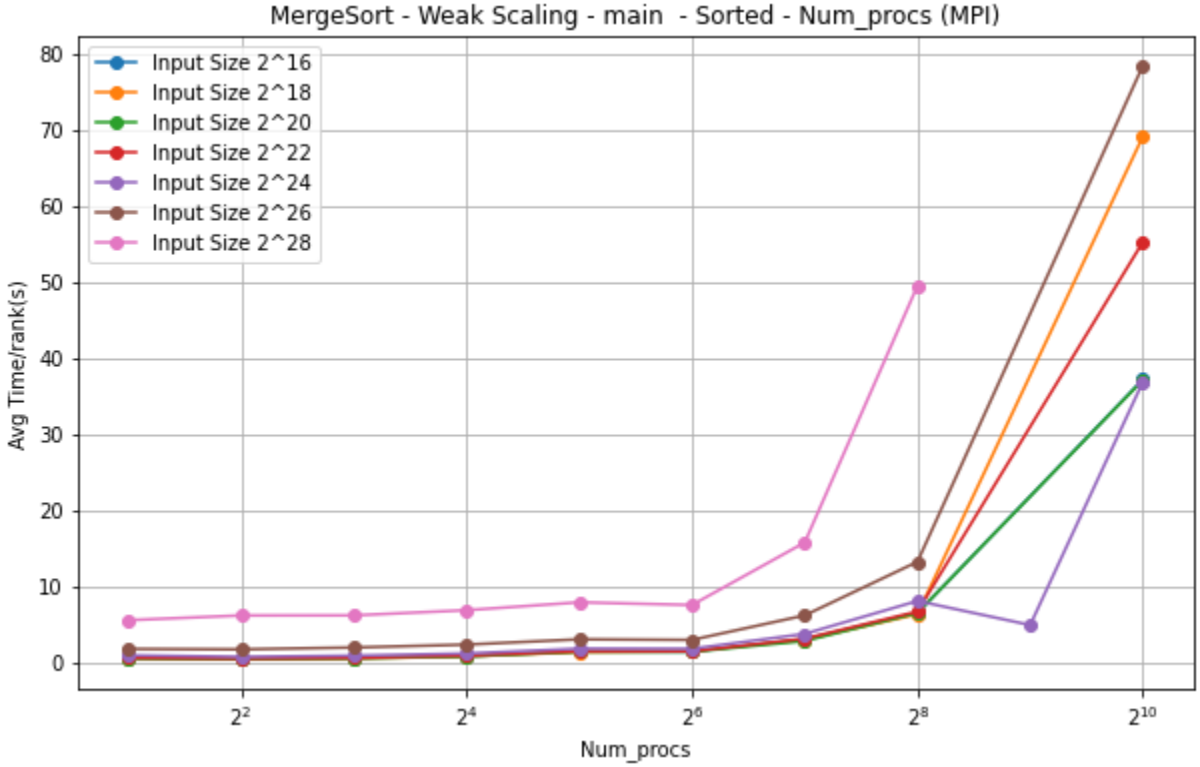

MergeSort - Strong scaling - comm (MPI, Input Size: 268435456)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
2×10^0
10^0
6×10^{-1}
2^2
2^4
2^6
2^8
2^{10}
Num_procs

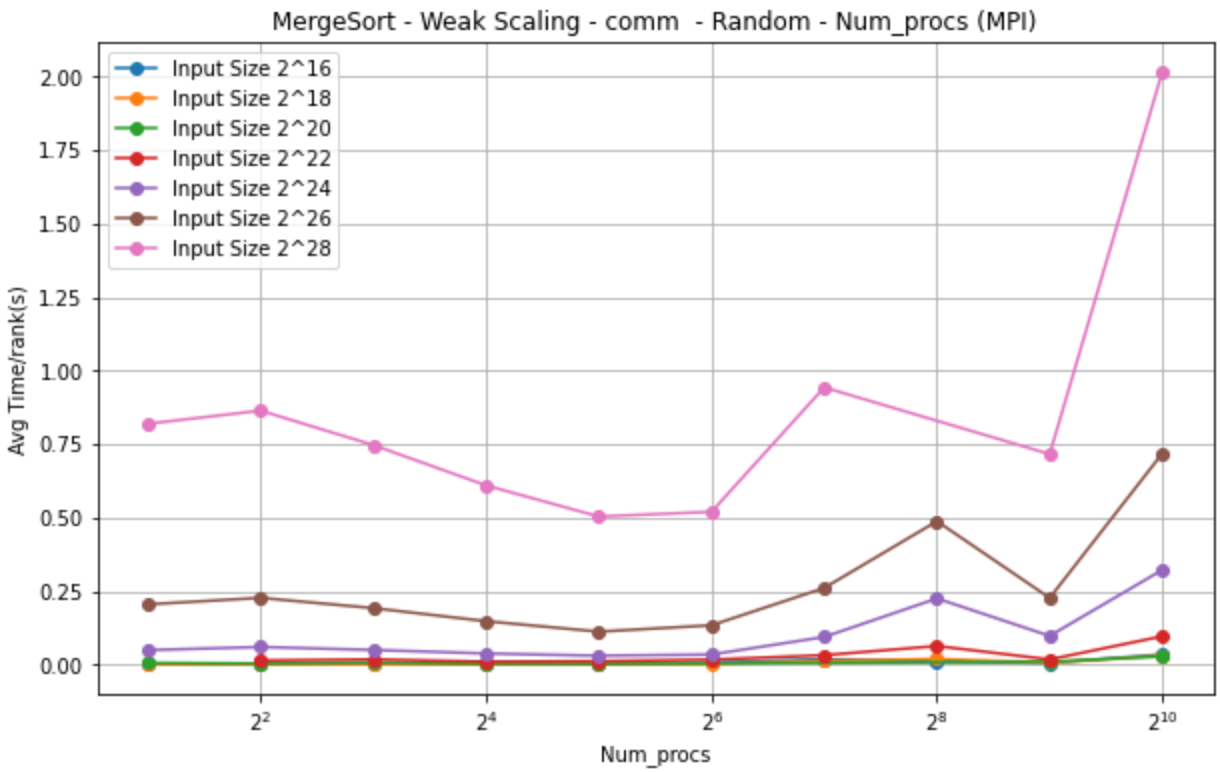


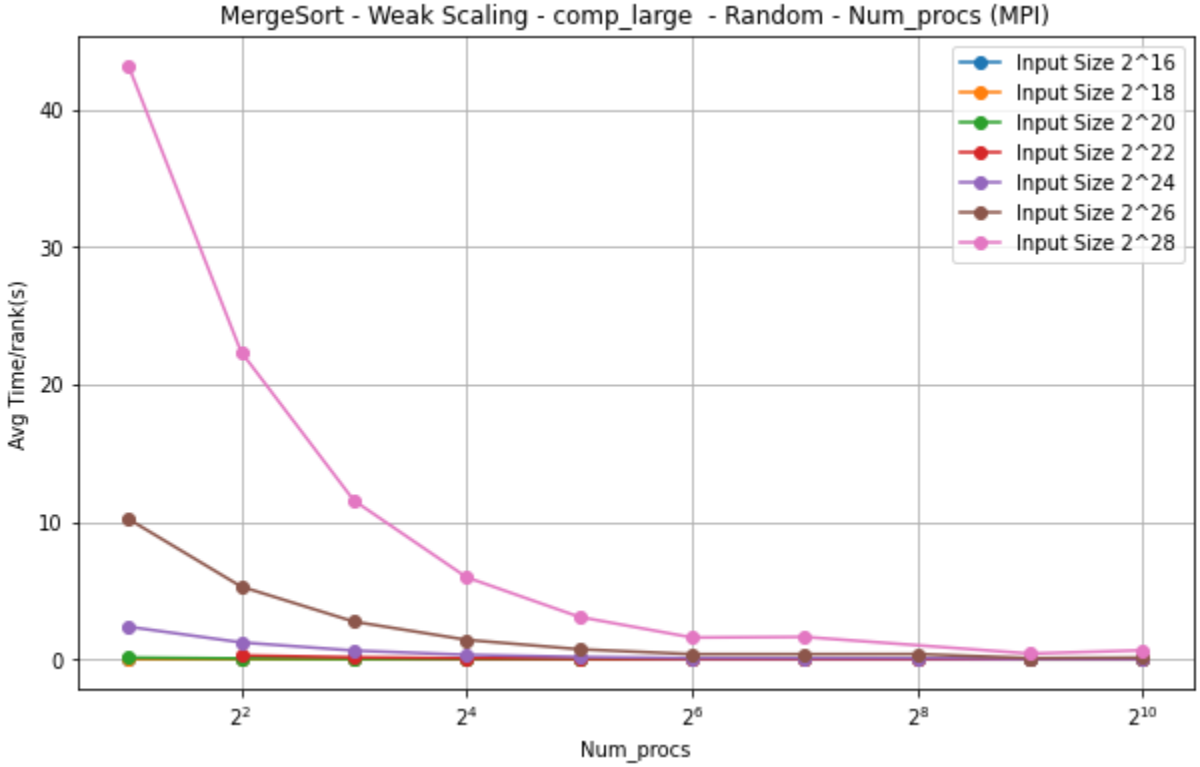
Weak Scaling:

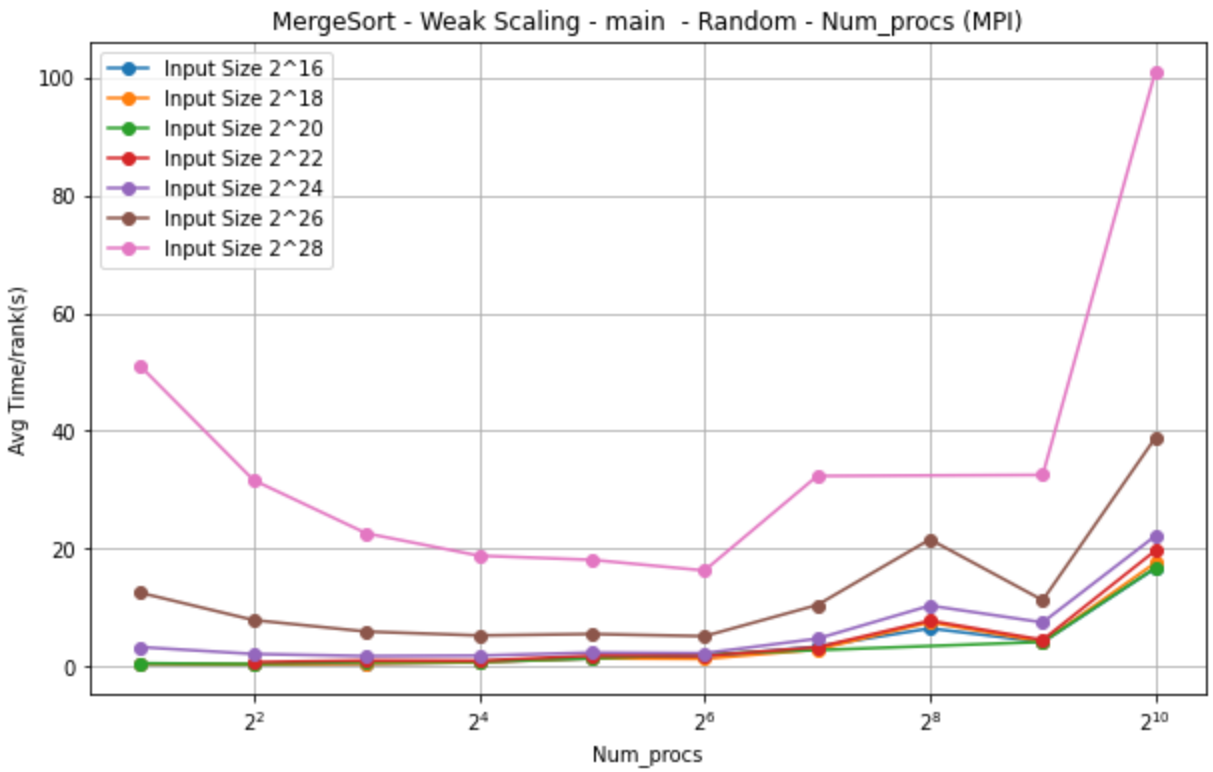


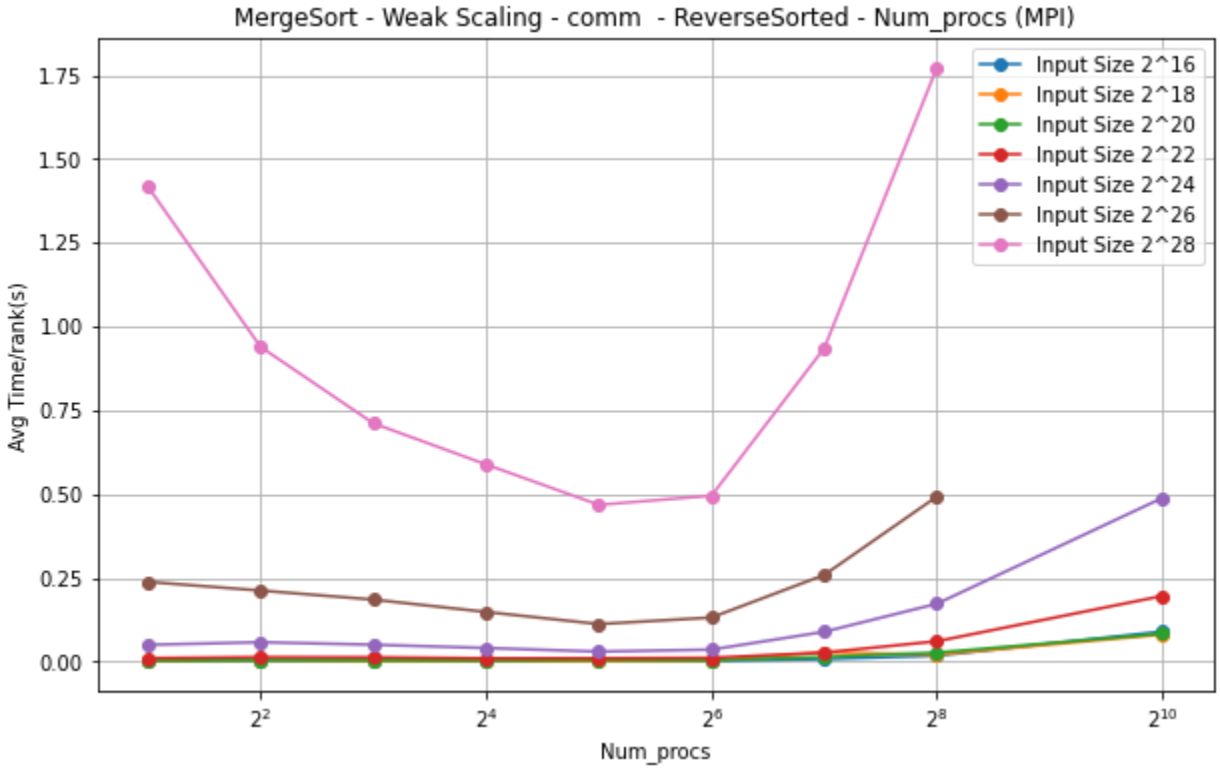

MergeSort - Weak Scaling - comp_large - Sorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
0.0
0.5
1.0
1.5
2.0
2.5
2^2
2^4
2^6
2^8
2^{10}
Num_procs

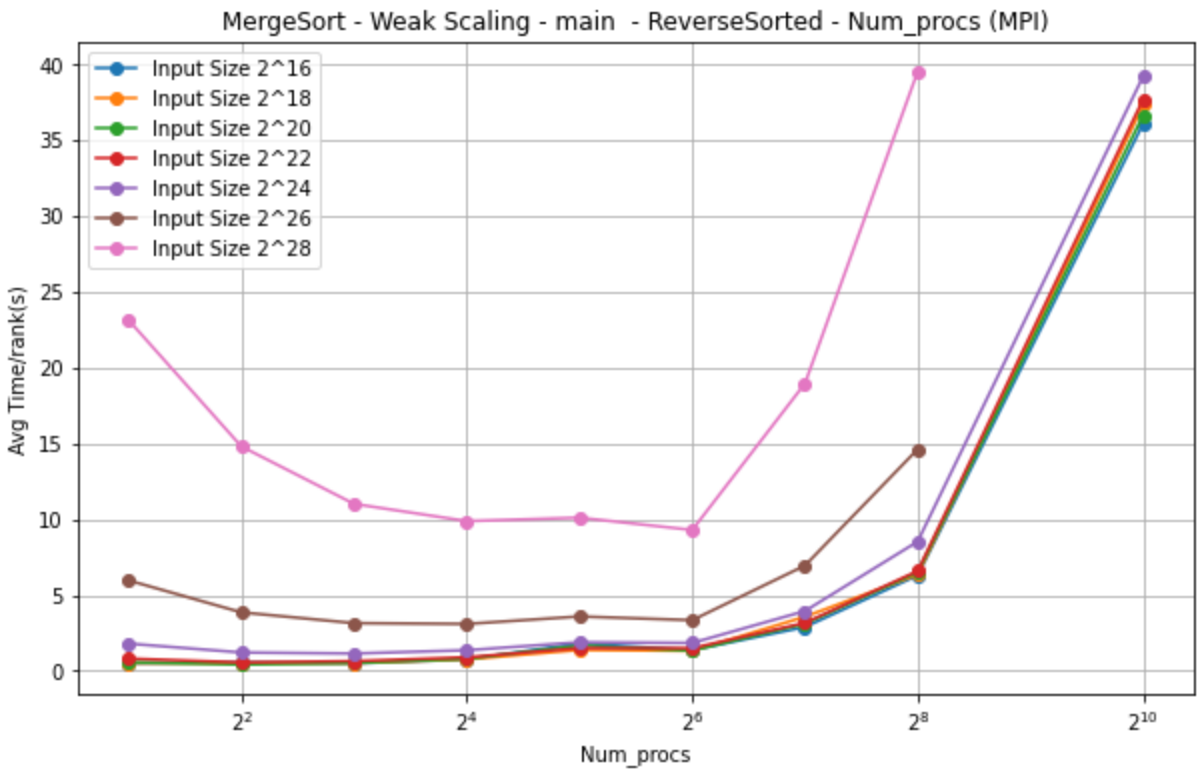

MergeSort - Weak Scaling - main - Sorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
0
10
20
30
40
50
60
70
80
2^2
2^4
2^6
2^8
2^{10}
Num_procs

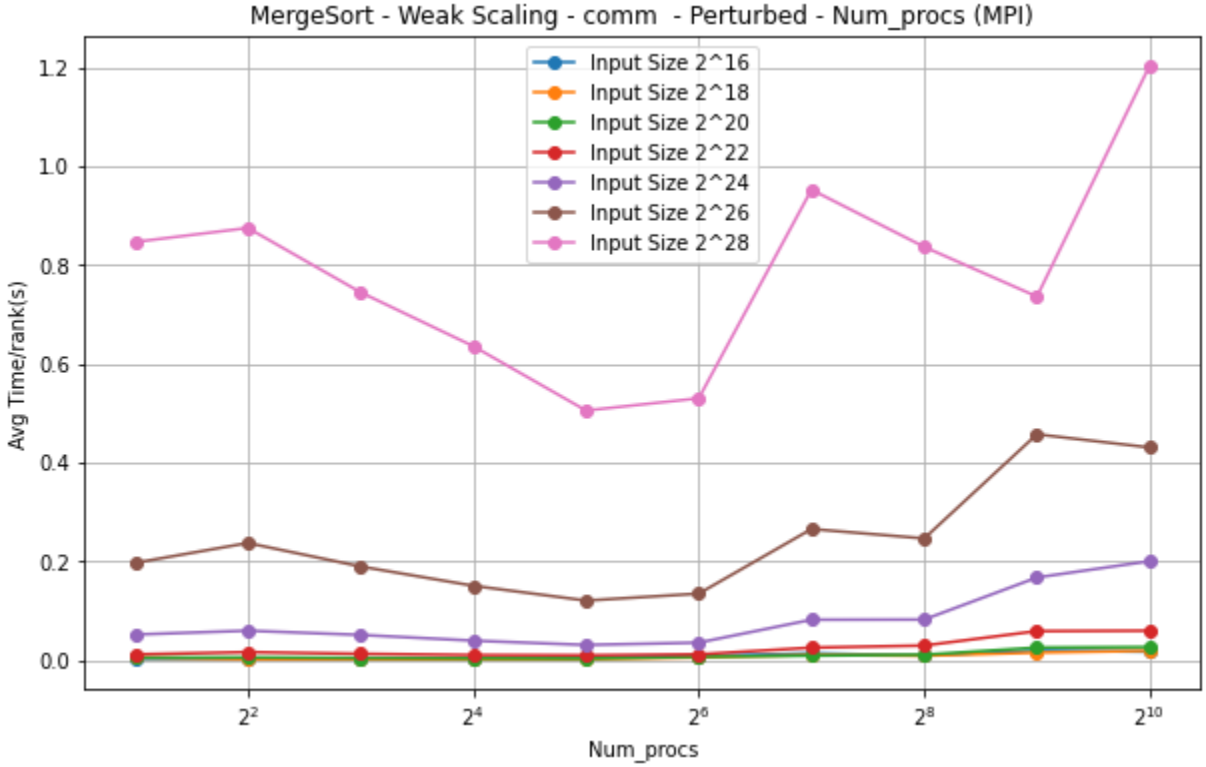

MergeSort - Weak Scaling - comm - Random - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
Num_procs
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2^2
2^4
2^6
2^8
2^{10}

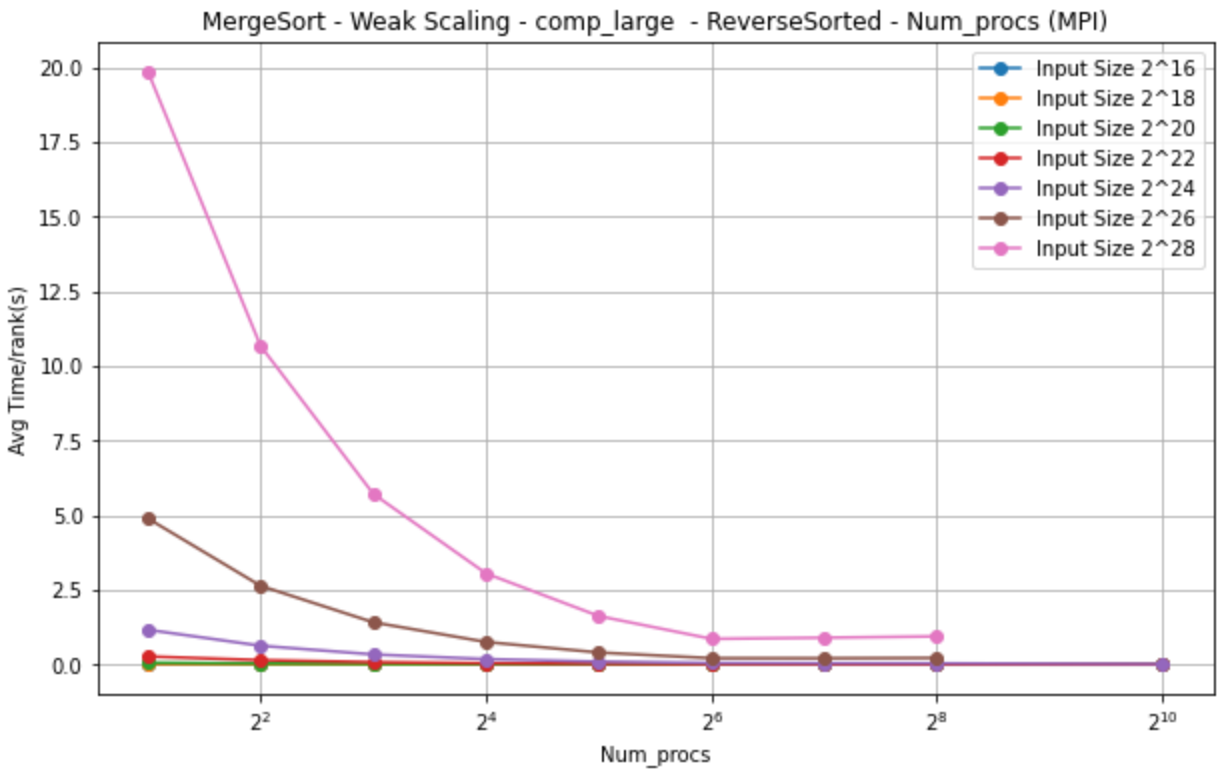

MergeSort - Weak Scaling - comp_large - Random - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
Num_procs
0
10
20
30
40
2^2
2^4
2^6
2^8
2^{10}

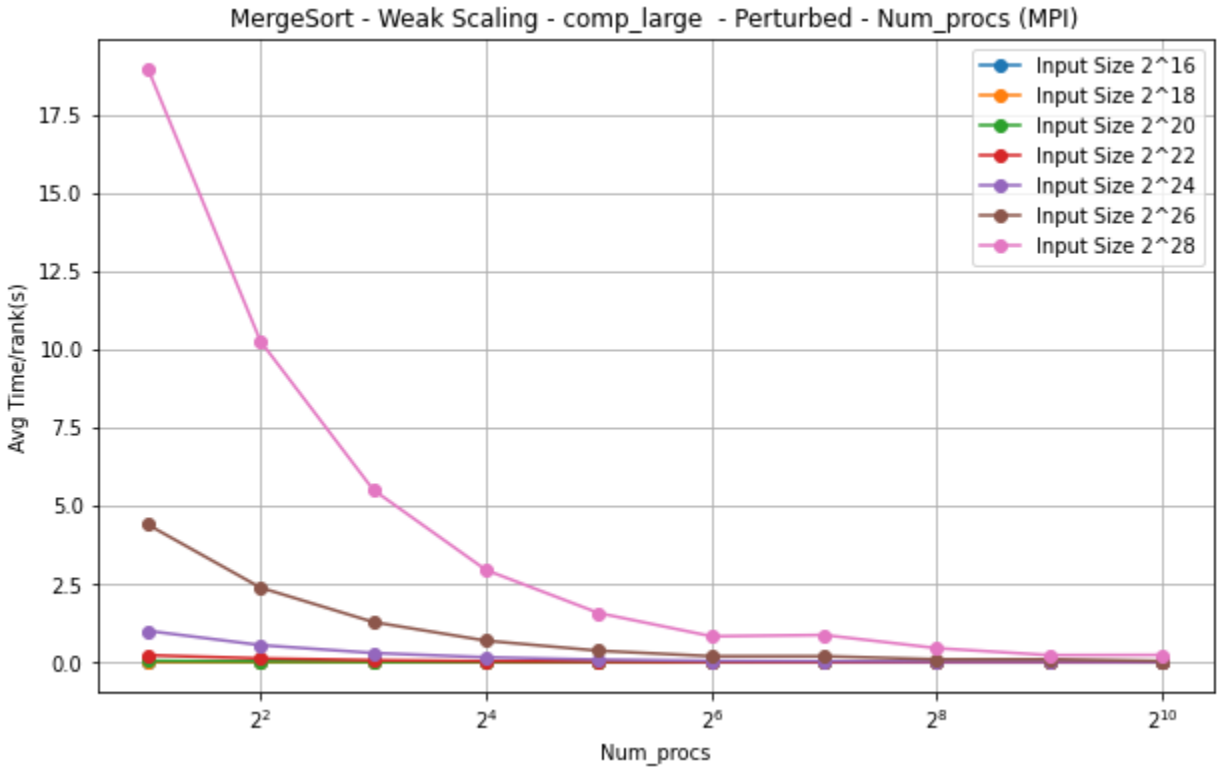

MergeSort - Weak Scaling - main - Random - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
Num_procs
0
20
40
60
80
100
2^2
2^4
2^6
2^8
2^{10}

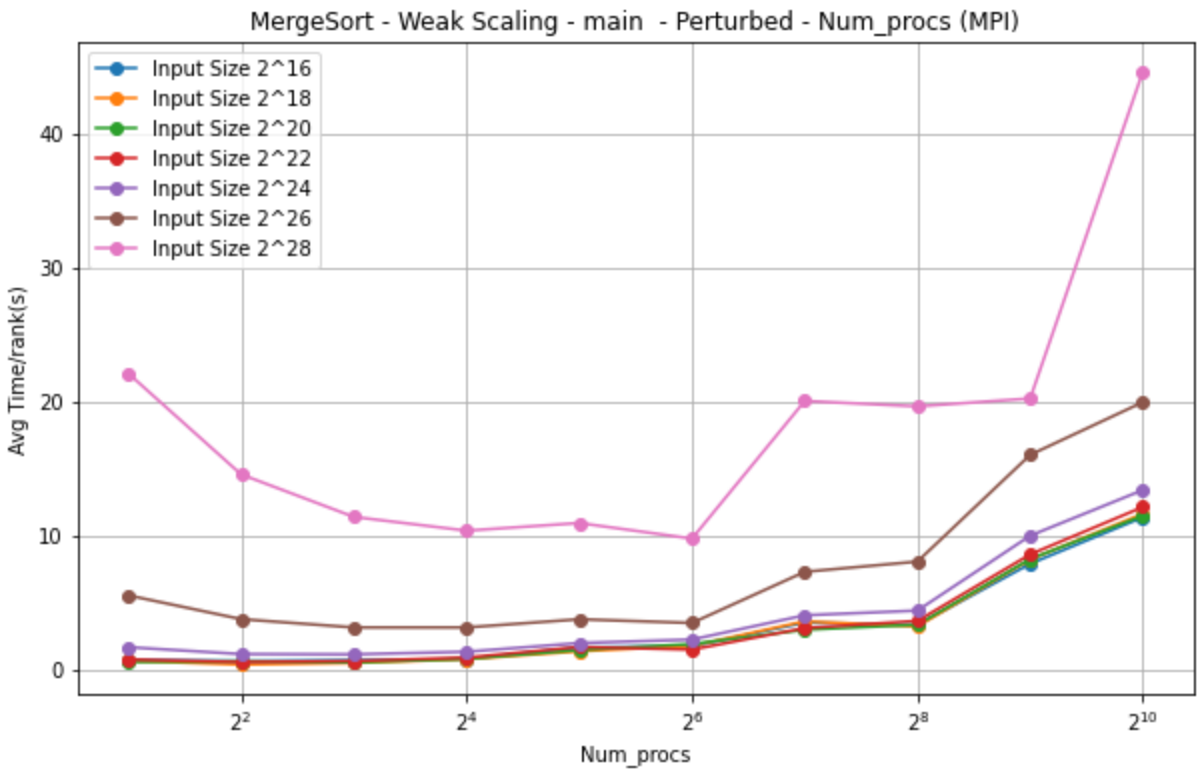

MergeSort - Weak Scaling - comm - ReverseSorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
Num_procs
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2^2
2^4
2^6
2^8
2^{10}


MergeSort - Weak Scaling - main - ReverseSorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
Num_procs

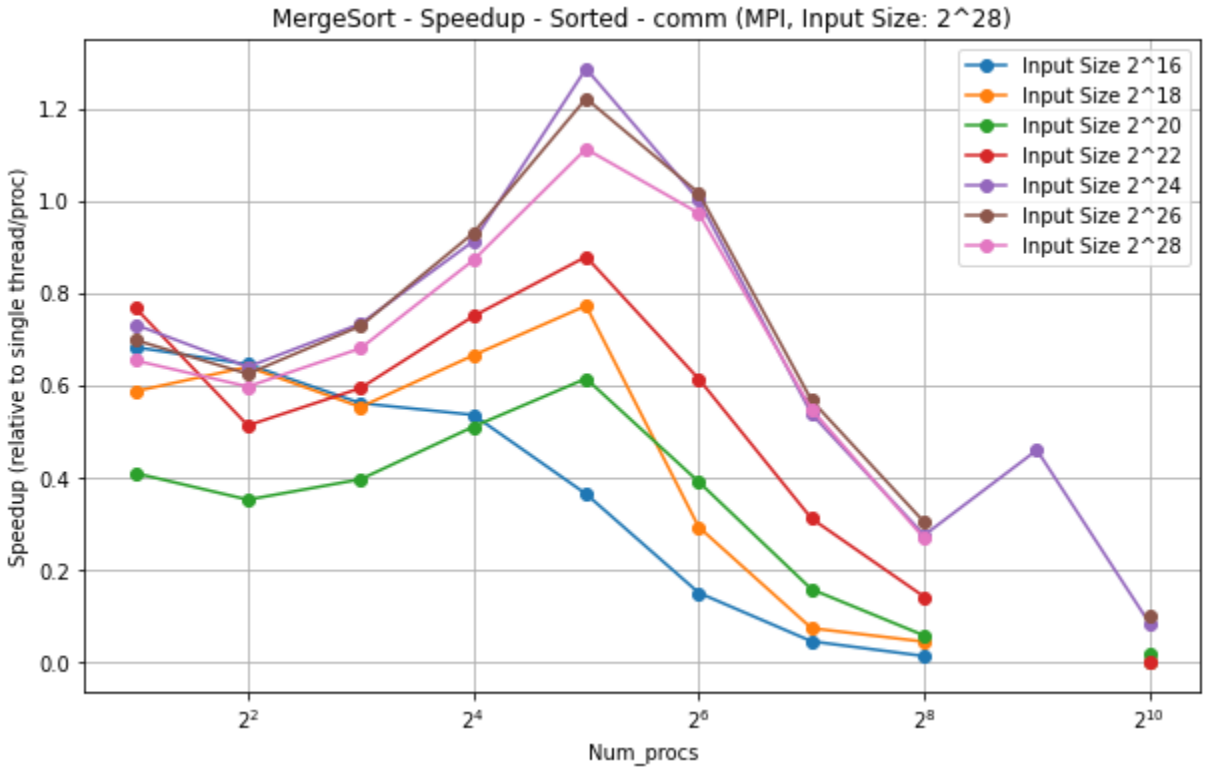

MergeSort - Weak Scaling - comm - Perturbed - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
Num_procs

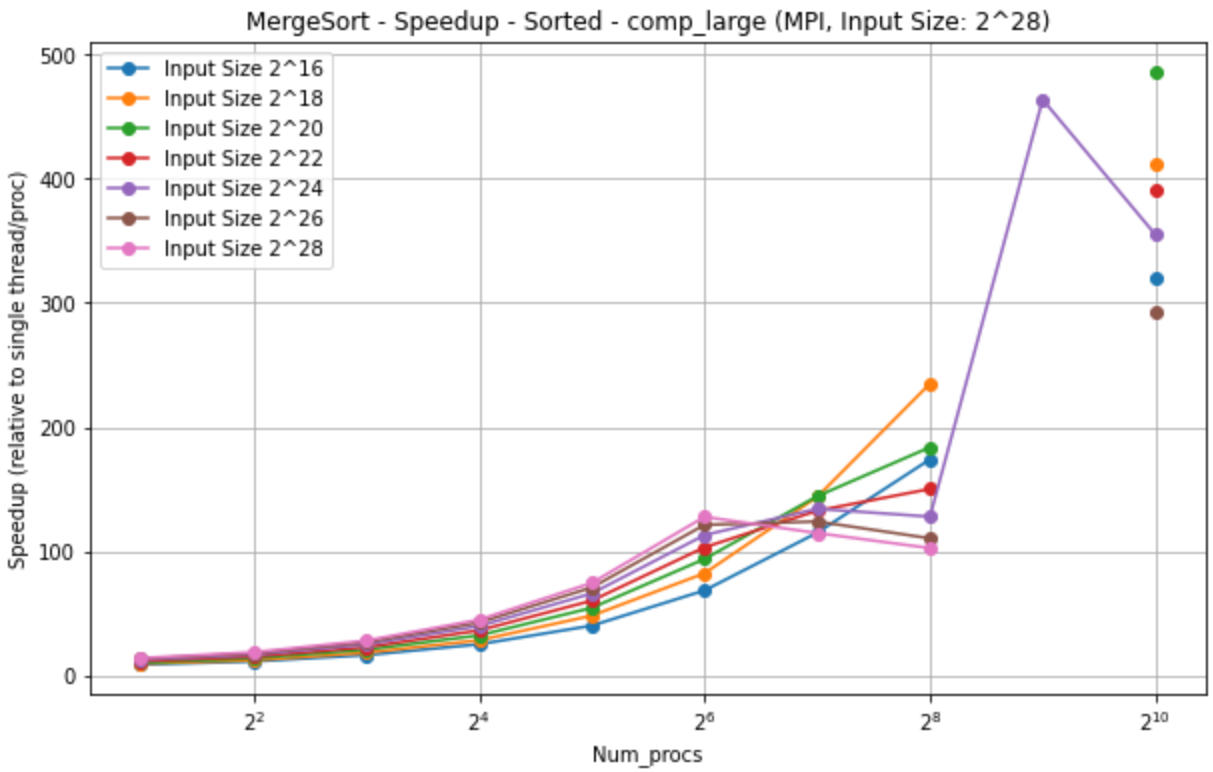

MergeSort - Weak Scaling - comp_large - ReverseSorted - Num_procs (MPI)
Avg Time/rank(s)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

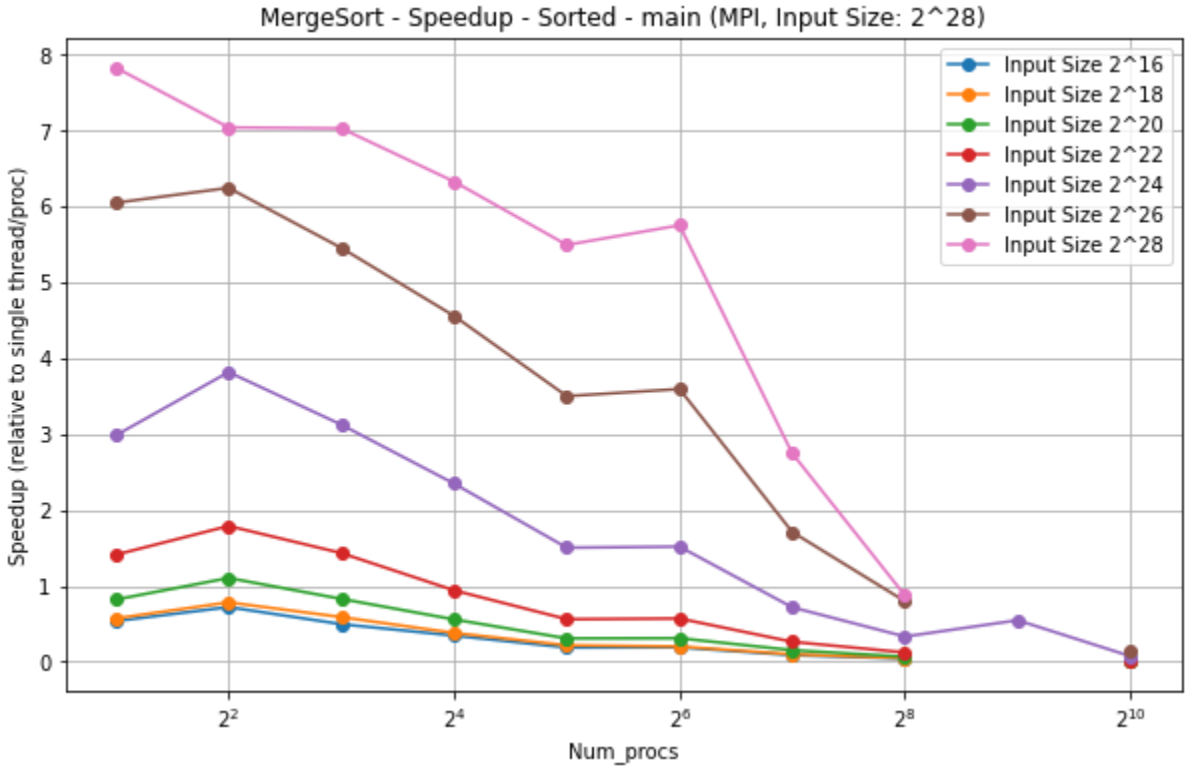

MergeSort - Weak Scaling - comp_large - Perturbed - Num_procs (MPI)
Avg Time/rank(s)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

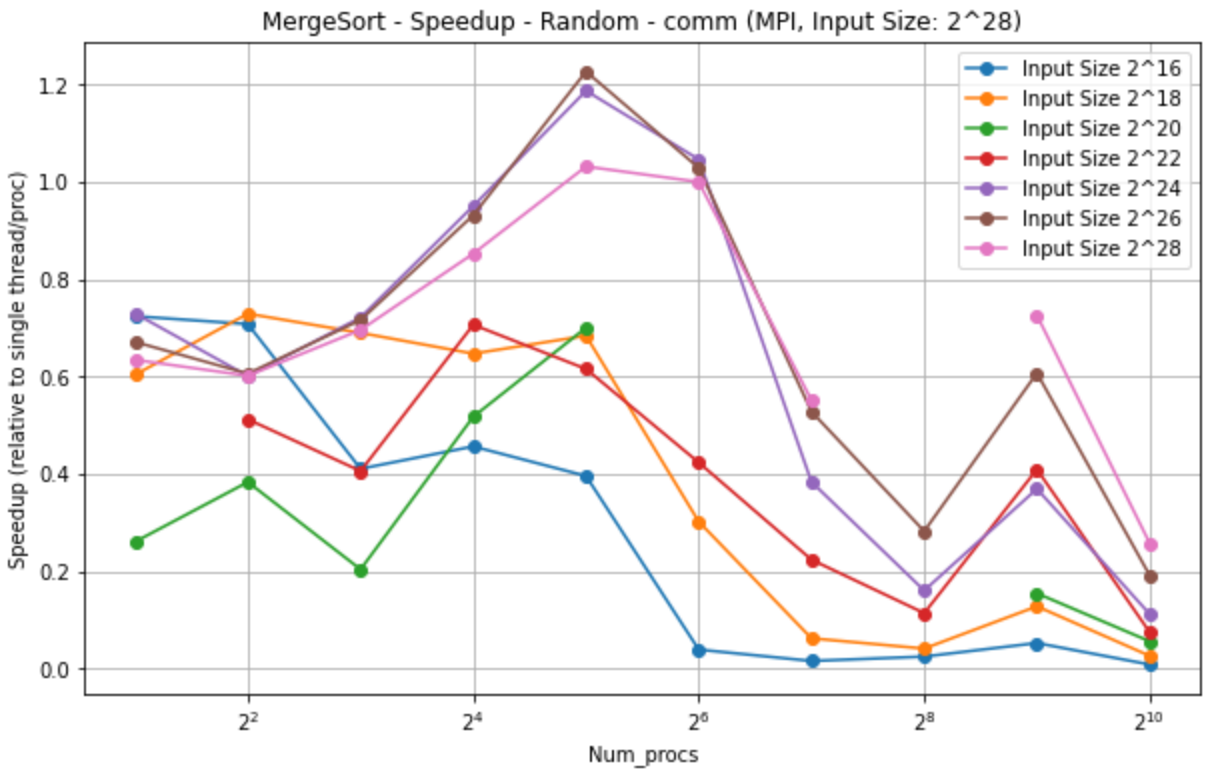

MergeSort - Weak Scaling - main - Perturbed - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
0
10
20
30
40
2^2
2^4
2^6
2^8
2^{10}
Num_procs

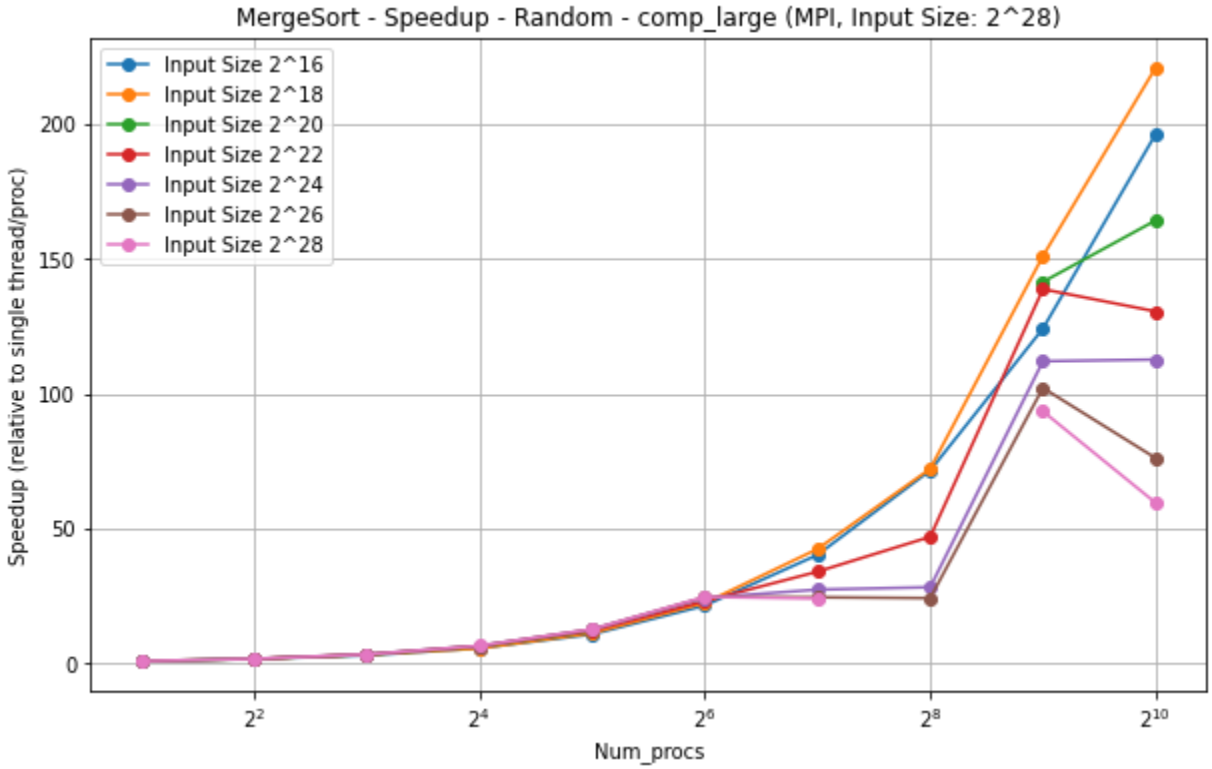
Speedup:

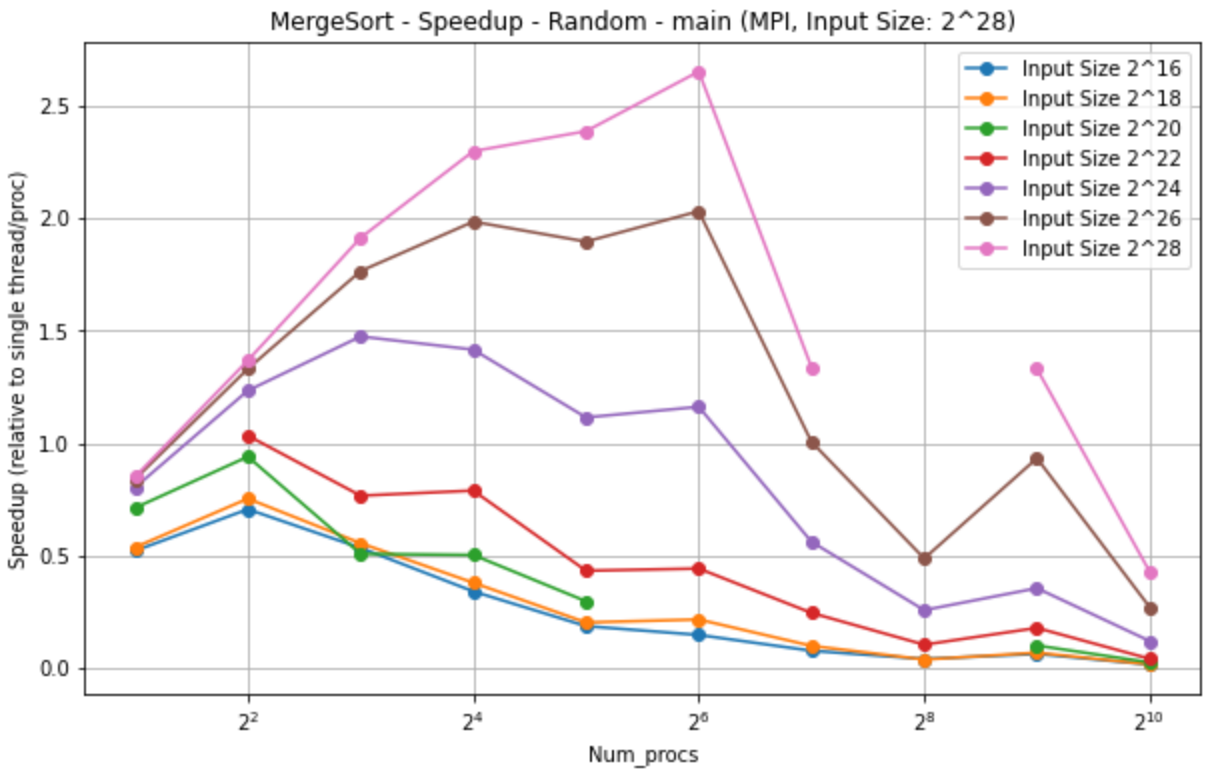

MergeSort - Speedup - Sorted - comm (MPI, Input Size: 2^28)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Speedup (relative to single thread/proc)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
2^2
2^4
2^6
2^8
2^{10}
Num_procs

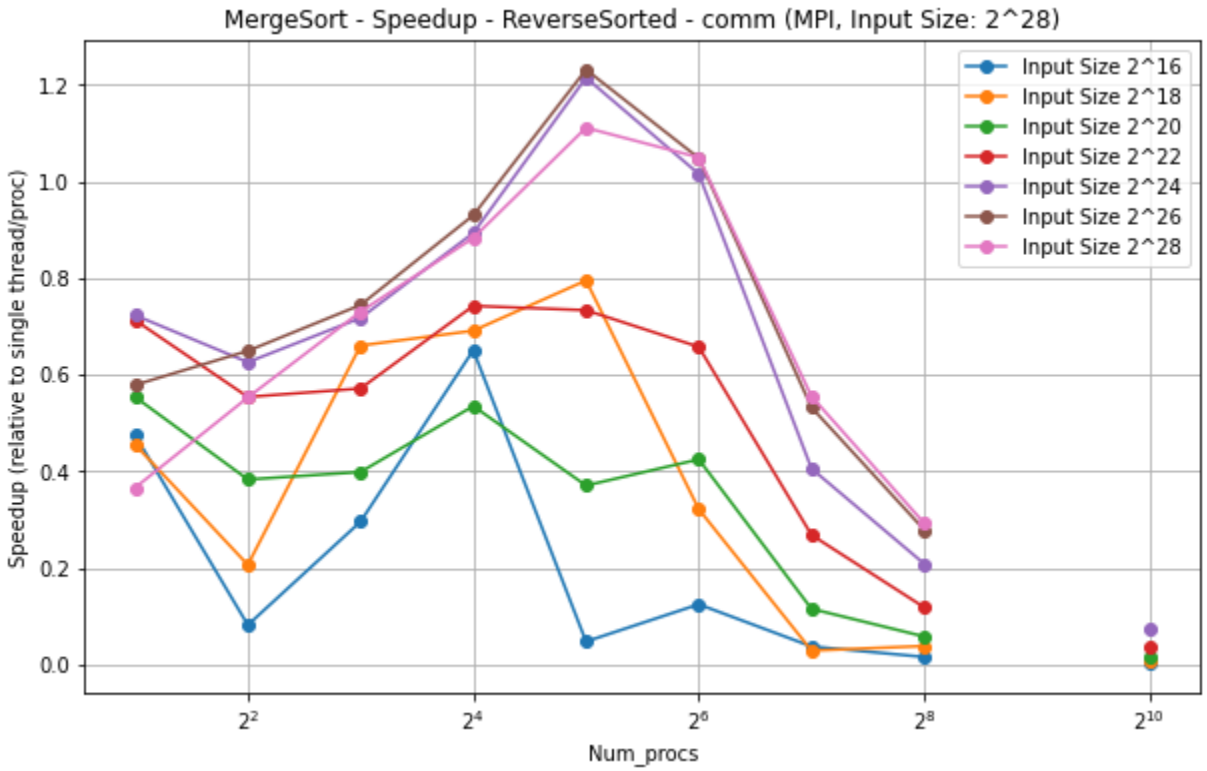

MergeSort - Speedup - Sorted - comp_large (MPI, Input Size: 2^28)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Speedup (relative to single thread/proc)
Num_procs

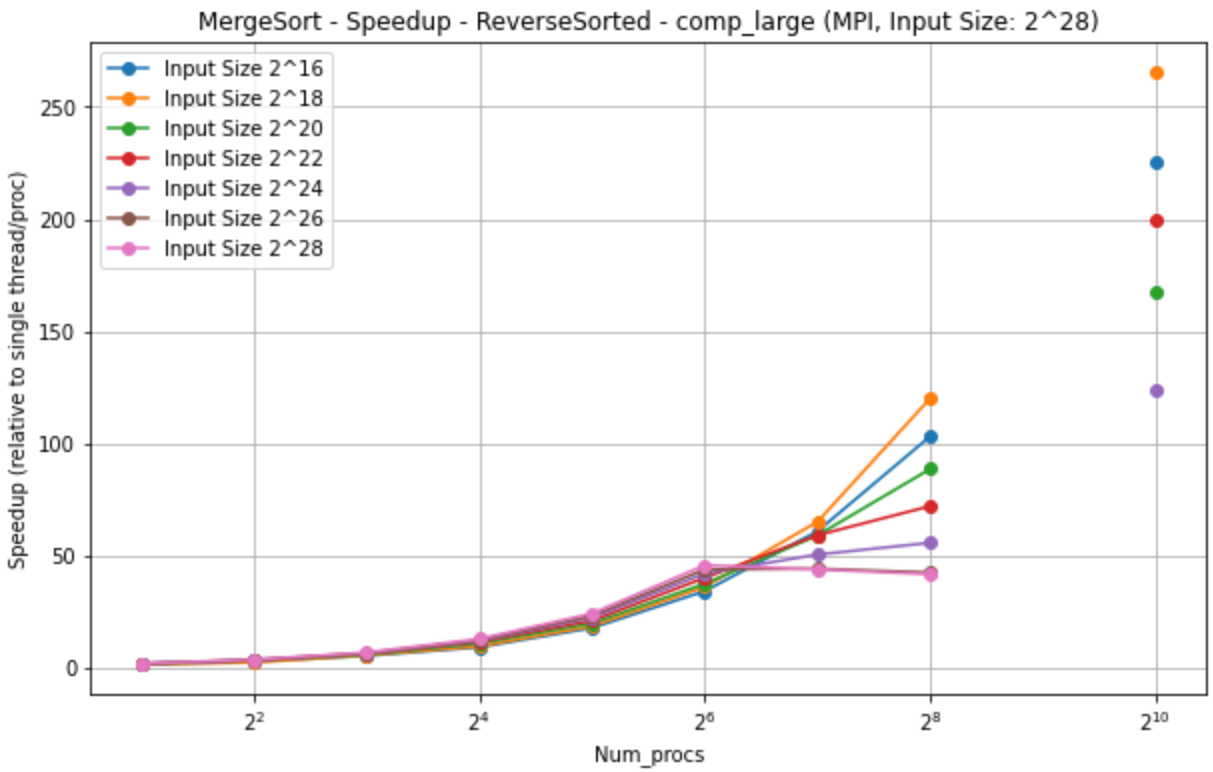

MergeSort - Speedup - Sorted - main (MPI, Input Size: 2^28)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Speedup (relative to single thread/proc)
Num_procs

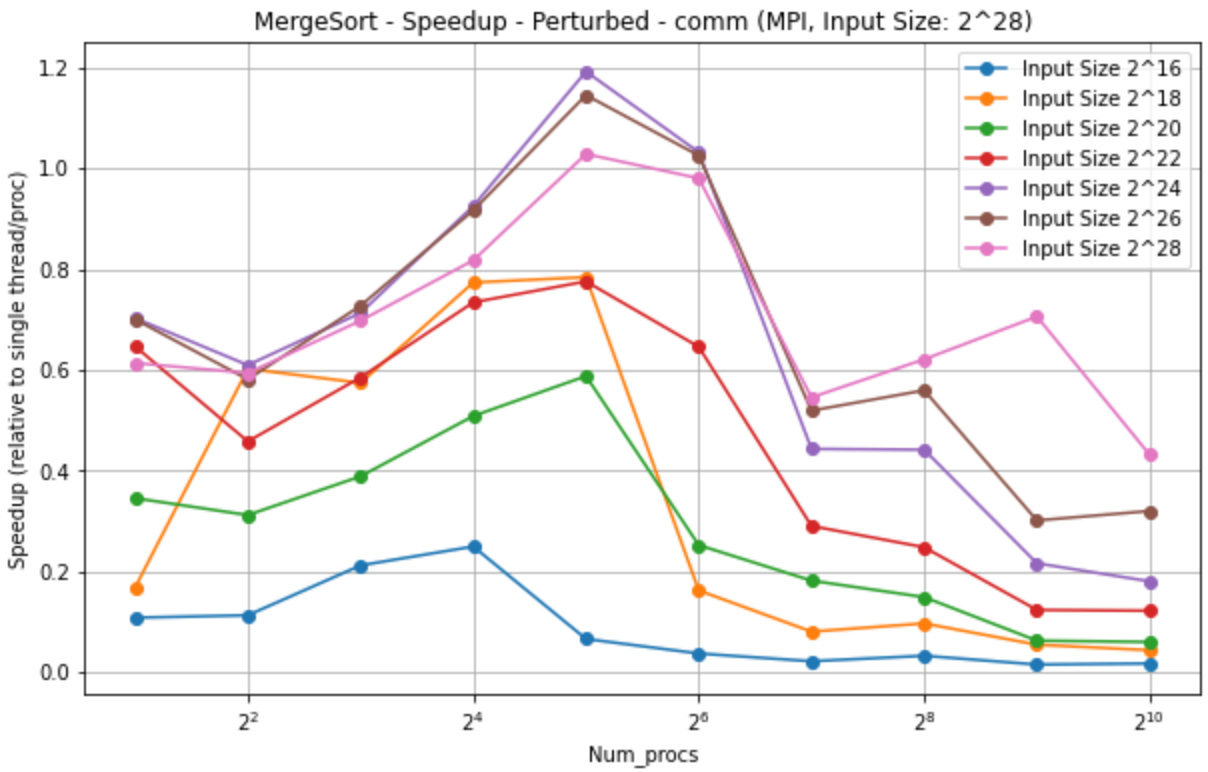

MergeSort - Speedup - Random - comm (MPI, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

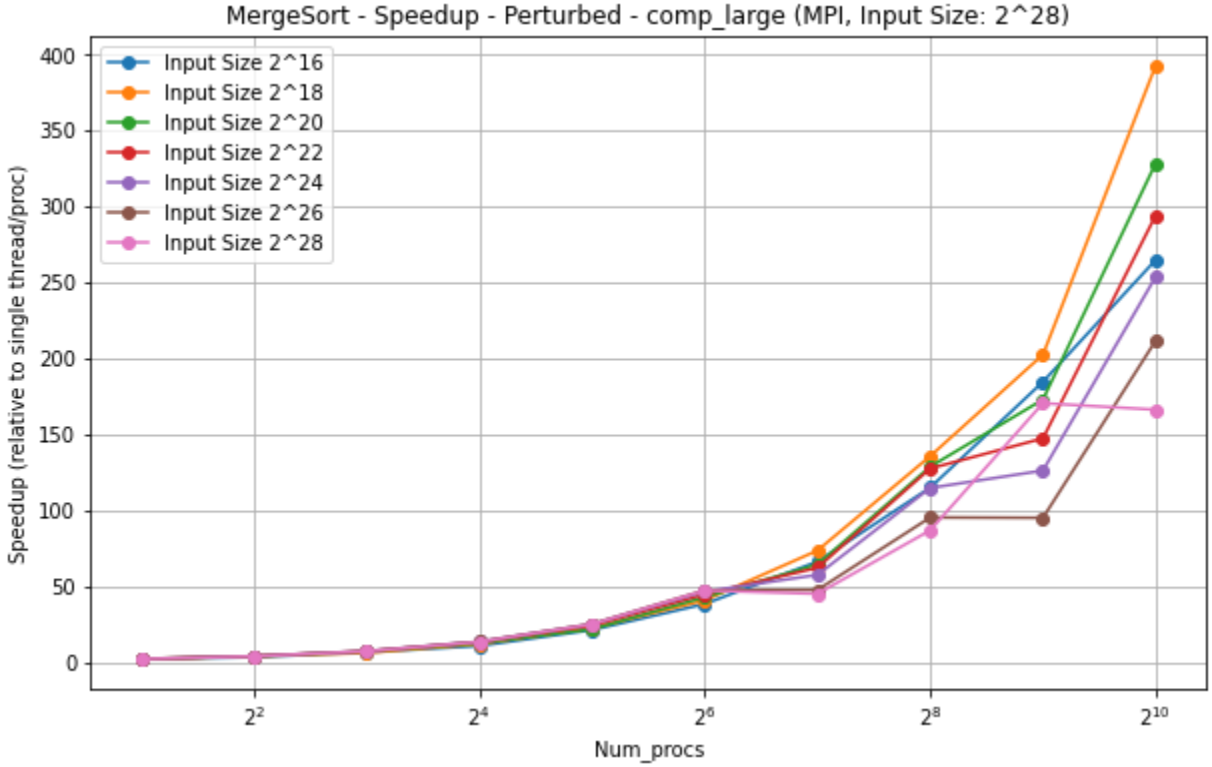

MergeSort - Speedup - Random - comp_large (MPI, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

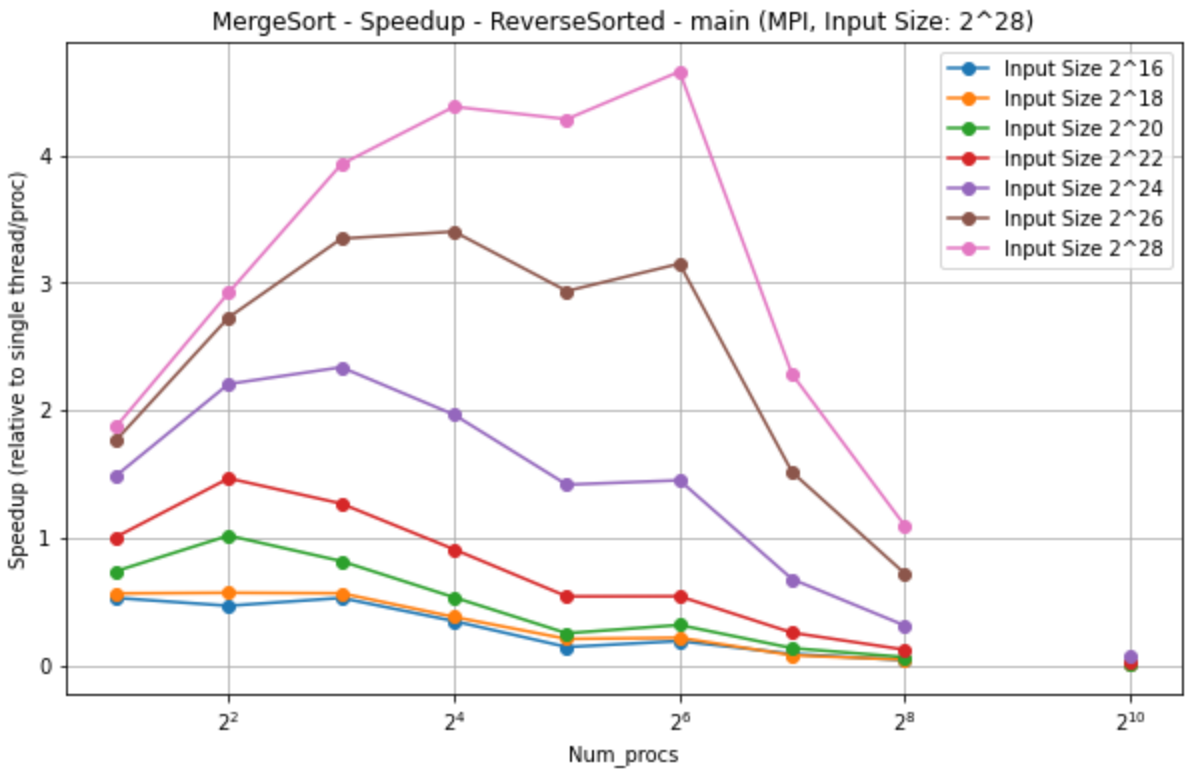

MergeSort - Speedup - Random - main (MPI, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

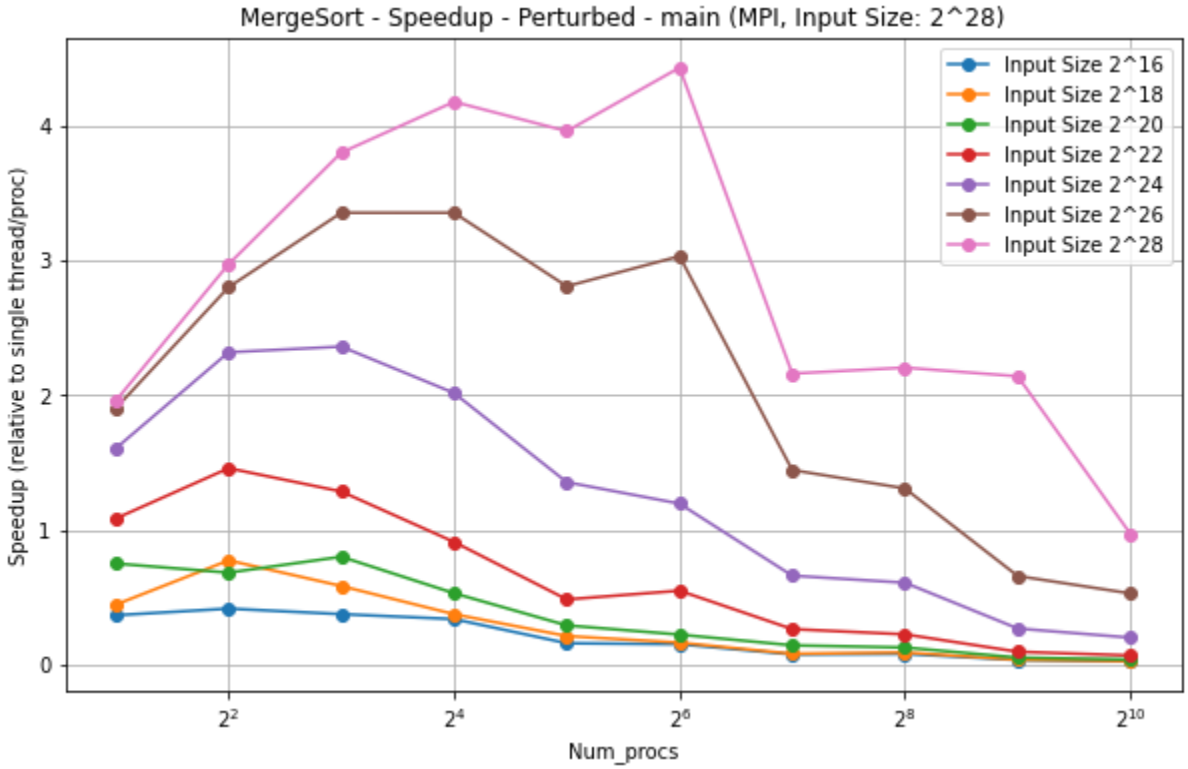

MergeSort - Speedup - ReverseSorted - comm (MPI, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28


MergeSort - Speedup - ReverseSorted - comp_large (MPI, Input Size: 2^28)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Speedup (relative to single thread/proc)
0
50
100
150
200
250
2^2
2^4
2^6
2^8
2^{10}
Num_procs


MergeSort - Speedup - Perturbed - comm (MPI, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
0.0
0.2
0.4
0.6
0.8
1.0
1.2
2^2
2^4
2^6
2^8
2^{10}

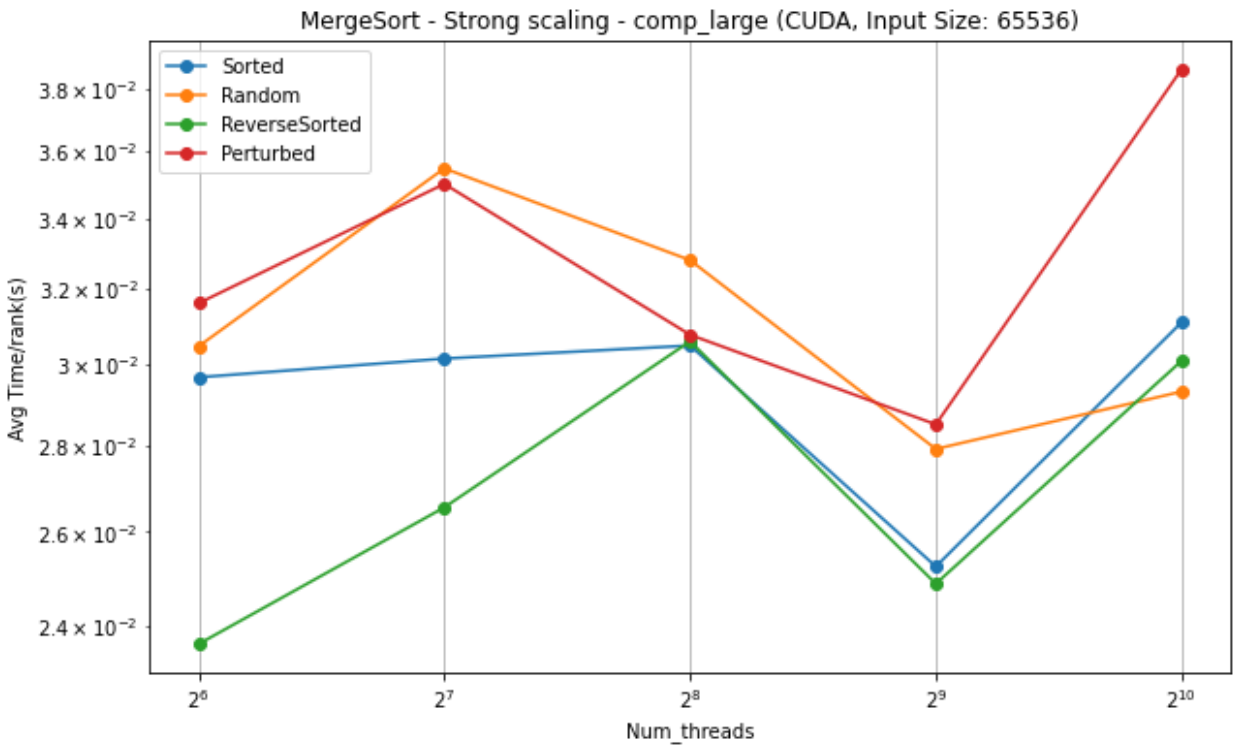

MergeSort - Speedup - Perturbed - comp_large (MPI, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
0
50
100
150
200
250
300
350
400
2^2
2^4
2^6
2^8
2^{10}

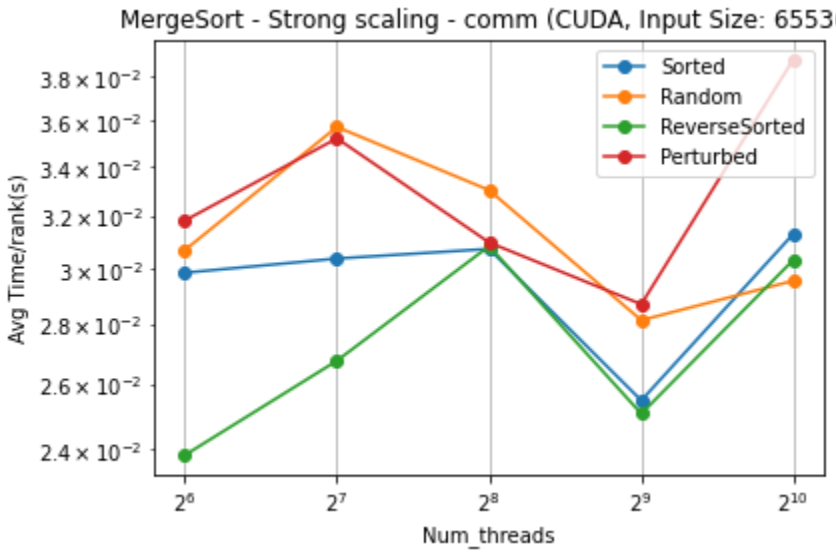


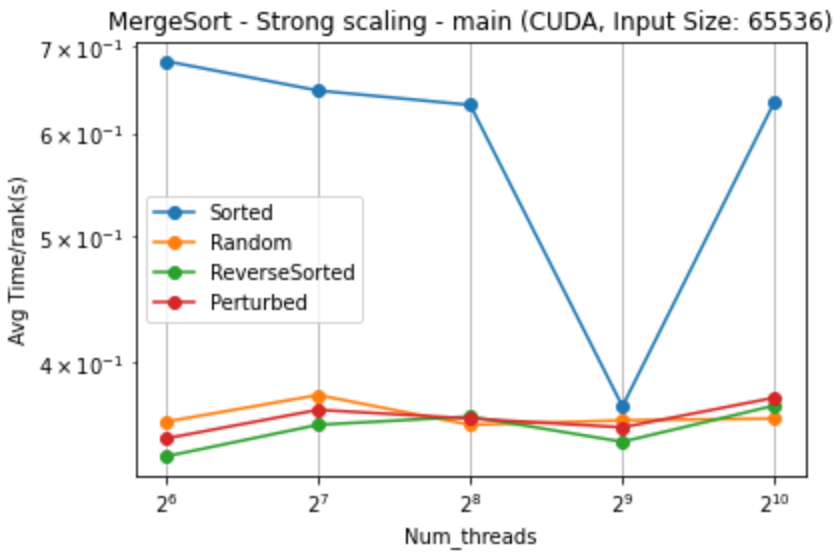


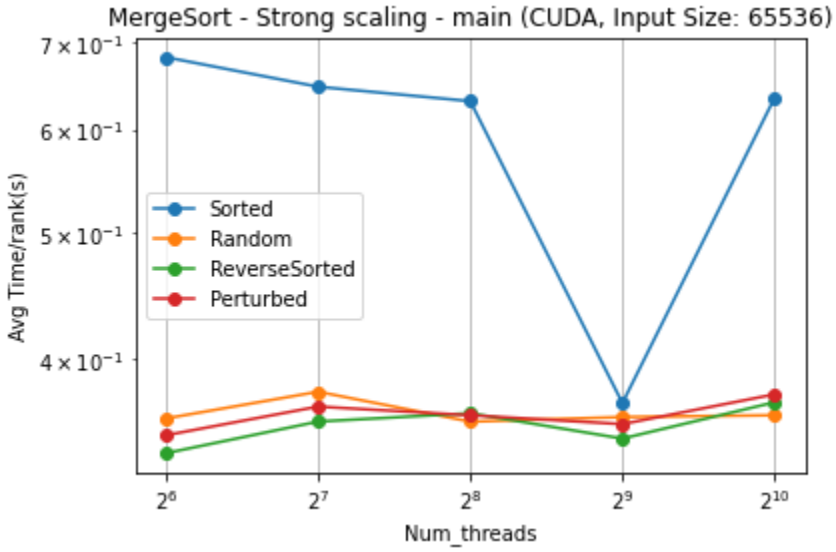
CUDA:

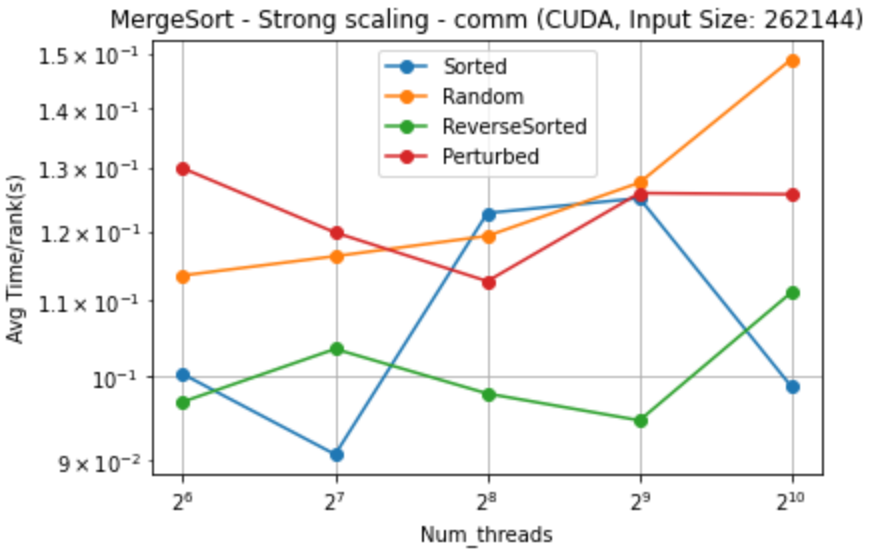
Strong Scaling:

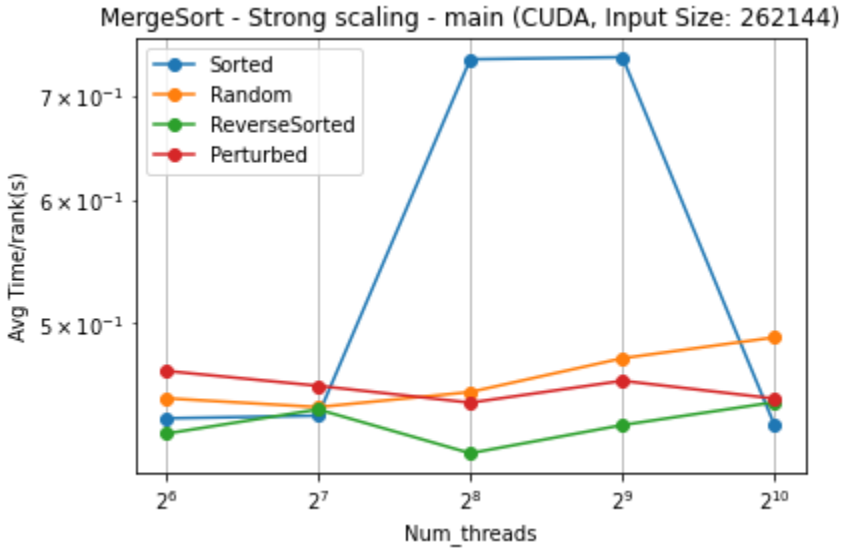

MergeSort - Strong scaling - comp_large (CUDA, Input Size: 65536)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_threads

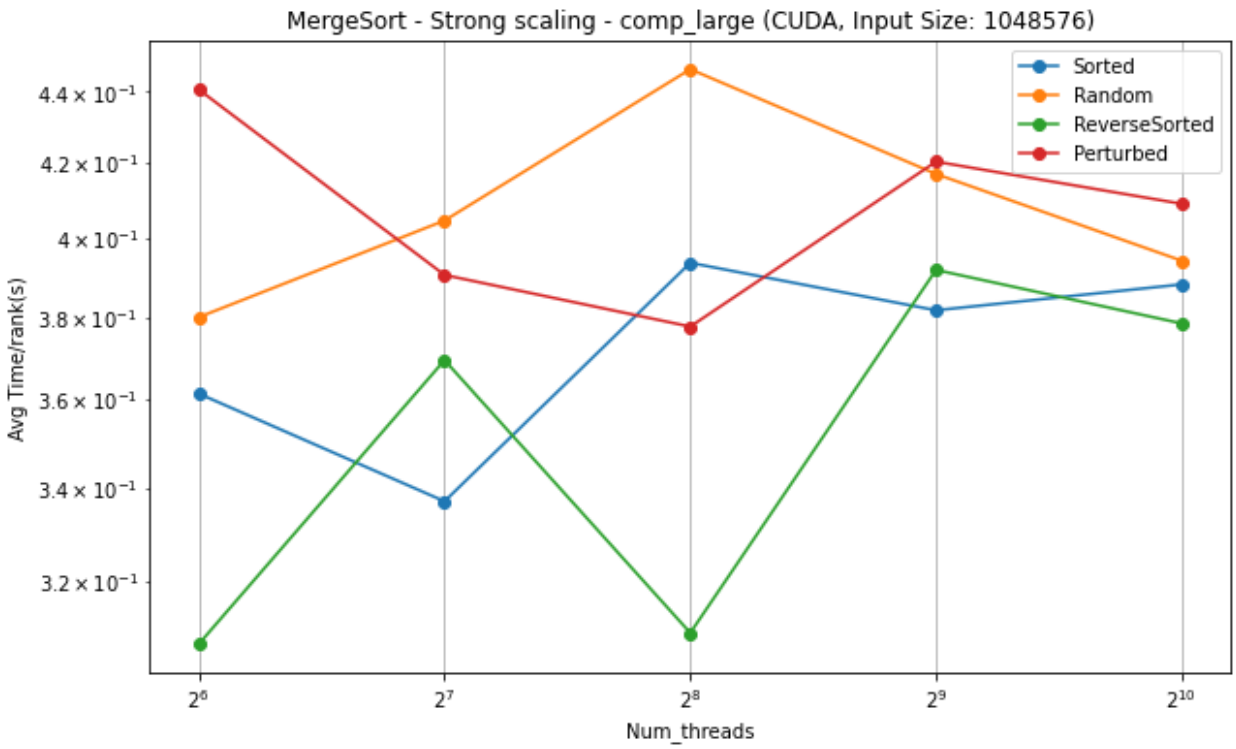

MergeSort - Strong scaling - comm (CUDA, Input Size: 65536)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_threads

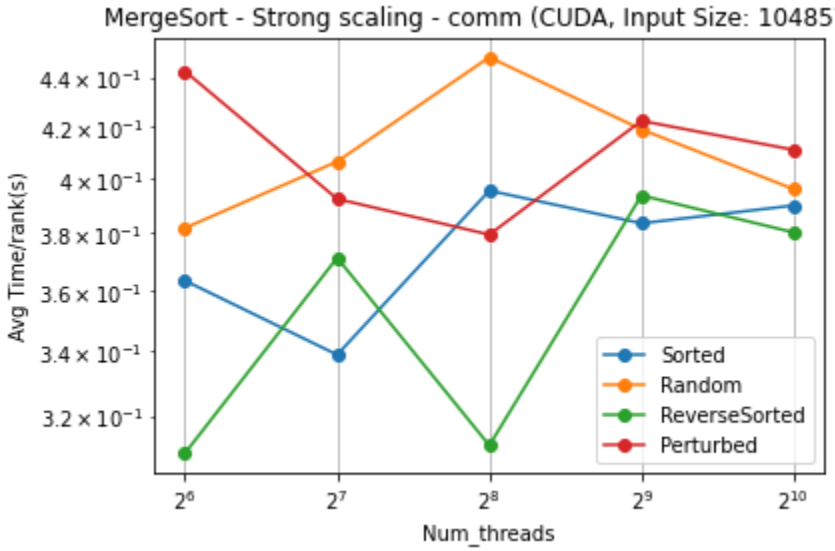

MergeSort - Strong scaling - main (CUDA, Input Size: 65536)
Avg Time/rank(s)
Num_threads
Sorted
Random
ReverseSorted
Perturbed

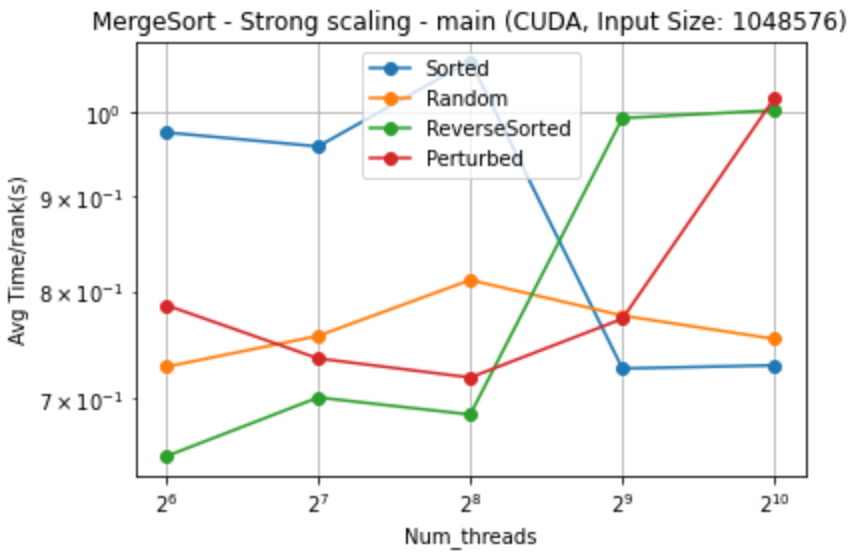

MergeSort - Strong scaling - main (CUDA, Input Size: 65536)
Avg Time/rank(s)
Num_threads
Sorted
Random
ReverseSorted
Perturbed

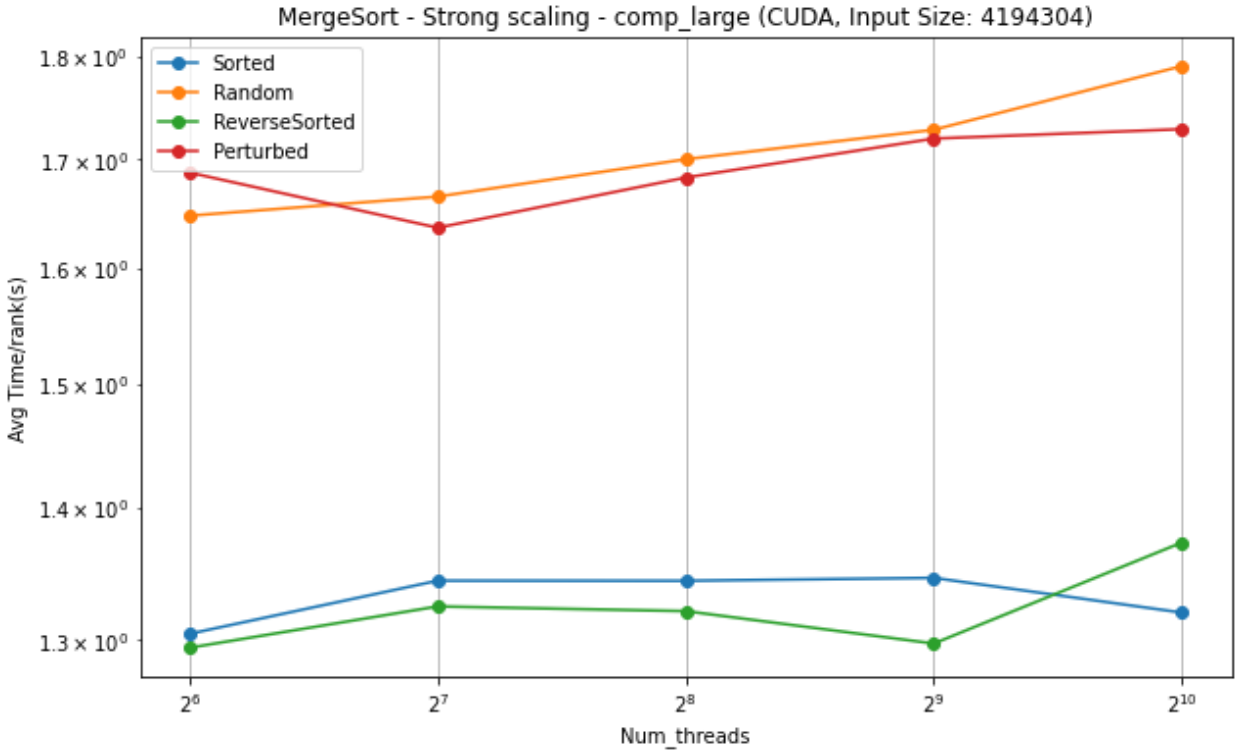

MergeSort - Strong scaling - comm (CUDA, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_threads

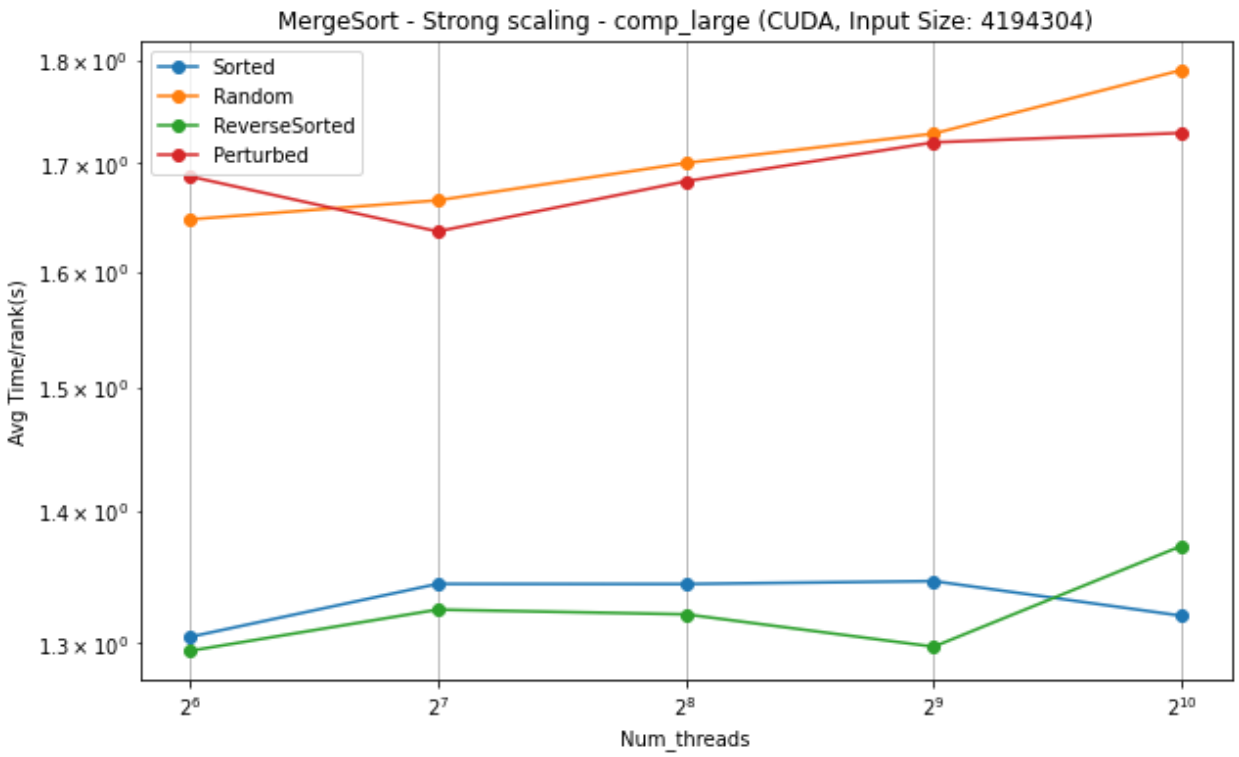

MergeSort - Strong scaling - main (CUDA, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_threads

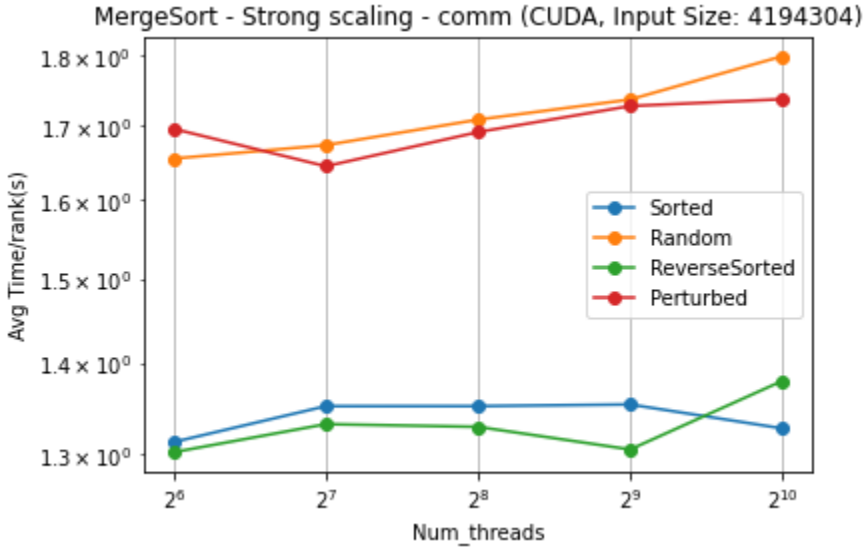

MergeSort - Strong scaling - comp_large (CUDA, Input Size: 1048576)
Avg Time/rank(s)
4.4 × 10⁻¹
4.2 × 10⁻¹
4 × 10⁻¹
3.8 × 10⁻¹
3.6 × 10⁻¹
3.4 × 10⁻¹
3.2 × 10⁻¹
2⁶
2⁷
2⁸
2⁹
2¹⁰
Num_threads
Sorted
Random
ReverseSorted
Perturbed

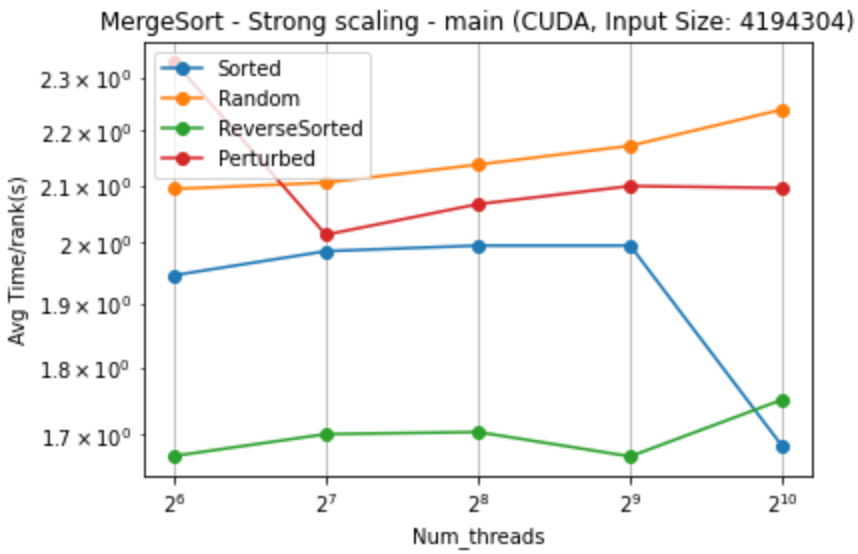

MergeSort - Strong scaling - comm (CUDA, Input Size: 1048576)
Avg Time/rank(s)
4.4 × 10⁻¹
4.2 × 10⁻¹
4 × 10⁻¹
3.8 × 10⁻¹
3.6 × 10⁻¹
3.4 × 10⁻¹
3.2 × 10⁻¹
2⁶
2⁷
2⁸
2⁹
2¹⁰
Num_threads
Sorted
Random
ReverseSorted
Perturbed

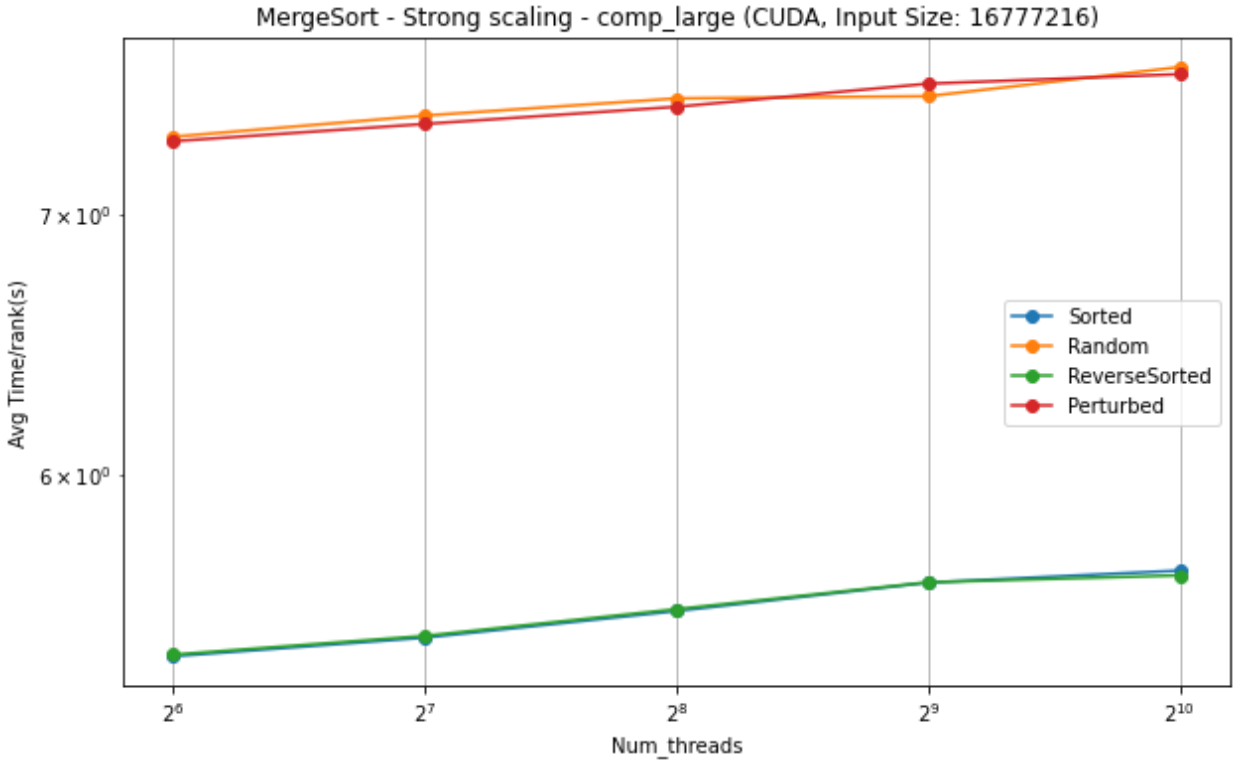

MergeSort - Strong scaling - main (CUDA, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
10^0
9×10^{-1}
8×10^{-1}
7×10^{-1}
Avg Time/rank(s)
2^6
2^7
2^8
2^9
2^{10}
Num_threads

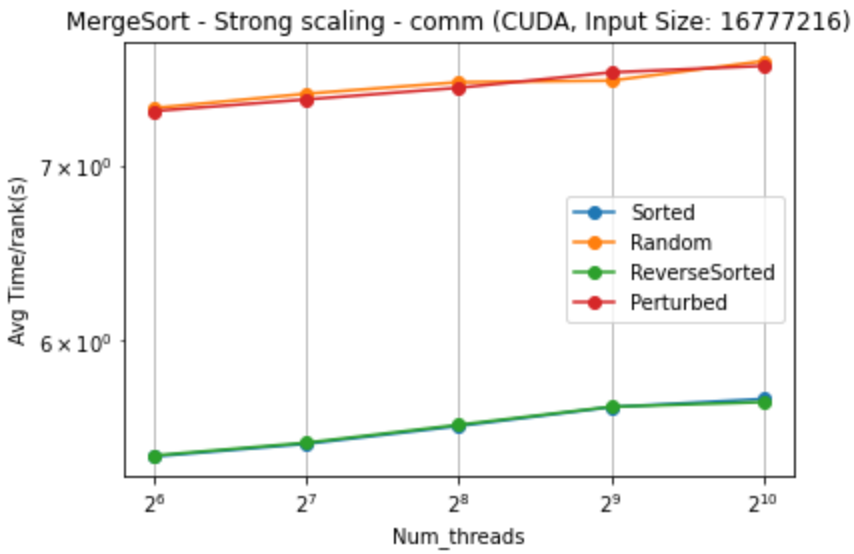

MergeSort - Strong scaling - comp_large (CUDA, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
1.8×10^0
1.7×10^0
1.6×10^0
1.5×10^0
1.4×10^0
1.3×10^0
Avg Time/rank(s)
2^6
2^7
2^8
2^9
2^{10}
Num_threads

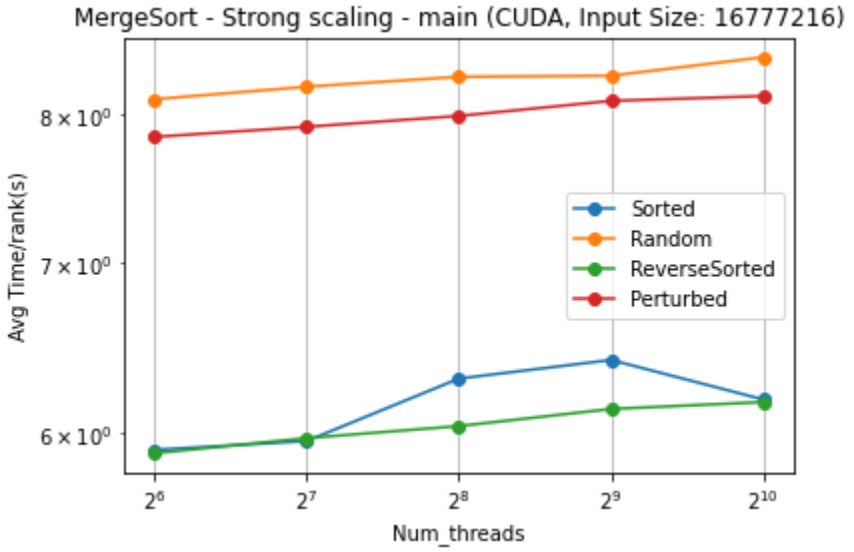

MergeSort - Strong scaling - comp_large (CUDA, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
1.8 × 10^0
1.7 × 10^0
1.6 × 10^0
1.5 × 10^0
1.4 × 10^0
1.3 × 10^0
2^6
2^7
2^8
2^9
2^10
Num_threads

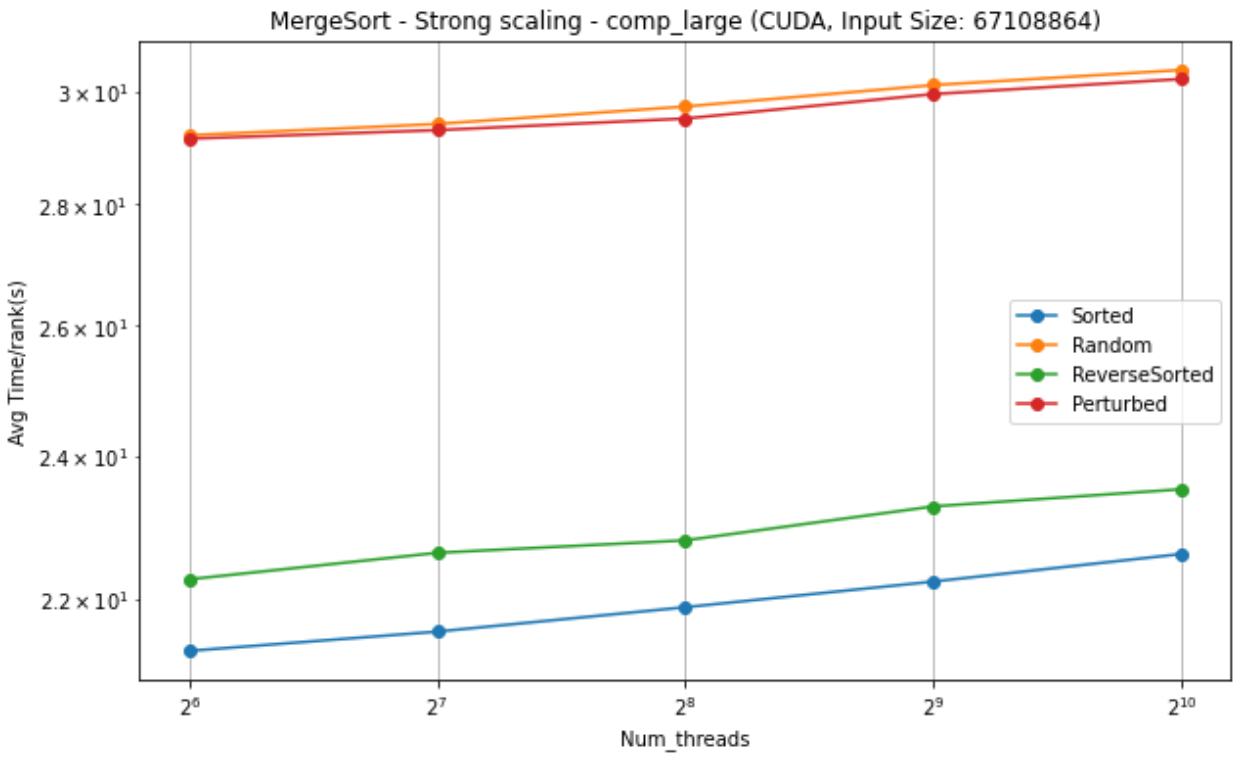

MergeSort - Strong scaling - comm (CUDA, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
1.8 × 10^0
1.7 × 10^0
1.6 × 10^0
1.5 × 10^0
1.4 × 10^0
1.3 × 10^0
2^6
2^7
2^8
2^9
2^10
Num_threads

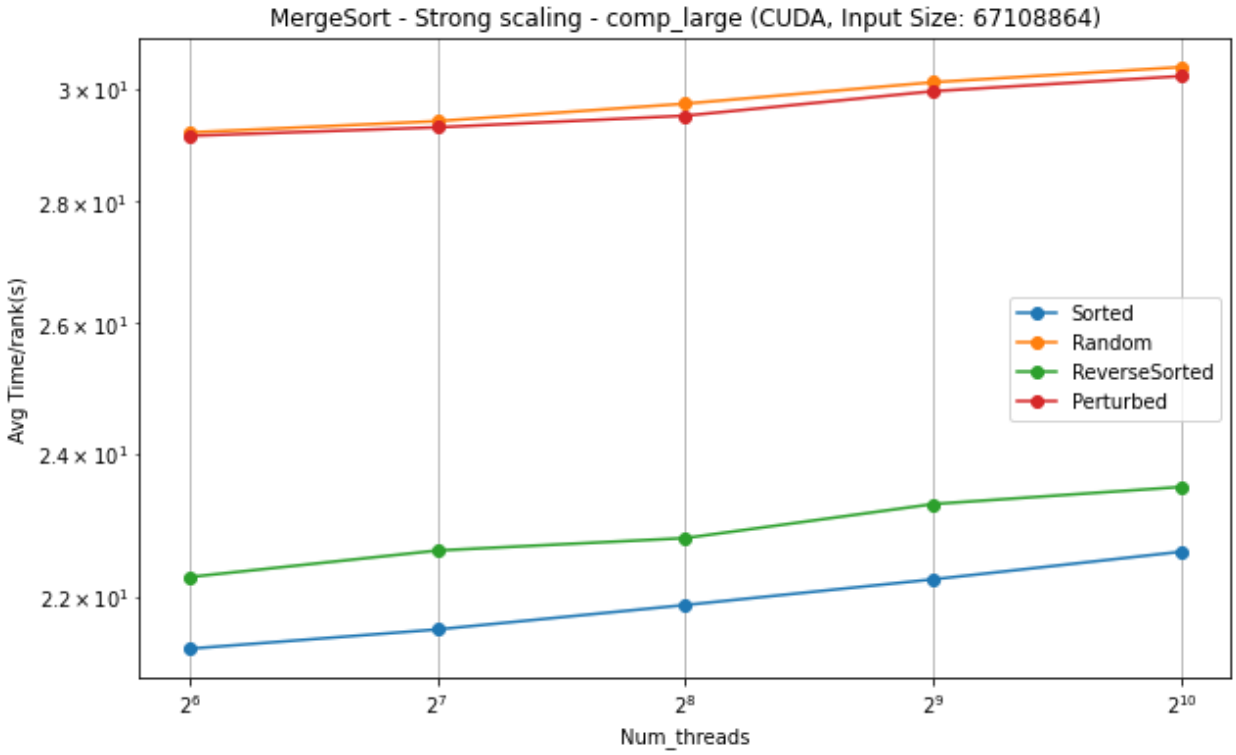

MergeSort - Strong scaling - main (CUDA, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_threads

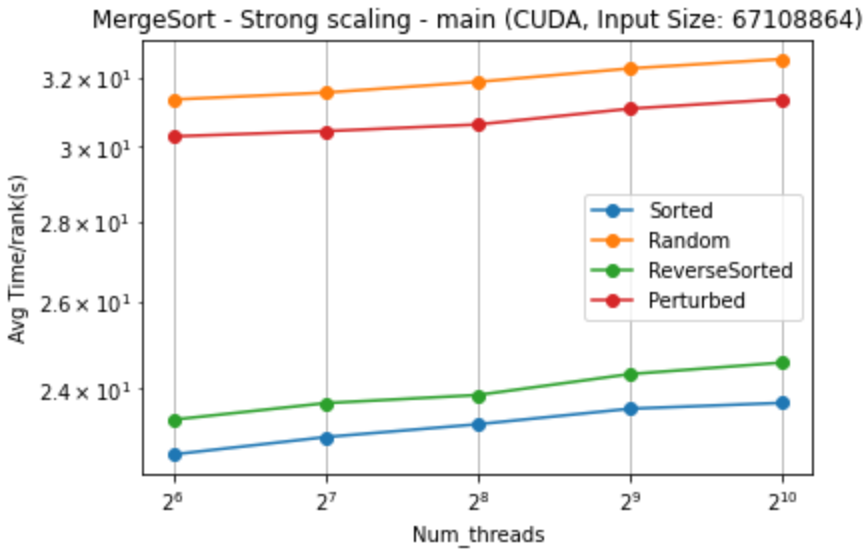

MergeSort - Strong scaling - comp_large (CUDA, Input Size: 16777216)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_threads

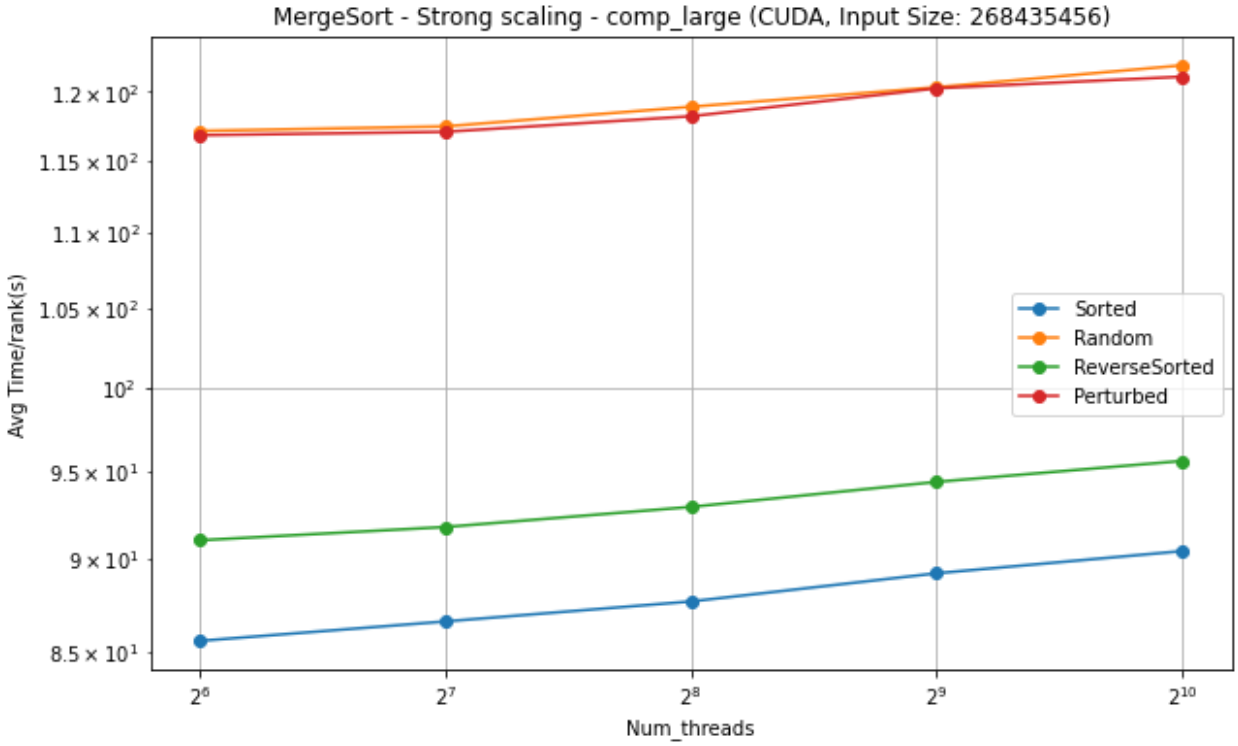

MergeSort - Strong scaling - comm (CUDA, Input Size: 16777216)
Avg Time/rank(s)
7×10^0
6×10^0
Sorted
Random
ReverseSorted
Perturbed
2^6
2^7
2^8
2^9
2^{10}
Num_threads

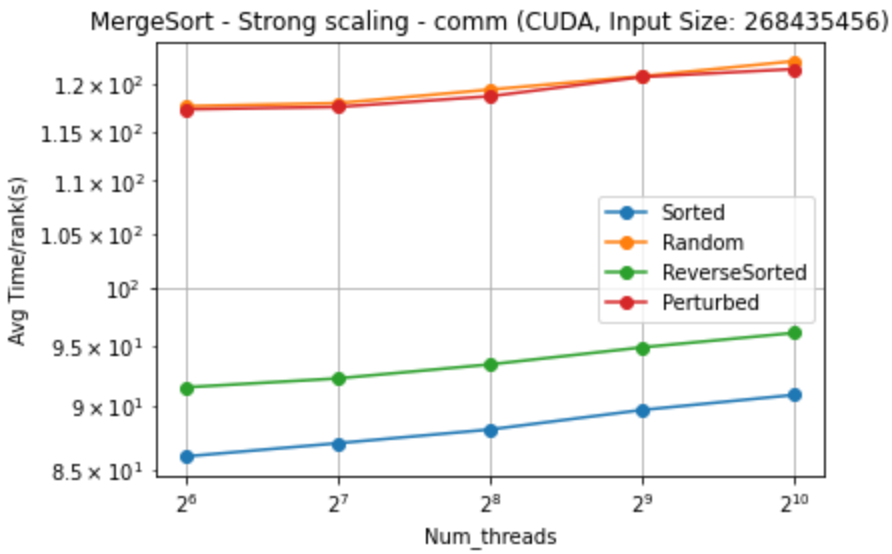

MergeSort - Strong scaling - main (CUDA, Input Size: 16777216)
Avg Time/rank(s)
8×10^0
7×10^0
6×10^0
Sorted
Random
ReverseSorted
Perturbed
2^6
2^7
2^8
2^9
2^{10}
Num_threads

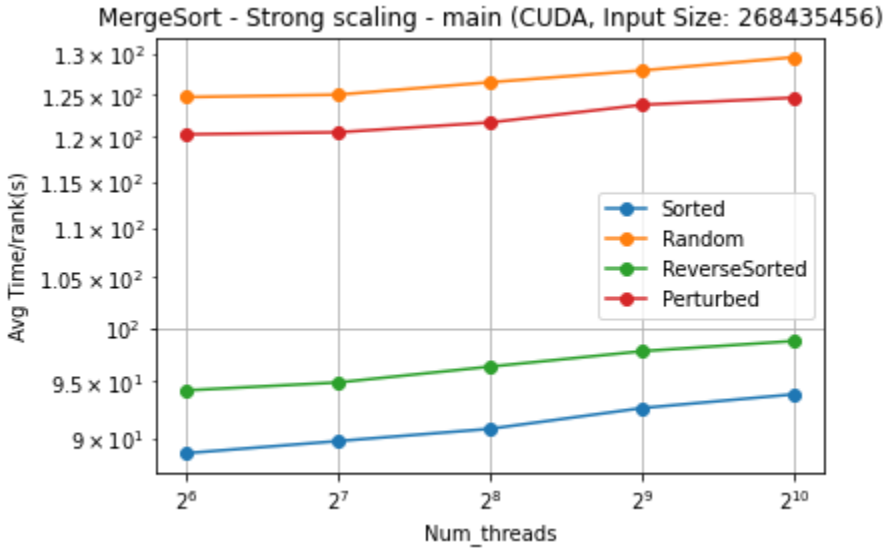

MergeSort - Strong scaling - comp_large (CUDA, Input Size: 67108864)
Avg Time/rank(s)
Num_threads
Sorted
Random
ReverseSorted
Perturbed


MergeSort - Strong scaling - comp_large (CUDA, Input Size: 67108864)
Avg Time/rank(s)
Num_threads
Sorted
Random
ReverseSorted
Perturbed

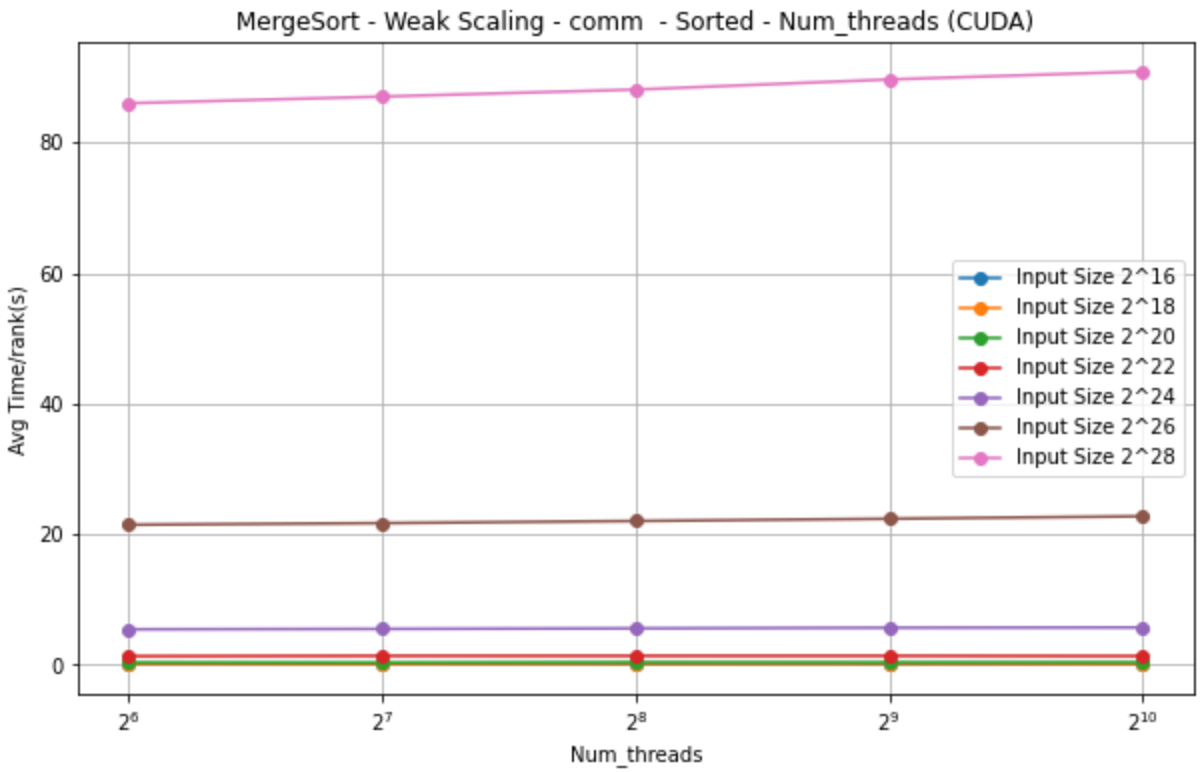

MergeSort - Strong scaling - main (CUDA, Input Size: 67108864)
Avg Time/rank(s)
Num_threads
Sorted
Random
ReverseSorted
Perturbed

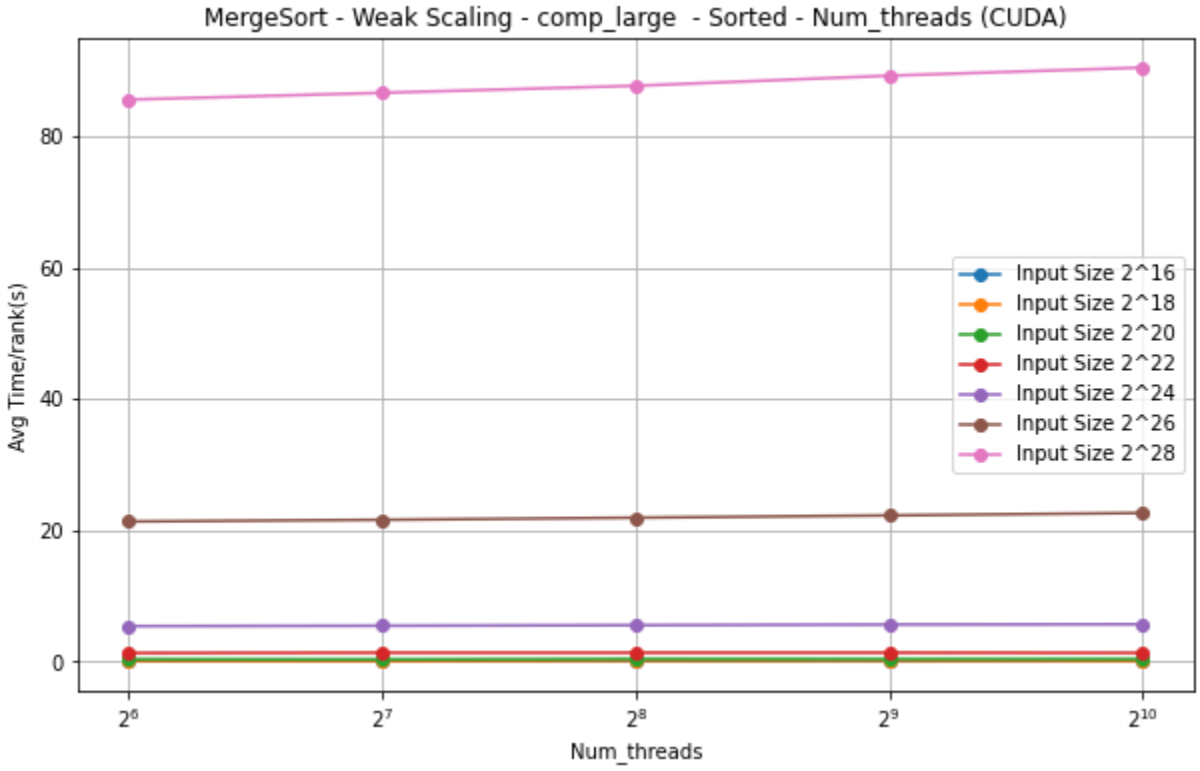

MergeSort - Strong scaling - comp_large (CUDA, Input Size: 268435456)
Avg Time/rank(s)
Num_threads
Sorted
Random
ReverseSorted
Perturbed

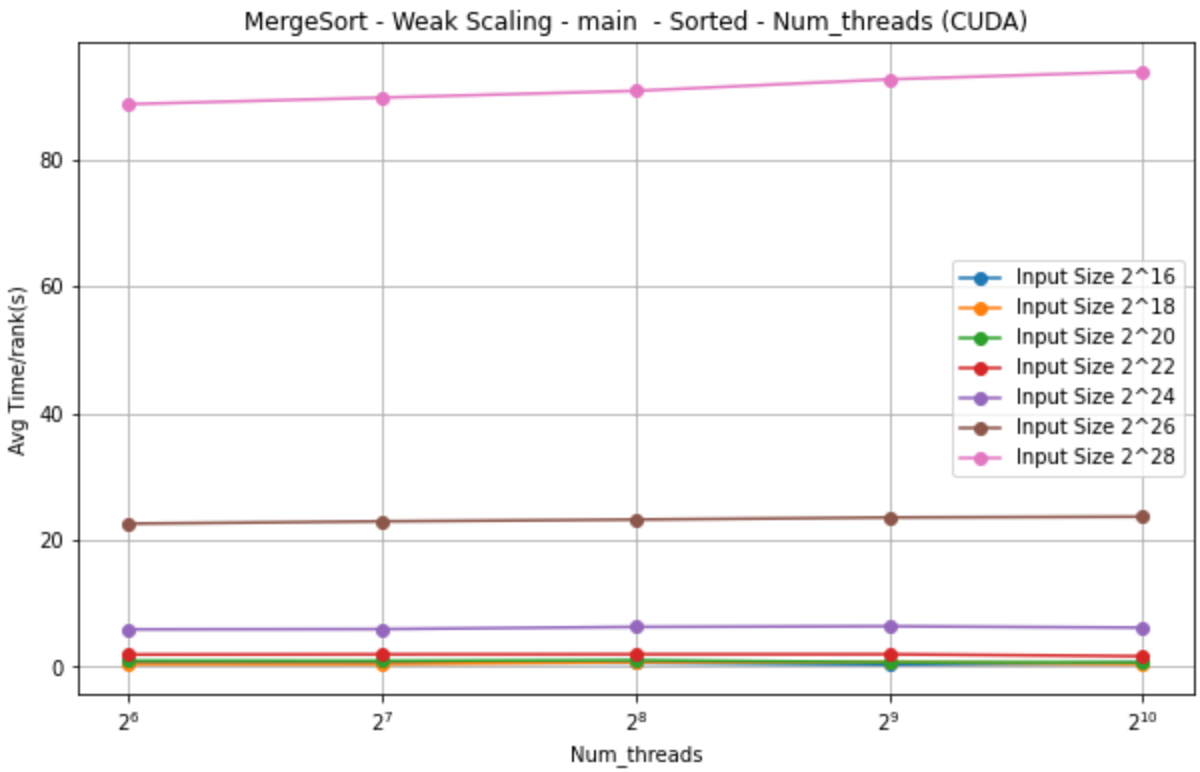


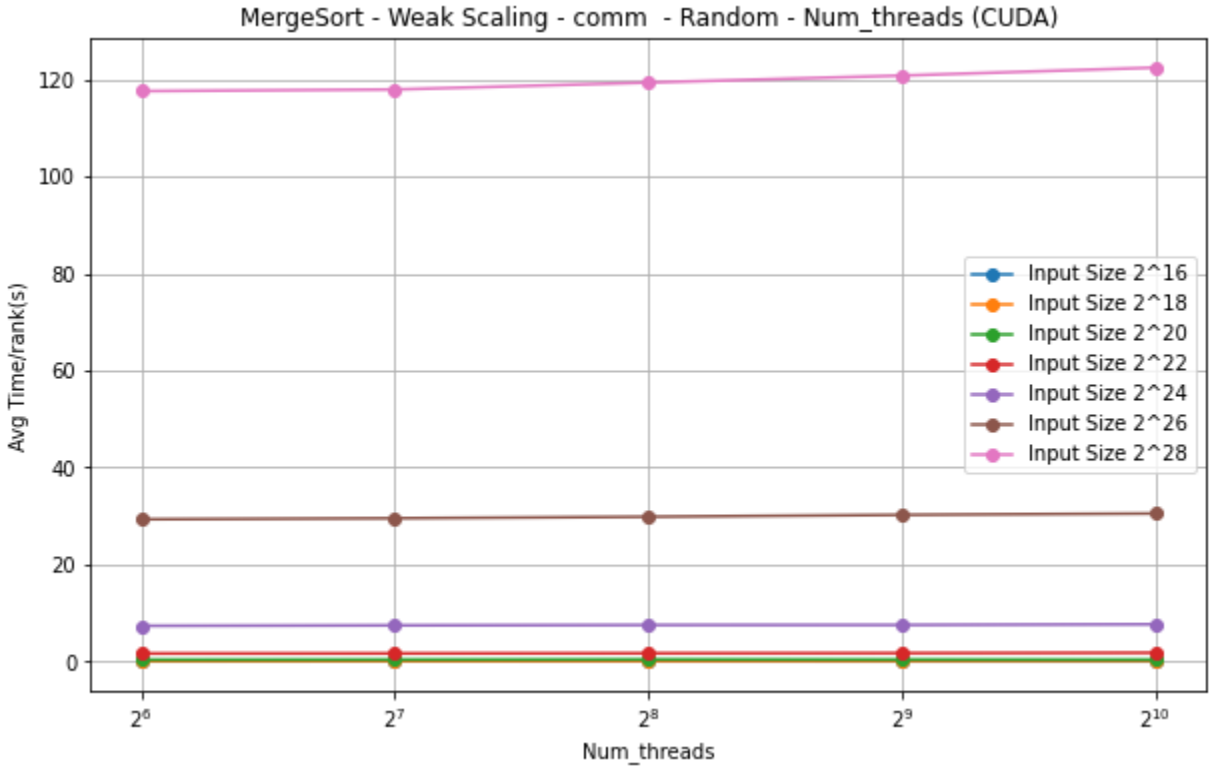


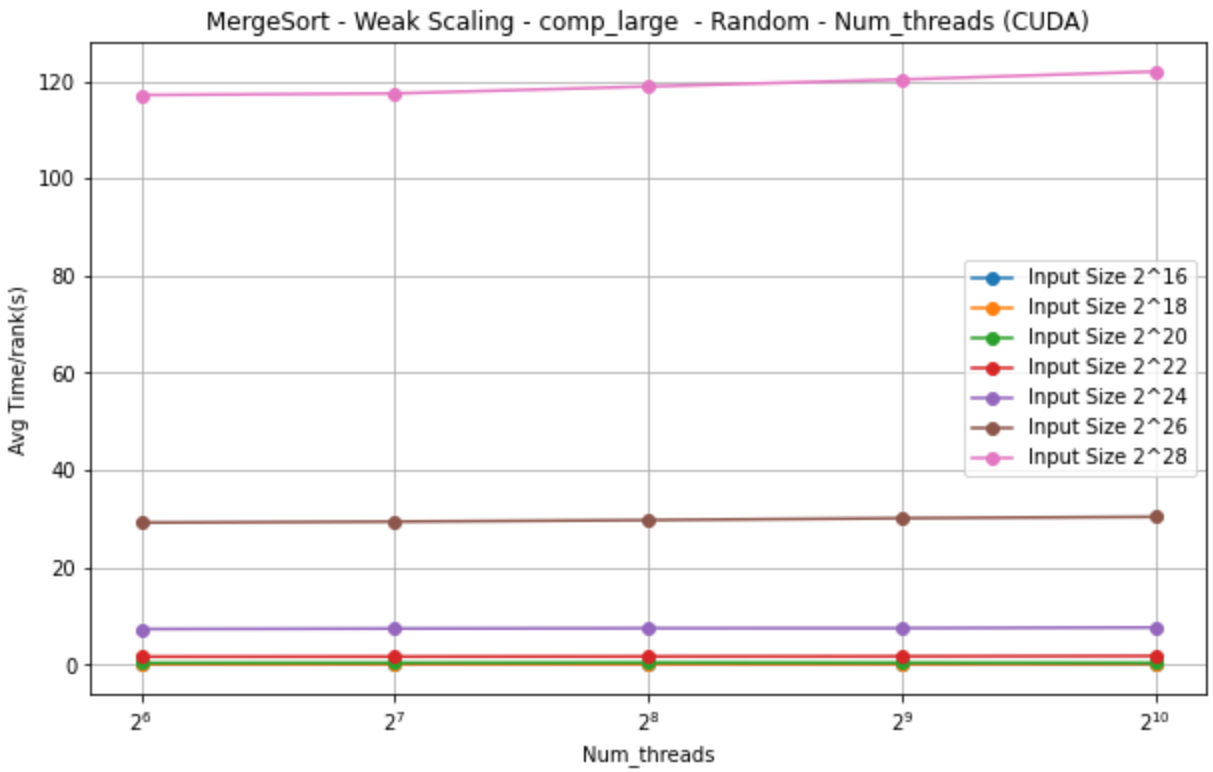
Weak Scaling:

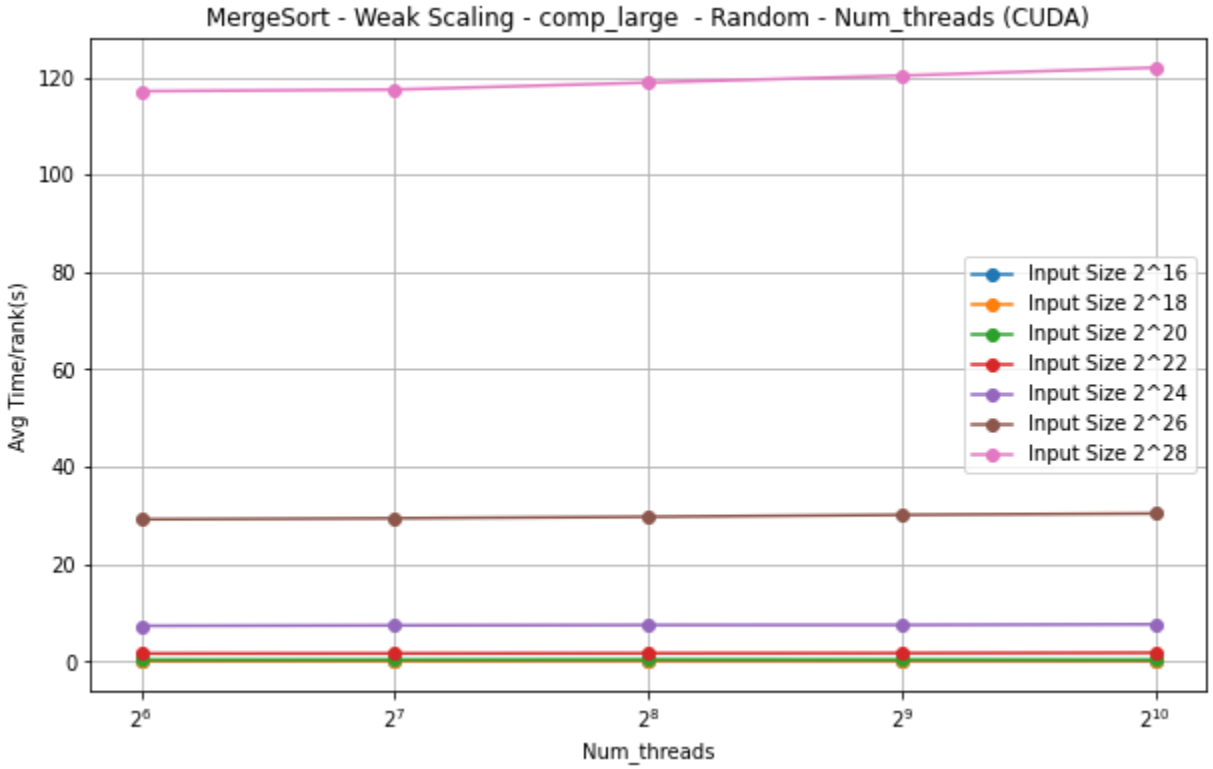

MergeSort - Weak Scaling - comm - Sorted - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

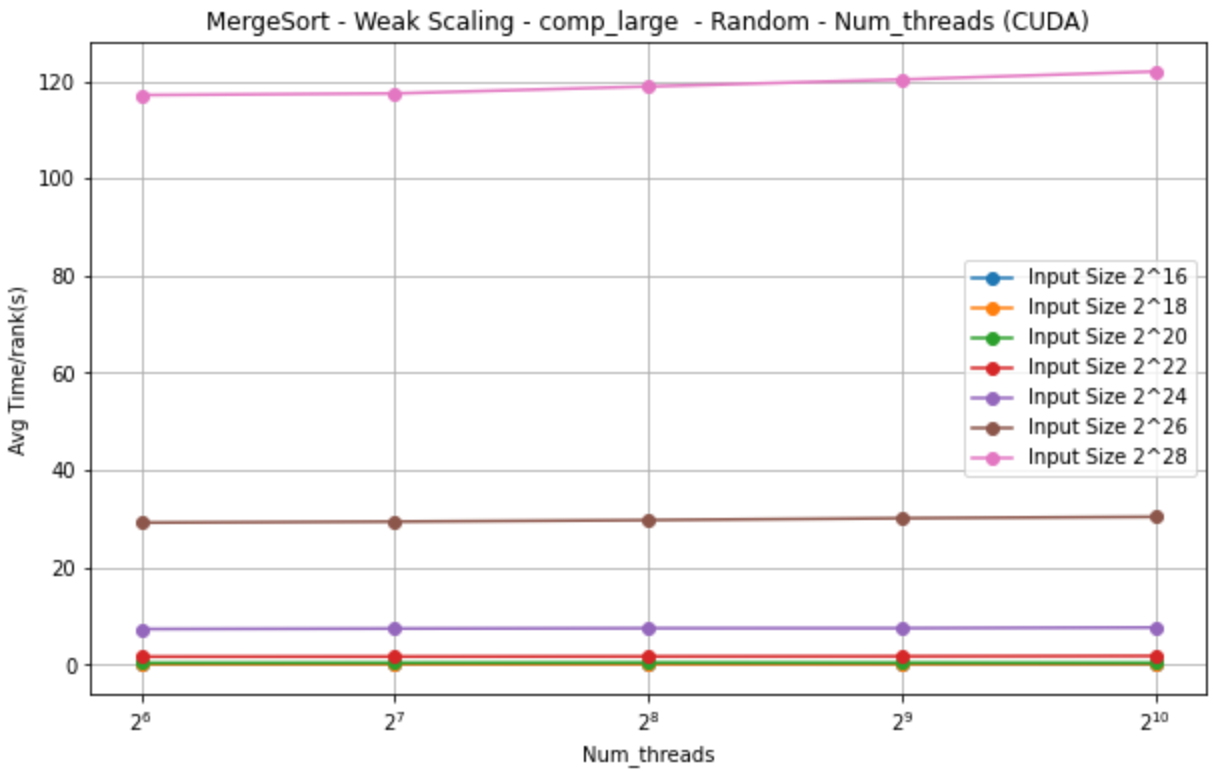

MergeSort - Weak Scaling - comp_large - Sorted - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

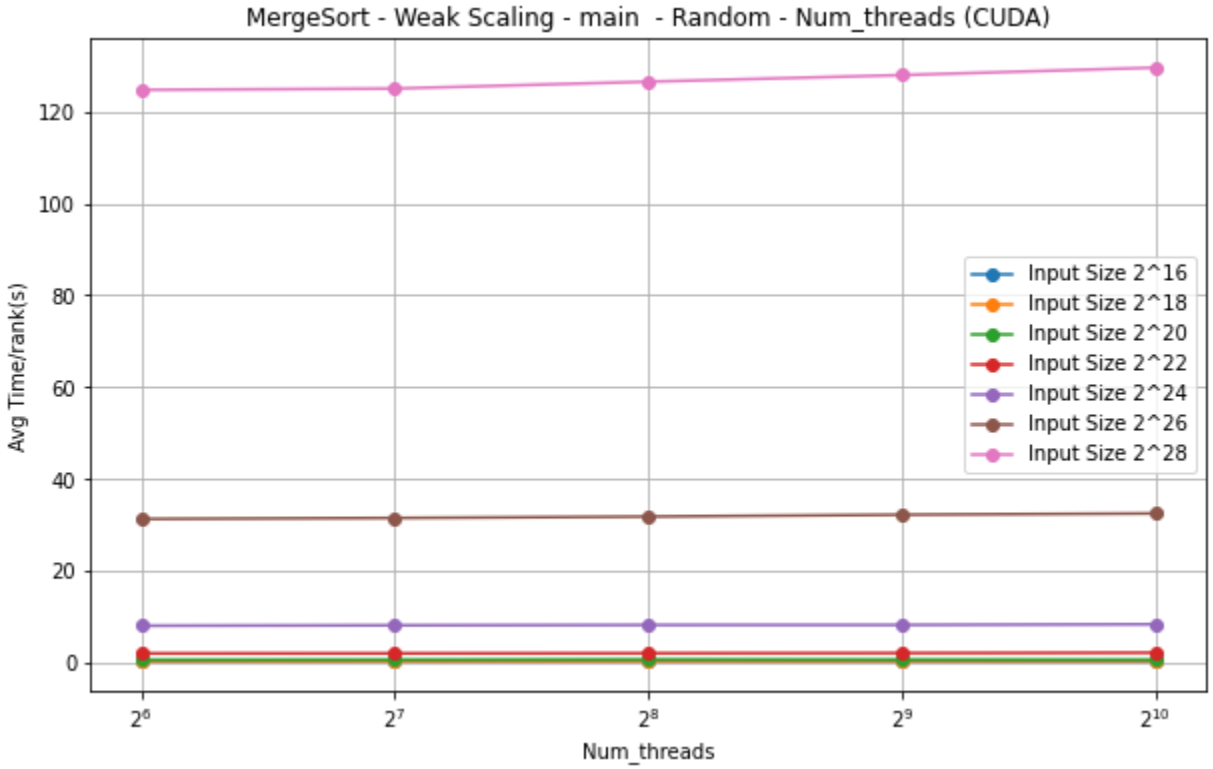

MergeSort - Weak Scaling - main - Sorted - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

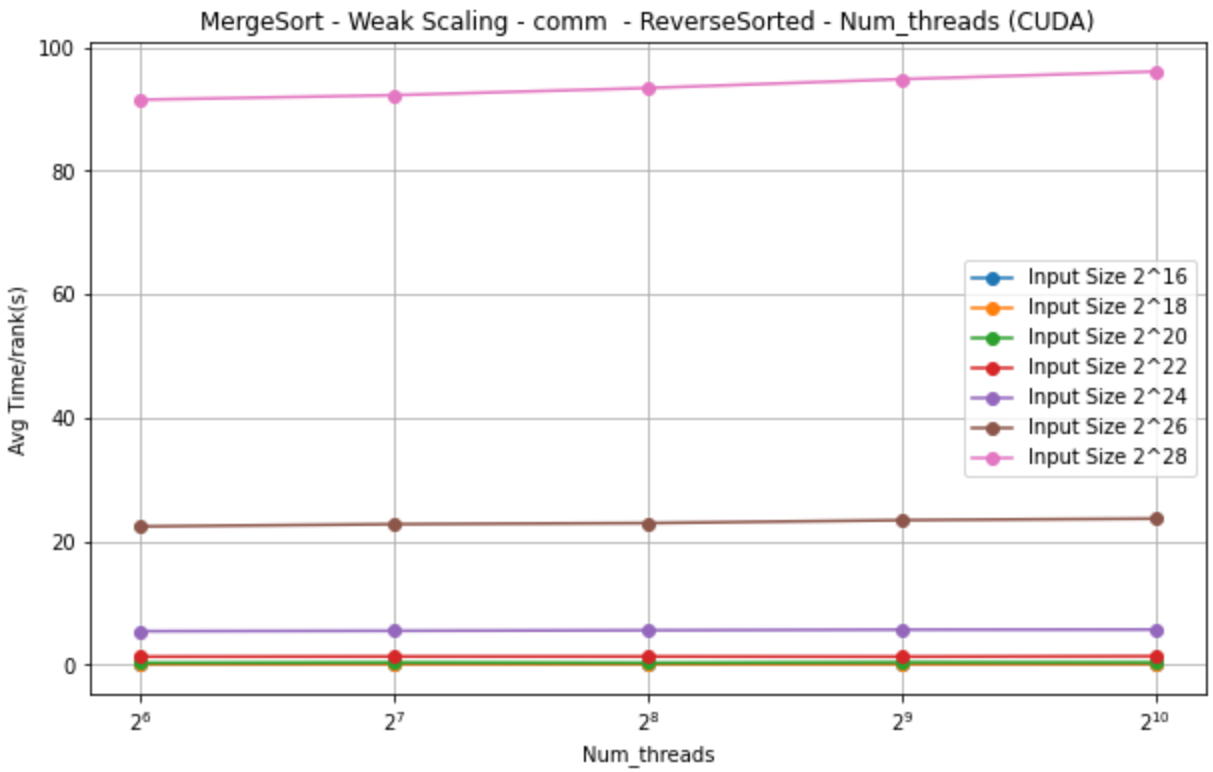

MergeSort - Weak Scaling - comm - Random - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

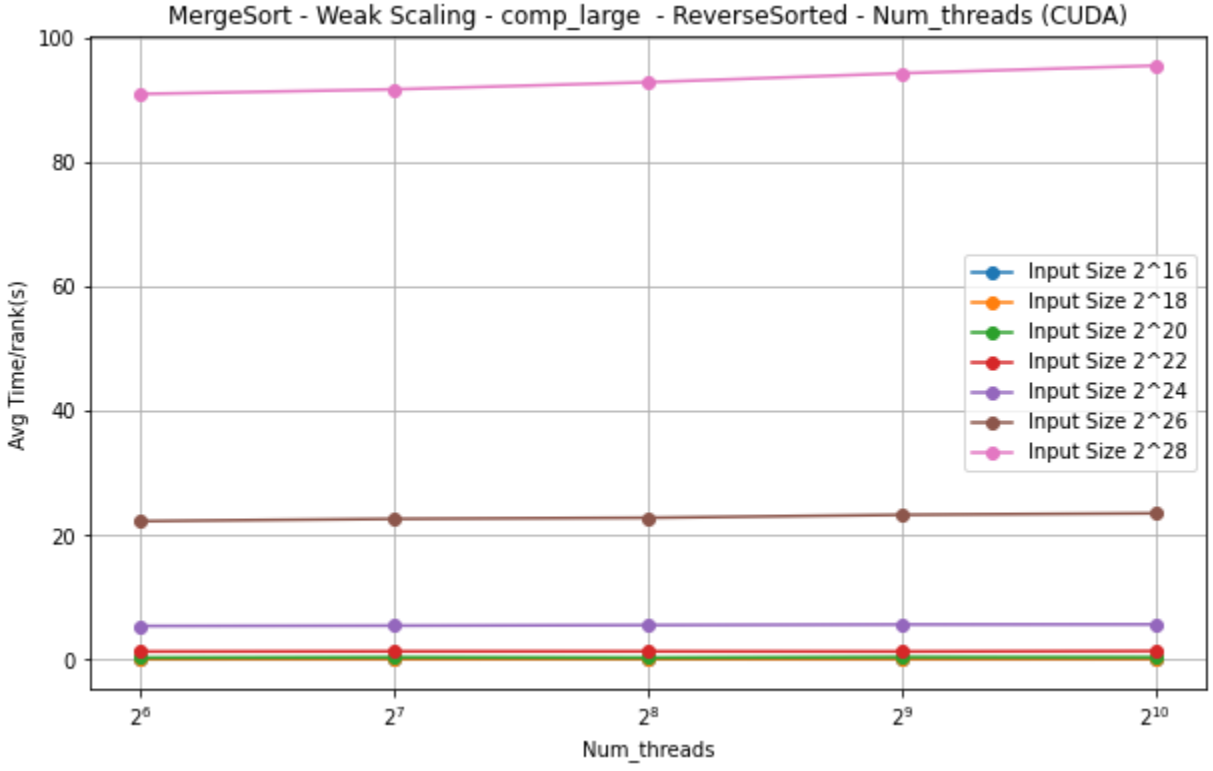

MergeSort - Weak Scaling - comp_large - Random - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

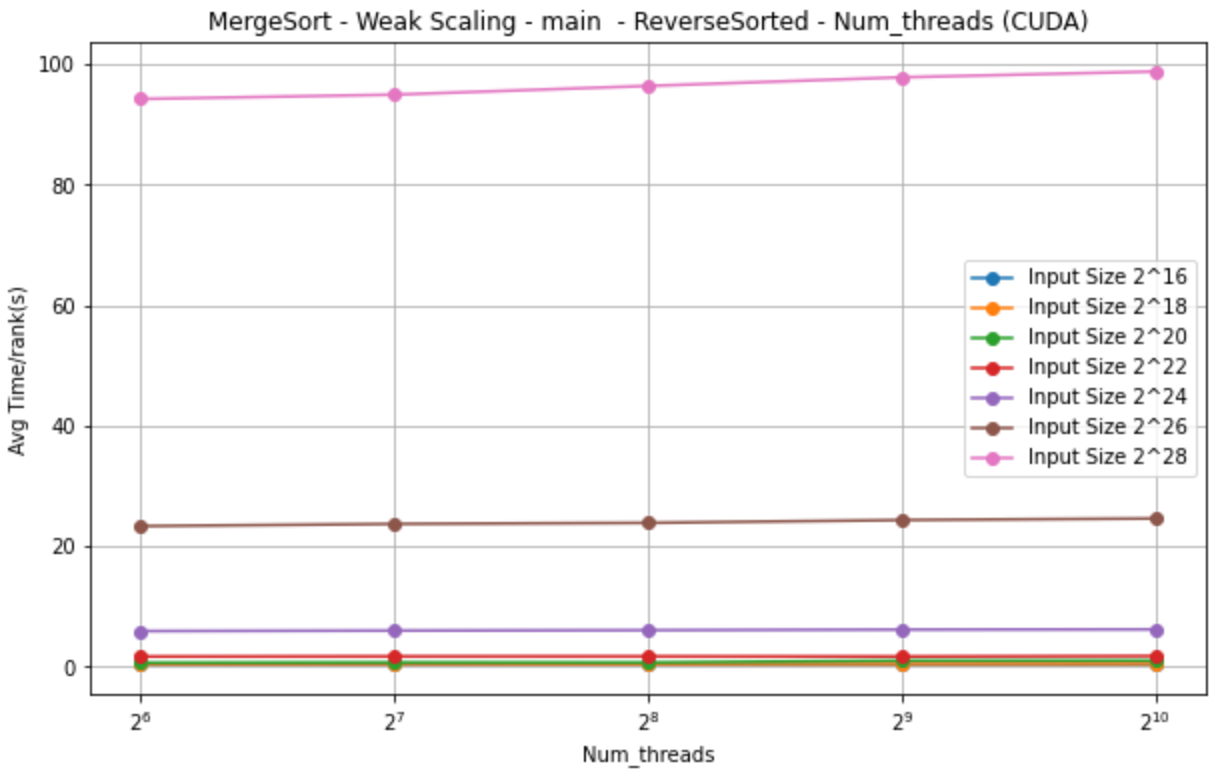

MergeSort - Weak Scaling - comp_large - Random - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

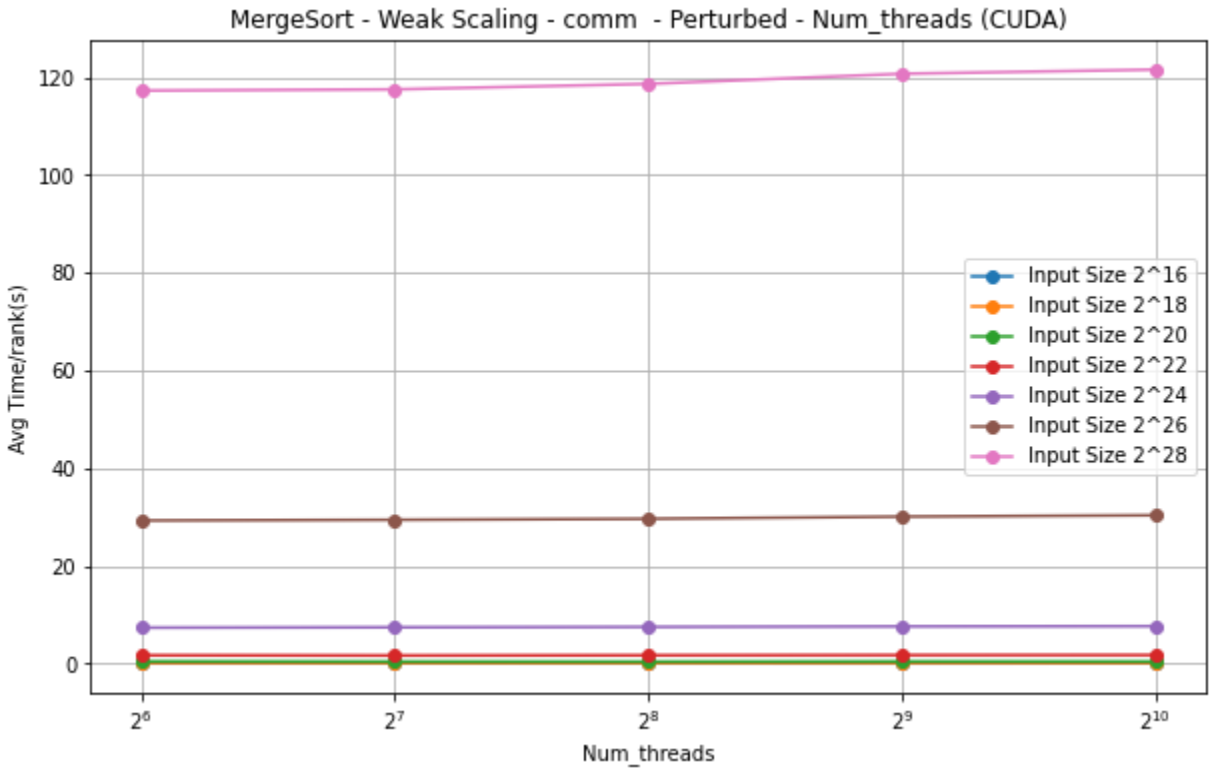

MergeSort - Weak Scaling - comp_large - Random - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

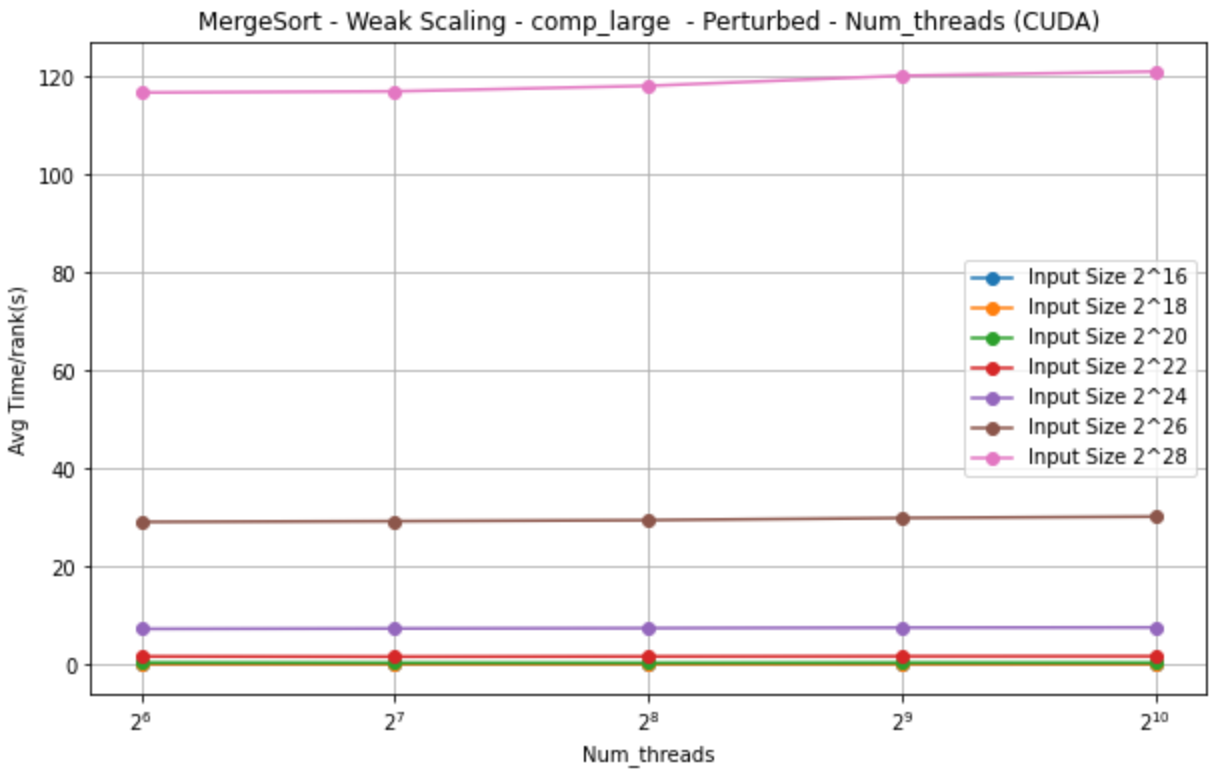

MergeSort - Weak Scaling - main - Random - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

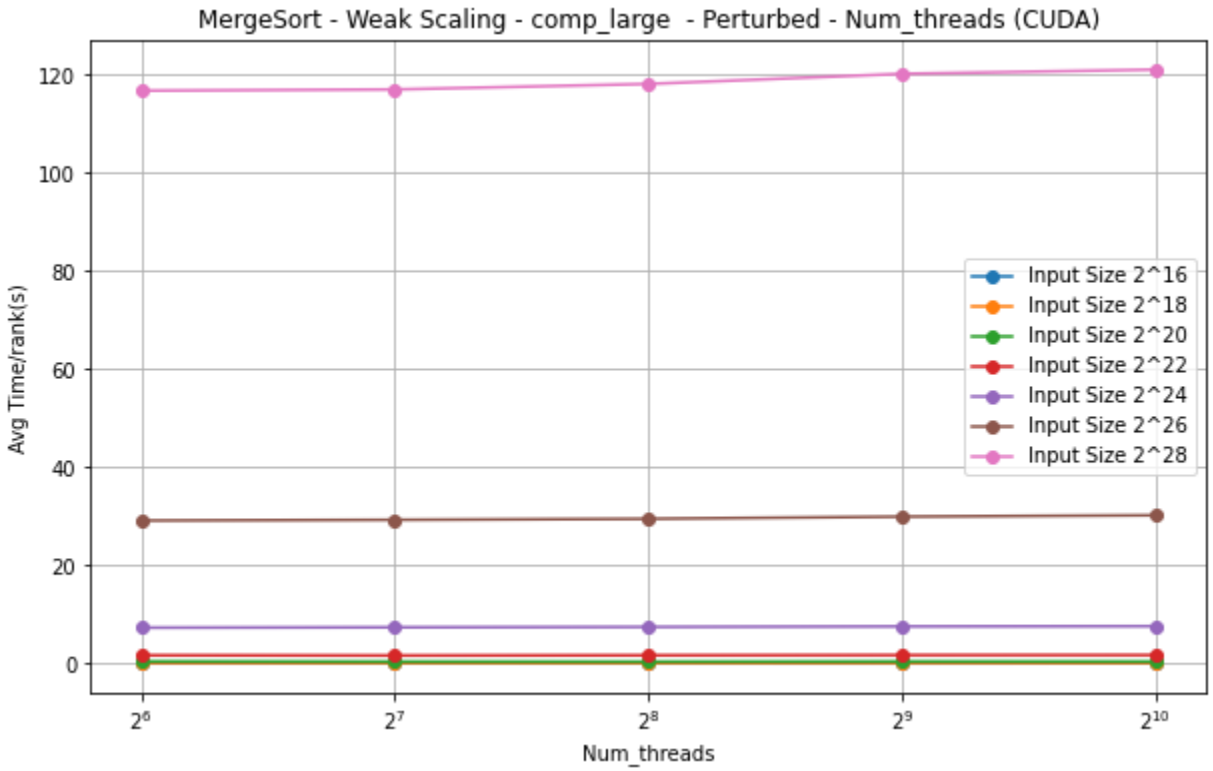

MergeSort - Weak Scaling - comm - ReverseSorted - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

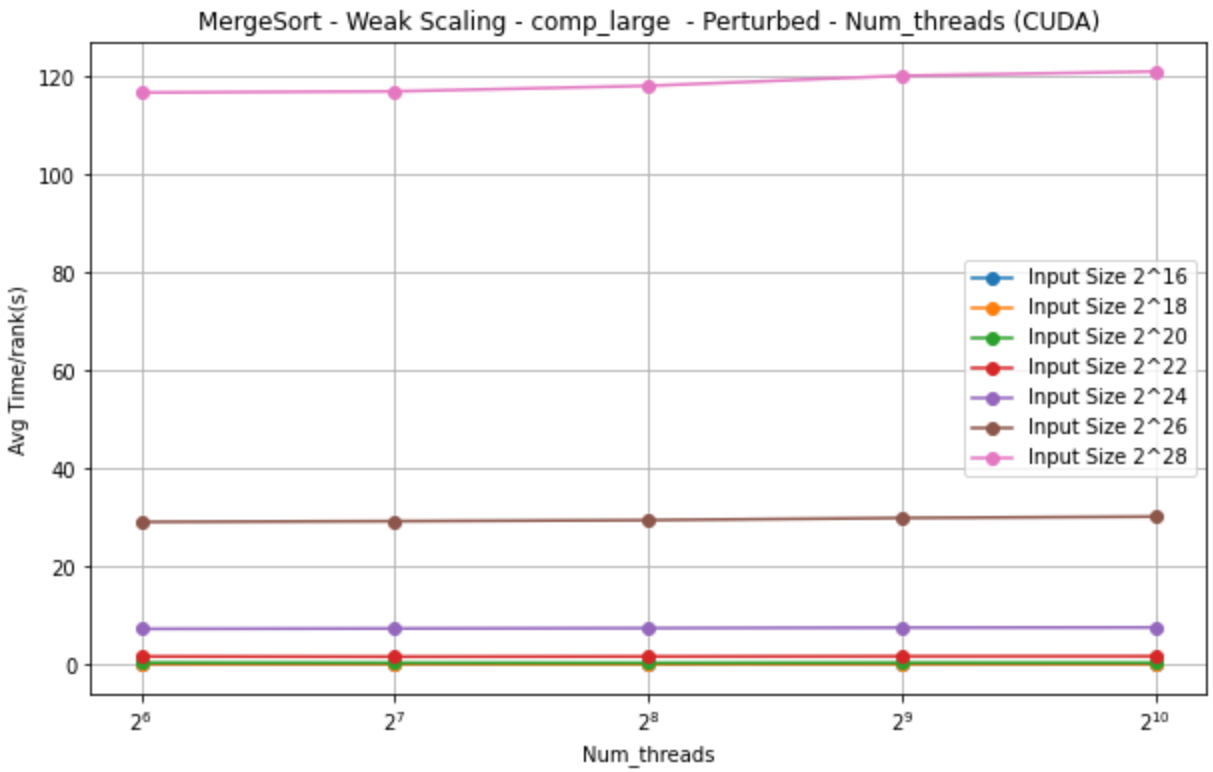

MergeSort - Weak Scaling - comp_large - ReverseSorted - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28


MergeSort - Weak Scaling - main - ReverseSorted - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

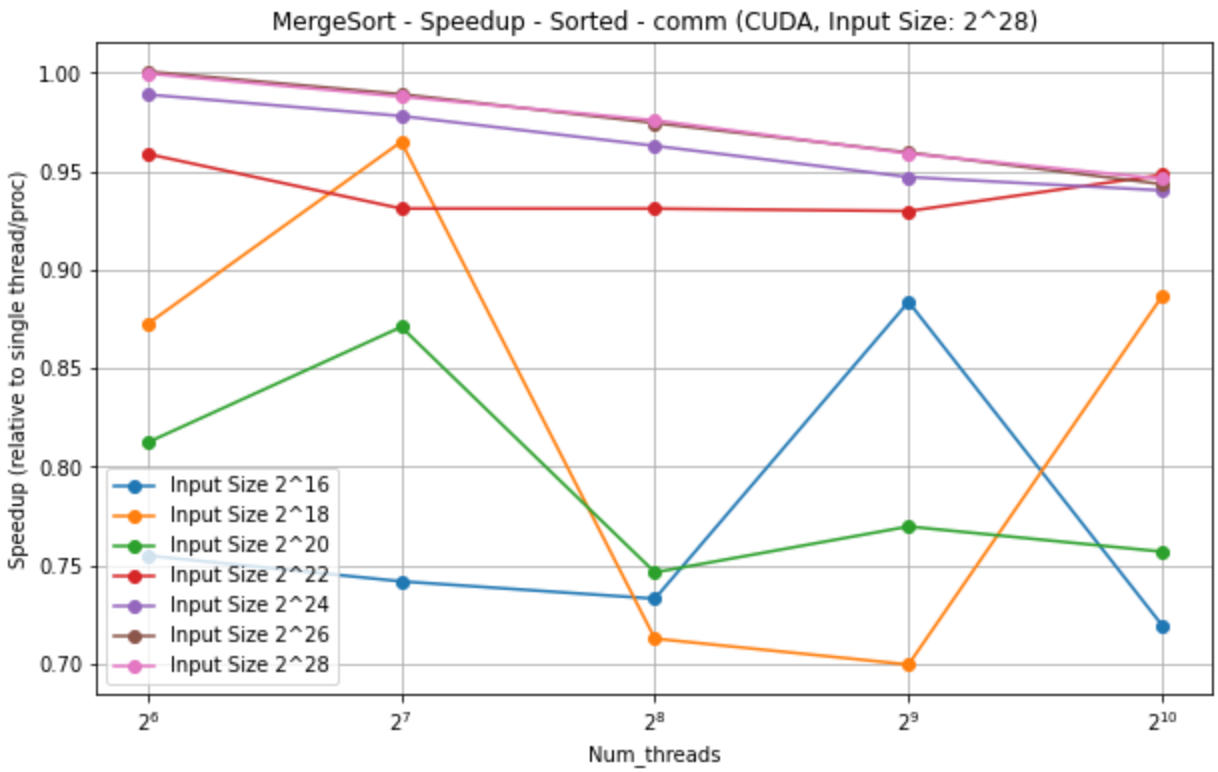

MergeSort - Weak Scaling - comm - Perturbed - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

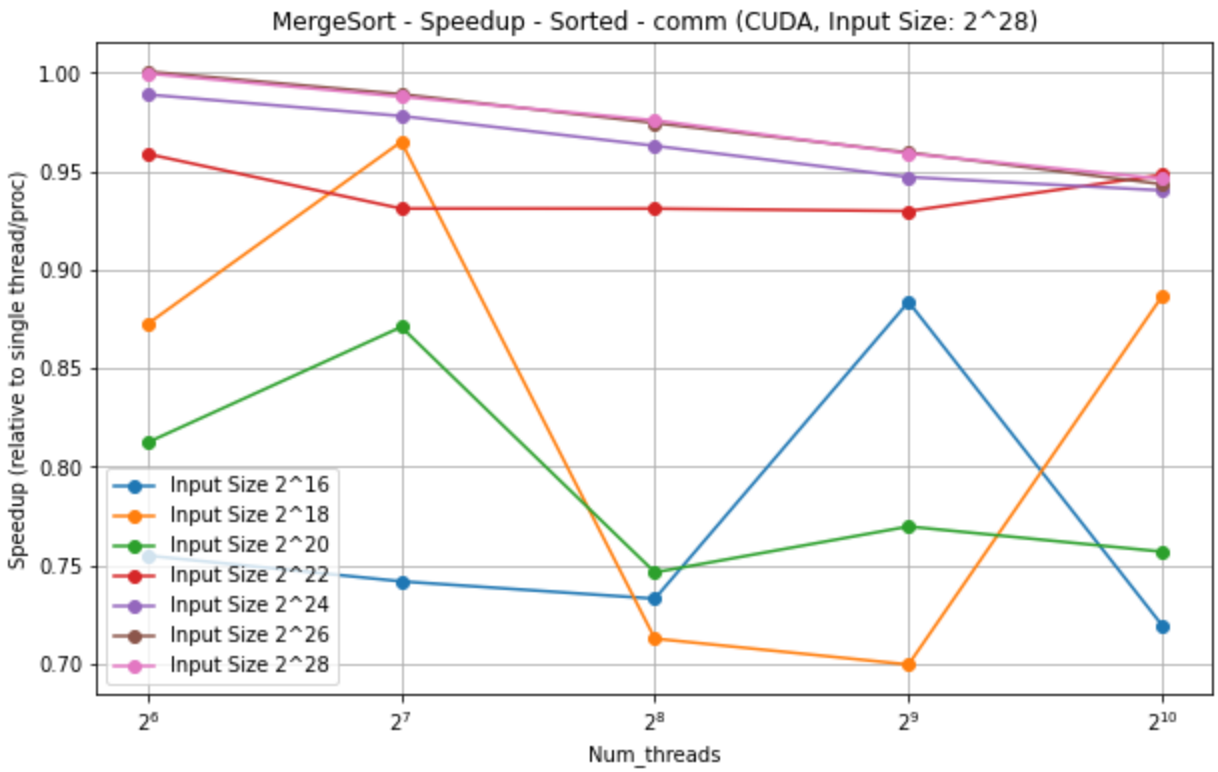

MergeSort - Weak Scaling - comp_large - Perturbed - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

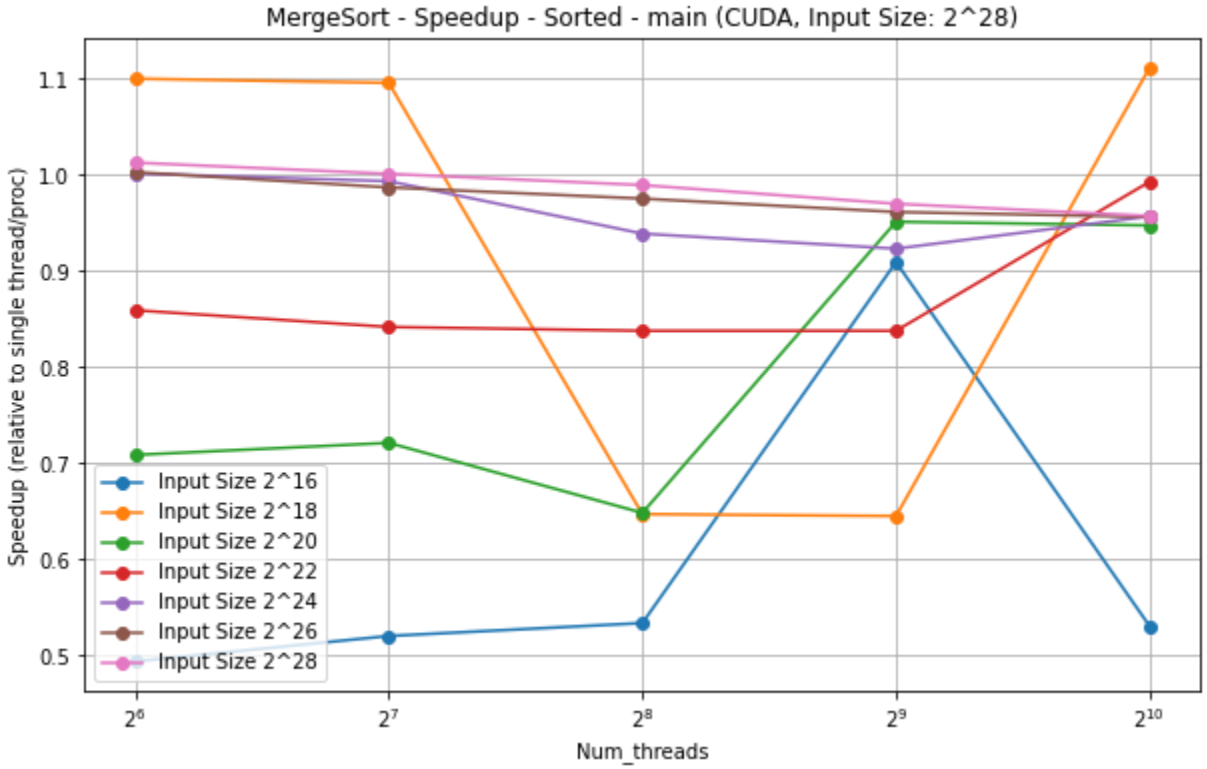

MergeSort - Weak Scaling - comp_large - Perturbed - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

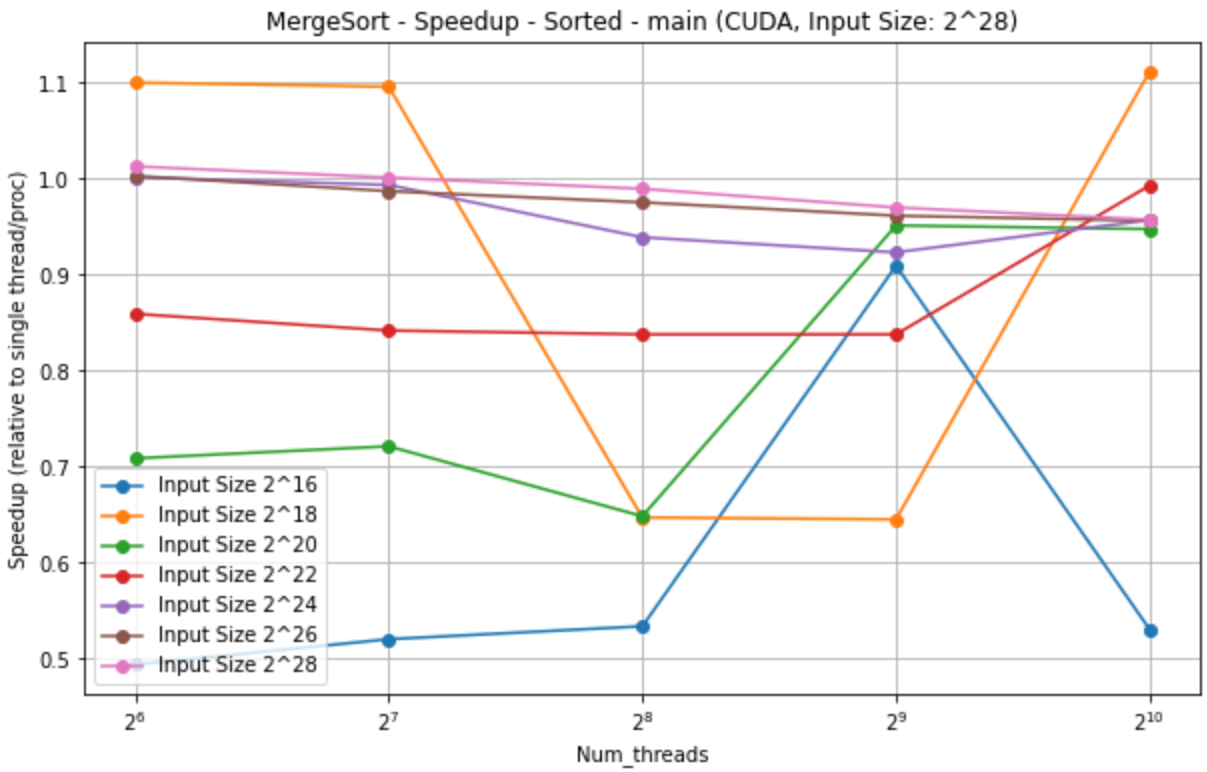

MergeSort - Weak Scaling - comp_large - Perturbed - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

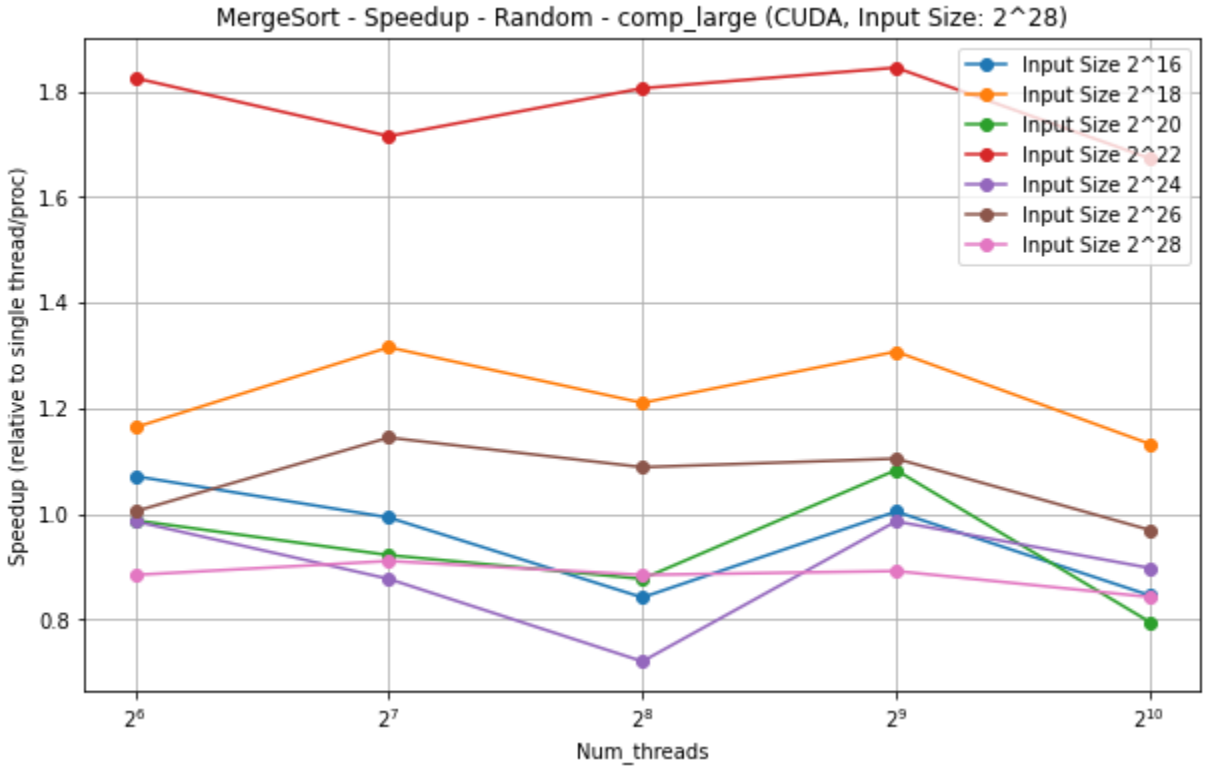
Speedup:

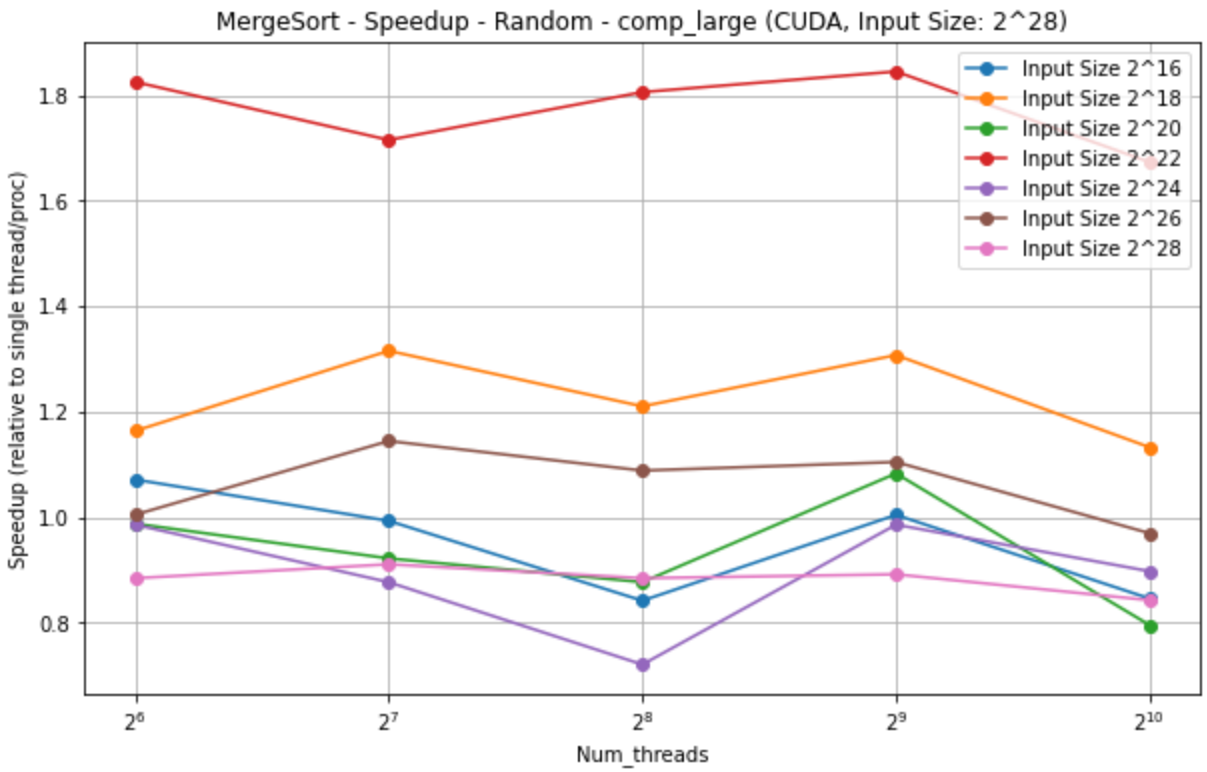

MergeSort - Speedup - Sorted - comm (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
2^6
2^7
2^8
2^9
2^{10}
Num_threads

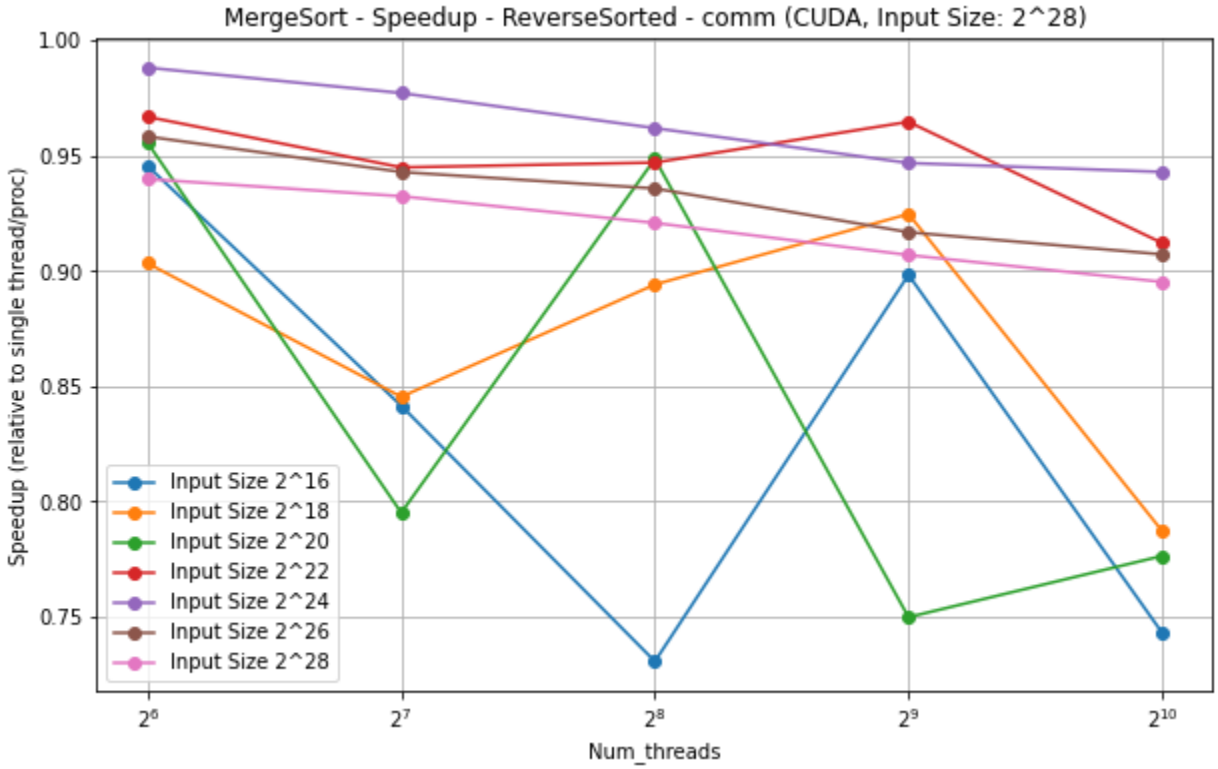

MergeSort - Speedup - Sorted - comm (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
2^6
2^7
2^8
2^9
2^{10}
Num_threads

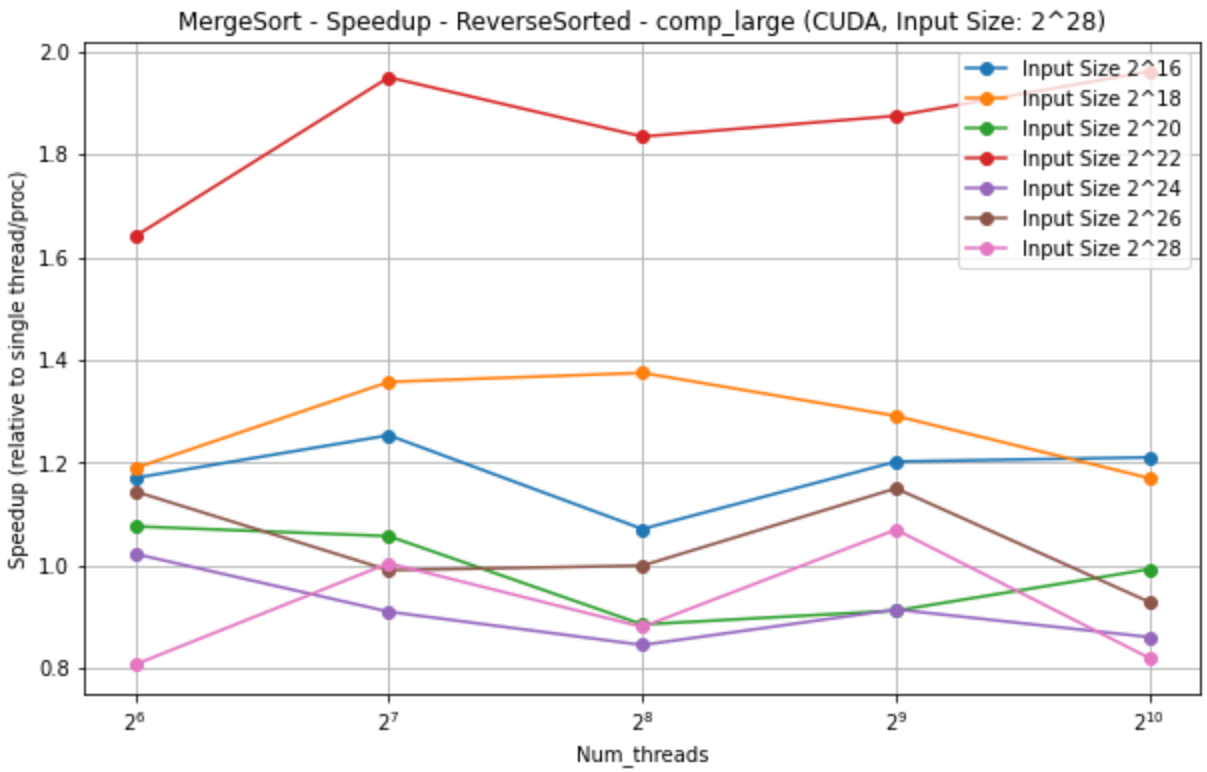

MergeSort - Speedup - Sorted - main (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
1.1
1.0
0.9
0.8
0.7
0.6
0.5
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
2^6
2^7
2^8
2^9
2^{10}
Num_threads

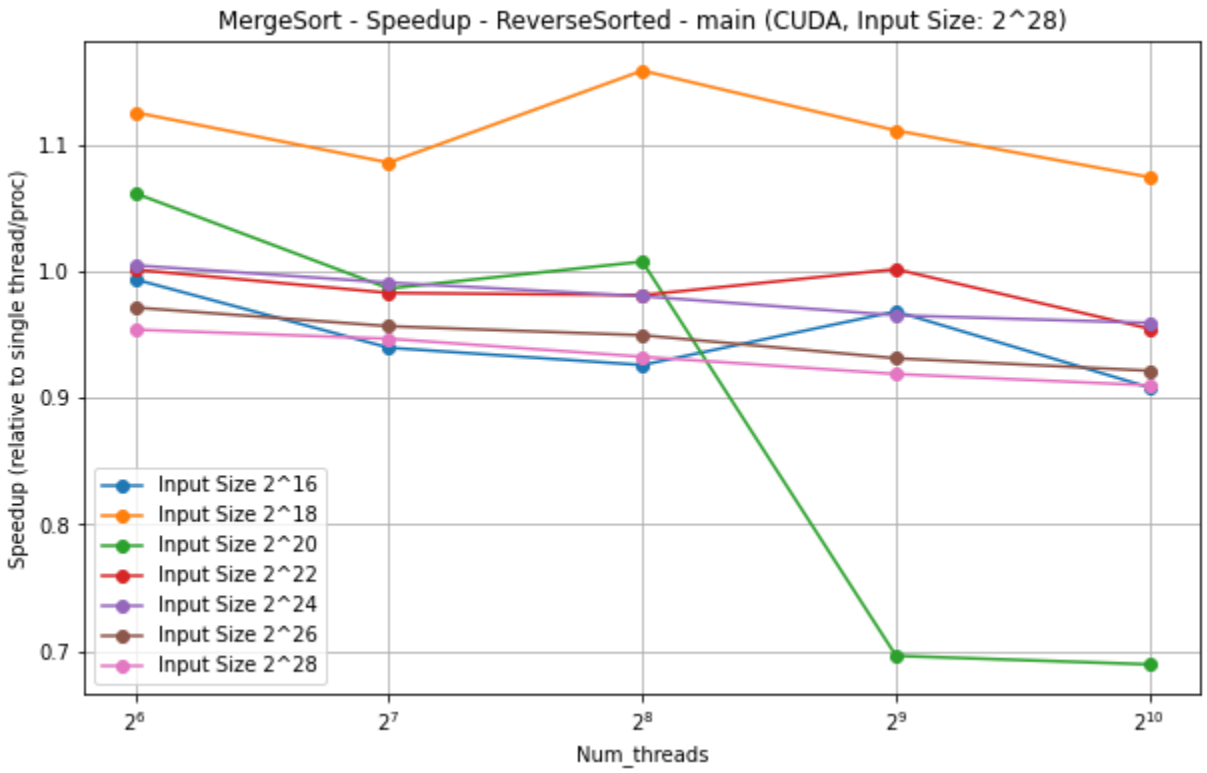

MergeSort - Speedup - Sorted - main (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

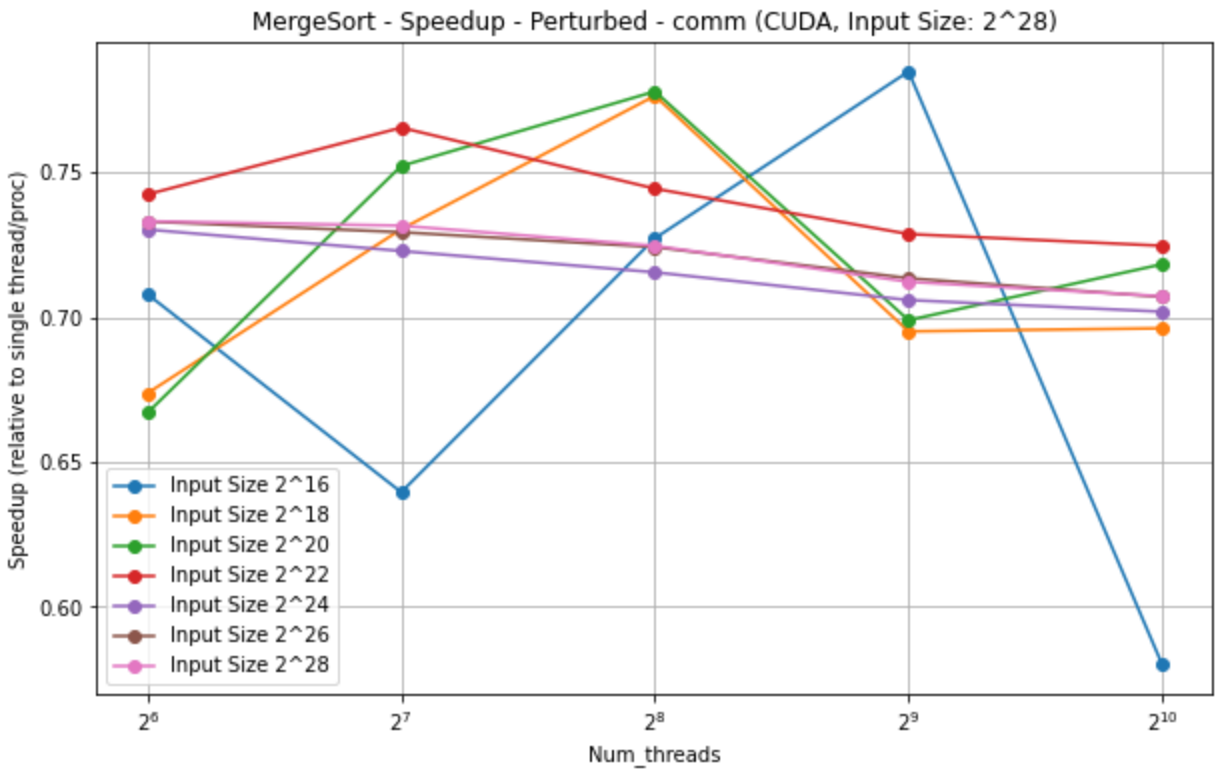

MergeSort - Speedup - Random - comp_large (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

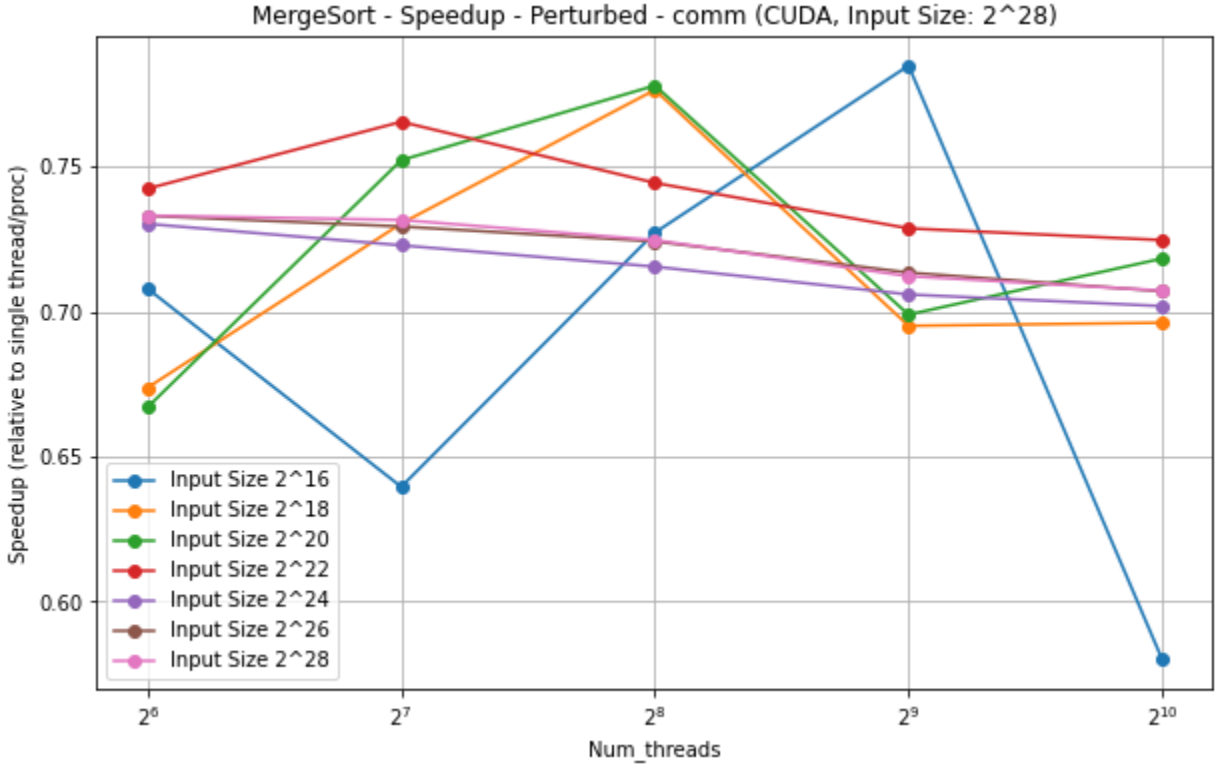

MergeSort - Speedup - Random - comp_large (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

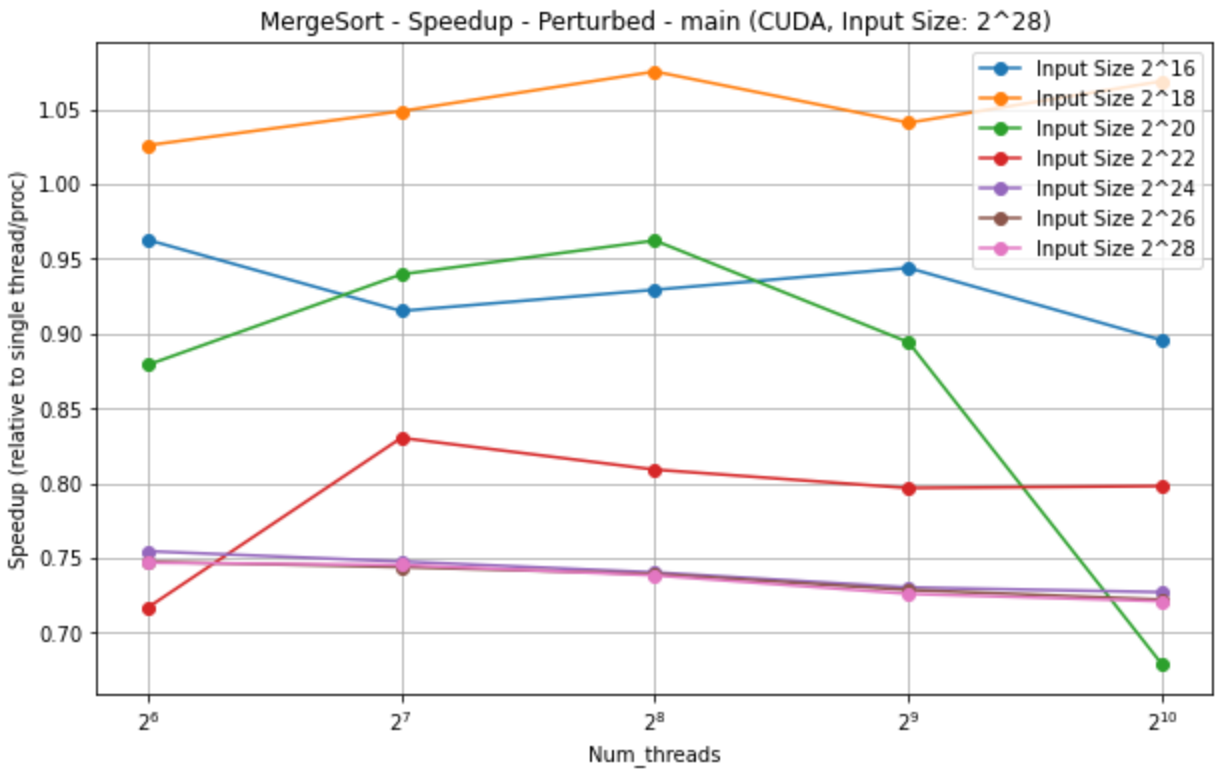

MergeSort - Speedup - ReverseSorted - comm (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28


MergeSort - Speedup - ReverseSorted - comp_large (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28


MergeSort - Speedup - ReverseSorted - main (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28


MergeSort - Speedup - Perturbed - comm (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
0.75
0.70
0.65
0.60
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
2^6
2^7
2^8
2^9
2^10
Num_threads


MergeSort - Speedup - Perturbed - comm (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
0.75
0.70
0.65
0.60
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
2^6
2^7
2^8
2^9
2^10
Num_threads



Bitonic Sort:

Considerations:

Bitonic sort is a parallel sorting algorithm that divides the input into subproblems that can be solved independently. It recursively generates a bitonic sequence, performs bitonic merges to sort, and recursively runs the process into smaller subarrays. Overall complexity of this algorithm is O(log^2(input_size)), so each thread must perform log^2(input_size) / num_processors, including both comparison and swap steps. When sorted in parallel, larger input sizes are more well-suited for bitonic sort.

The CUDA implementation of bitonic sort was able to generate cali files for all number of threads, input size, and input type. However, for MPI implementation, some combinations of number of processes and input size were not able to generate cali files due to the runtime exceeding 30 minutes. Most of the combinations that exceeded 30 minutes were with input size 2^{28}, and when used 1024 processes.

Performance Analysis:

As the algorithm naturally divides the data into subproblems that can be independently sorted, the bitonic sort algorithm is particularly suitable for parallelization. It did not show the advantage of parallel processing, as it performed badly for smaller input sizes. However, the performance

improved with increase in input size, showing that the algorithm takes advantage of parallelism with larger input size.

Strong scaling:
For MPI implementation, we can clearly see that the average runtime decreases as the number of processes increases for all input types. For most plots, sorted input types had the best performance, and random input type performed the worst. Both communication and computation for all input types and sizes scaled well, showing good parallelism.

For CUDA implementation, we can see that both communication and computation with smaller input size does not perform better with more threads. However, as the input size increases, it starts to perform better, parallelizing its tasks. Overall, input types of Sorted and Reverse sorted performed better than the other two types, which is likely due to how the algorithm sorts its array in a bitonic sequence.

Weak scaling:
For MPI implementation, we can see that larger input sizes scaled better for both communication and computation than the smaller input sizes, showing that the algorithm works best with larger input sizes. The input type did not seem to make much difference in the performance of each input sizes.

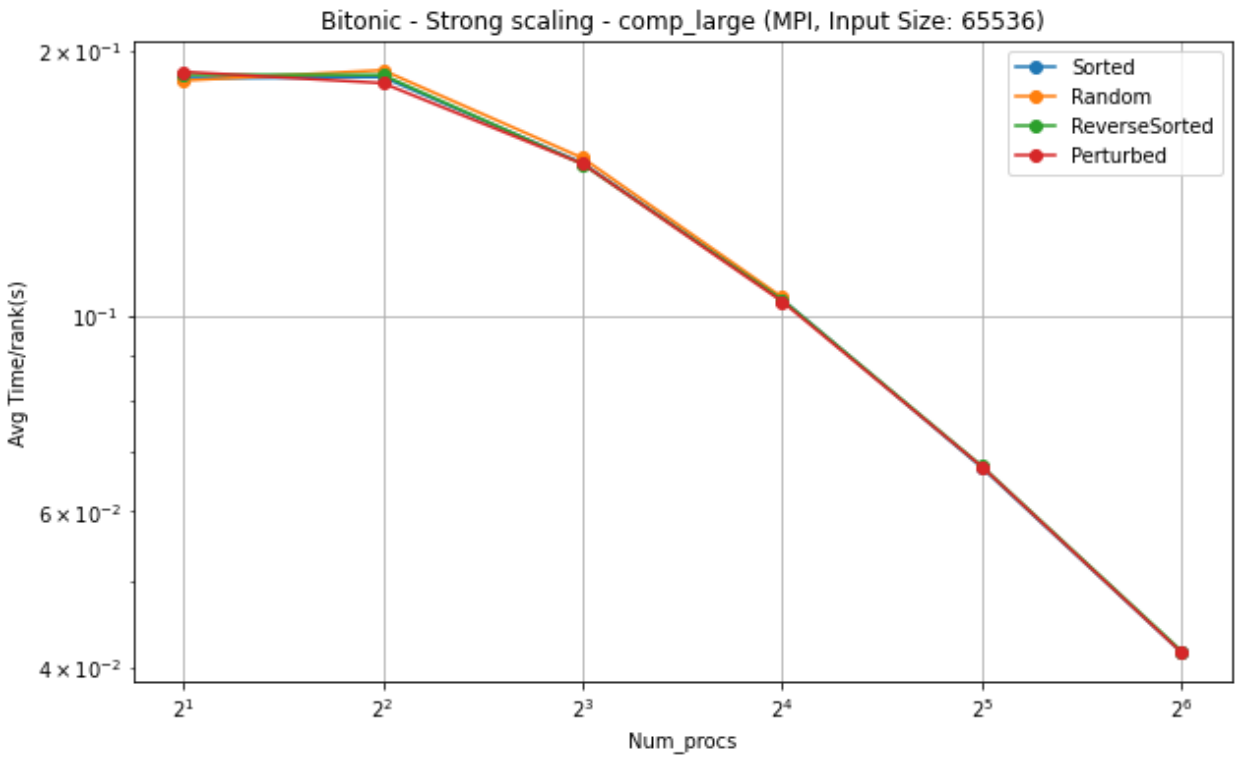
For CUDA implementation, we can again see that larger input sizes have steeper slopes for both communication and computation than the smaller input sizes. We can also see that communication scaled a lot worse than computation, providing a bottleneck which led to a slight worse performance in main.

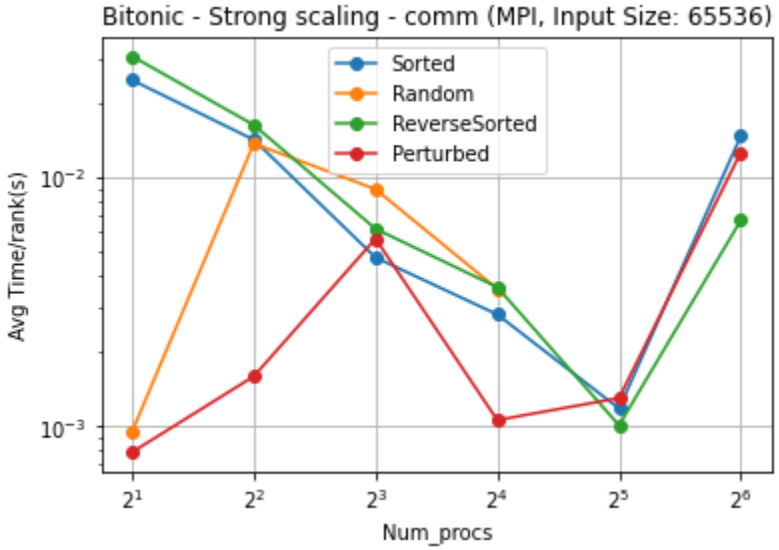
Speedup:
For MPI implementation, the speedup increases continuously with an increasing number of processors, and peaks at 2^6 for most of the plots.

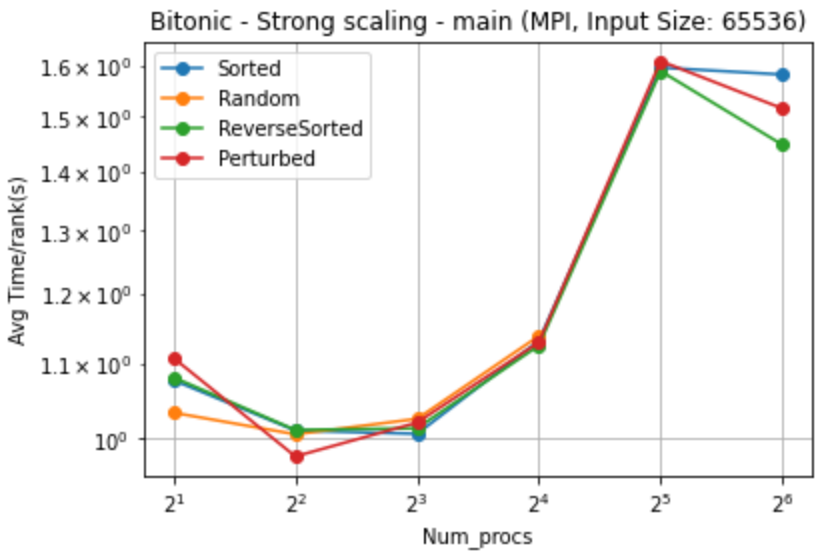
For CUDA implementation, the communication plots show a good increase in speedup with the number of threads, with peak at 2^9 for most of the plots. However, no trend could be found in computation. Therefore main plot has a slight steady increase for speedup. Another observation is that the speedup for smaller input sizes tended to decrease for some plots, and speedup for larger input sizes always had a large increase.

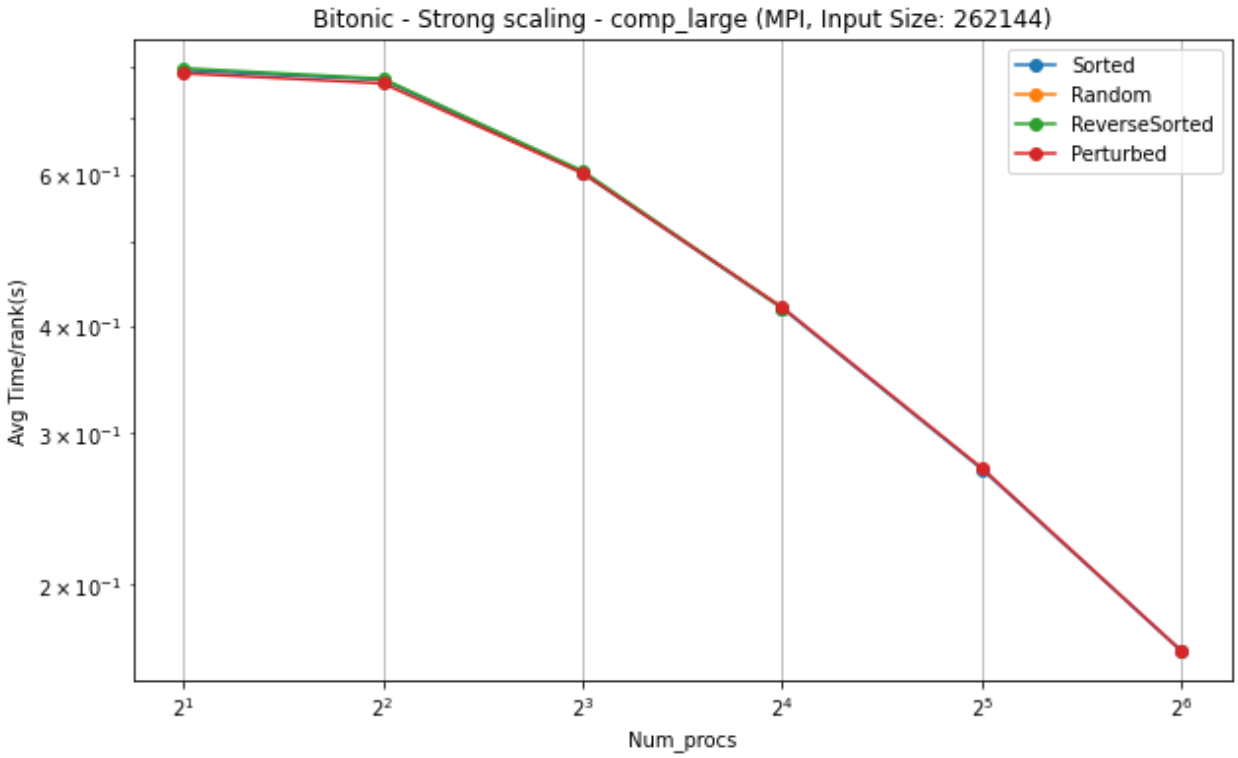
Bitonic Sort Plots:

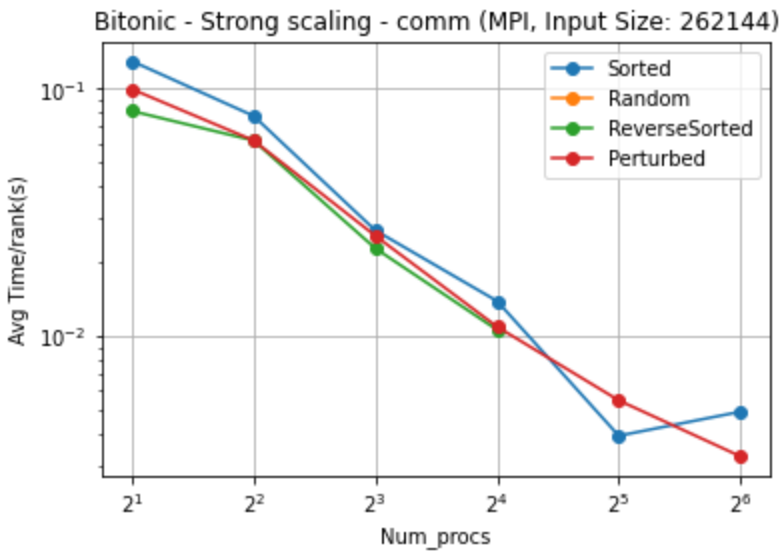
MPI:
Strong Scaling:

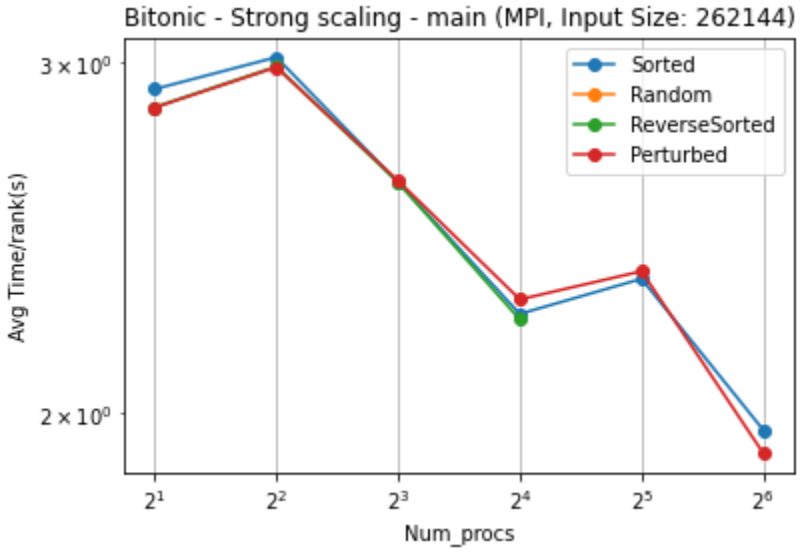

Bitonic - Strong scaling - comp_large (MPI, Input Size: 65536)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_procs

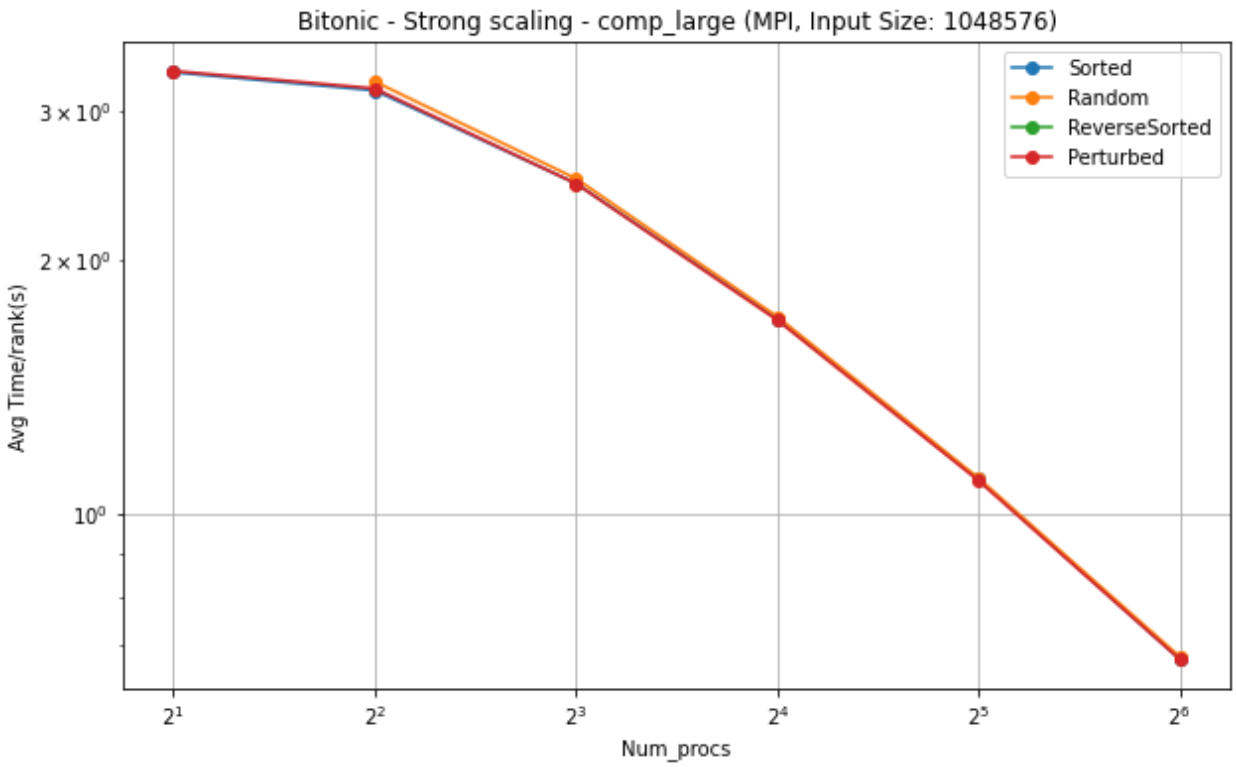

Bitonic - Strong scaling - comm (MPI, Input Size: 65536)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_procs

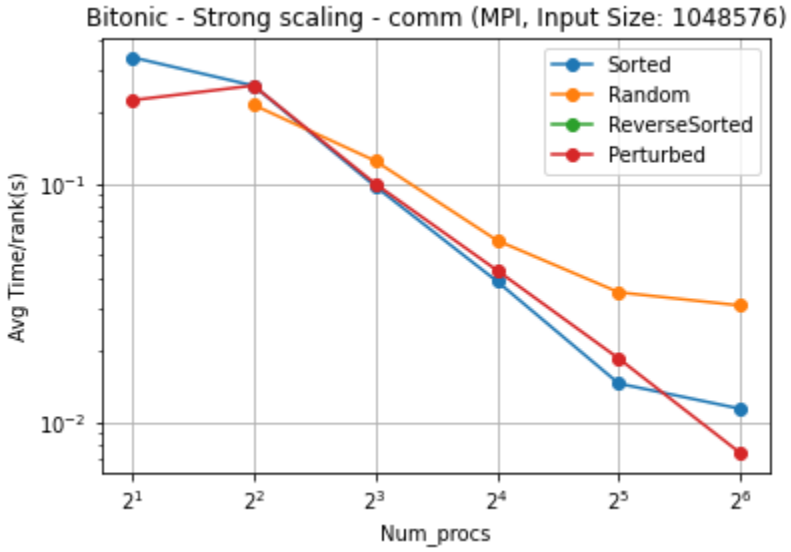

Bitonic - Strong scaling - main (MPI, Input Size: 65536)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_procs

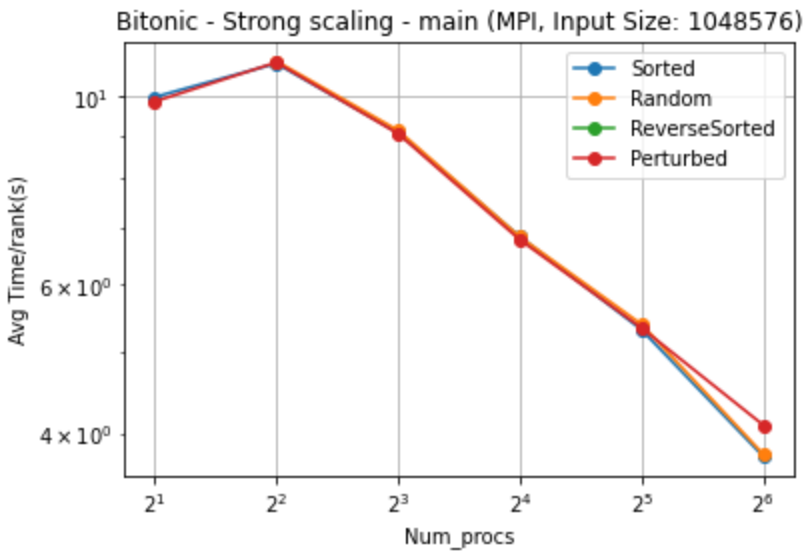

Bitonic - Strong scaling - comp_large (MPI, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_procs

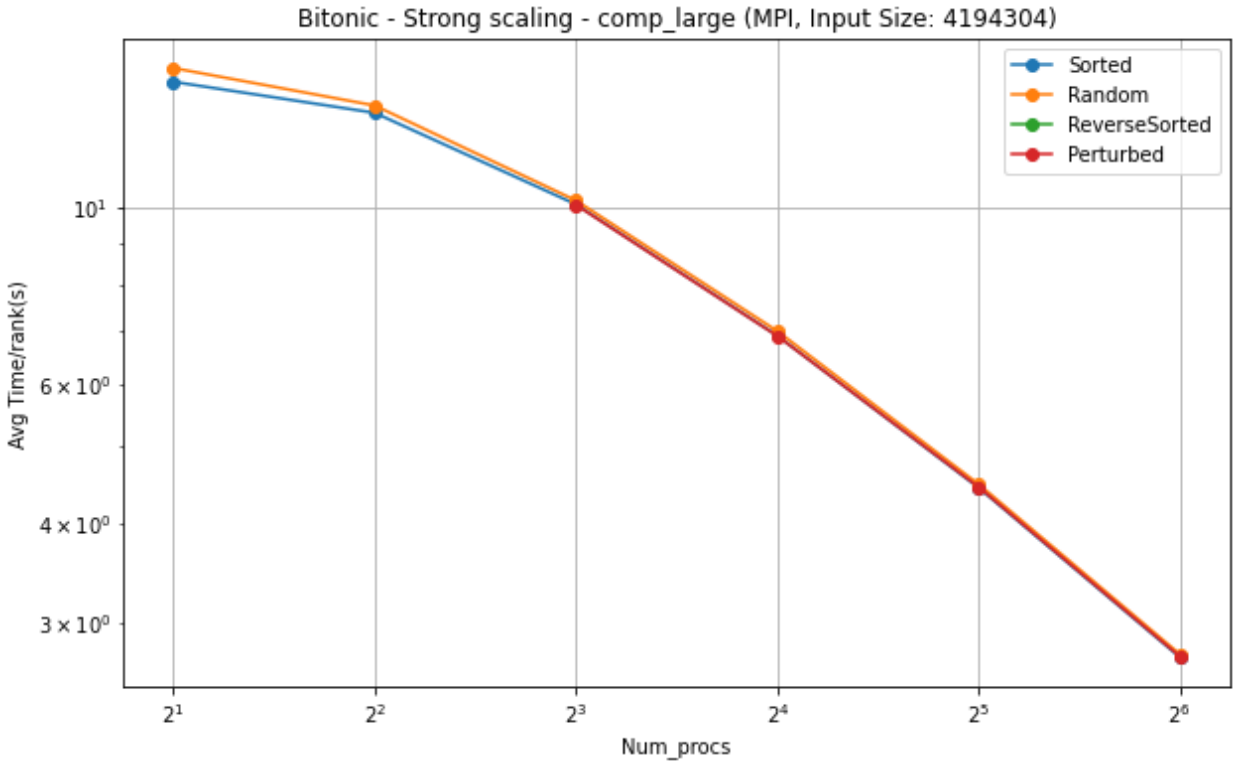

Bitonic - Strong scaling - comm (MPI, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{-1}
10^{-2}
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs

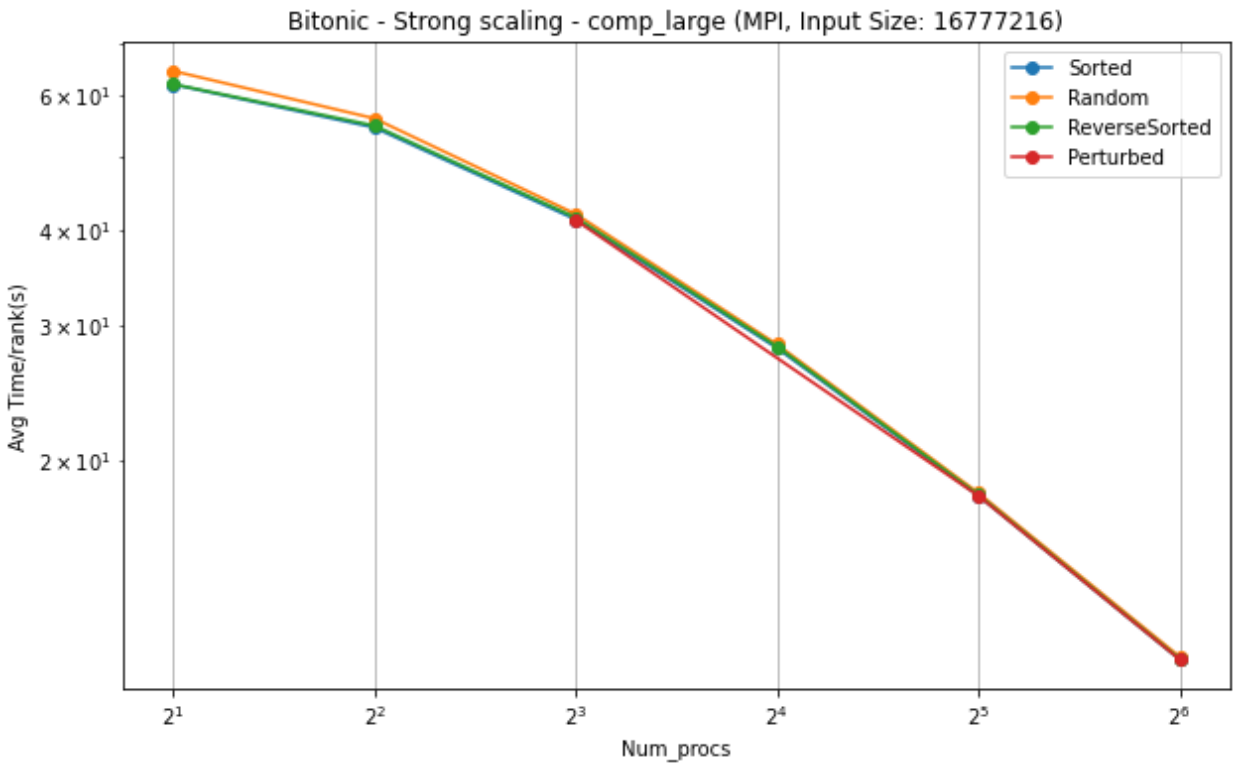

Bitonic - Strong scaling - main (MPI, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
3×10^0
2×10^0
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs

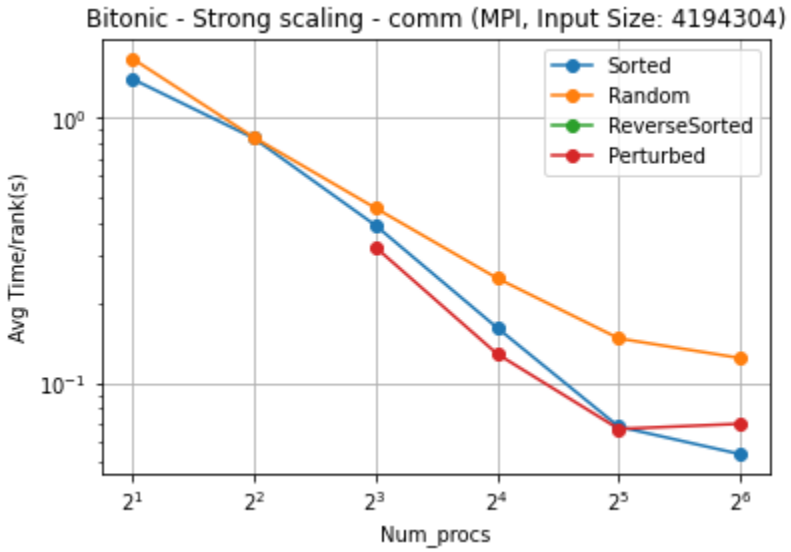

Bitonic - Strong scaling - comp_large (MPI, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
3×10^0
2×10^0
10^0
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs

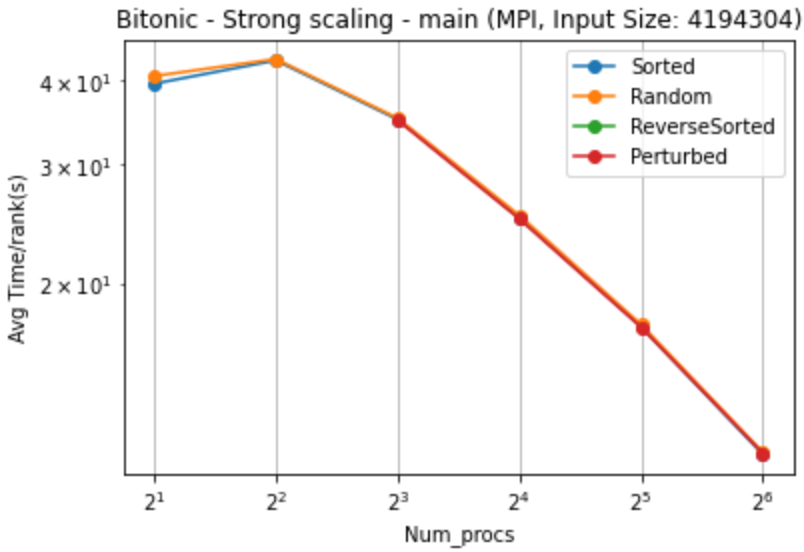

Bitonic - Strong scaling - comm (MPI, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{-1}
10^{-2}
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs

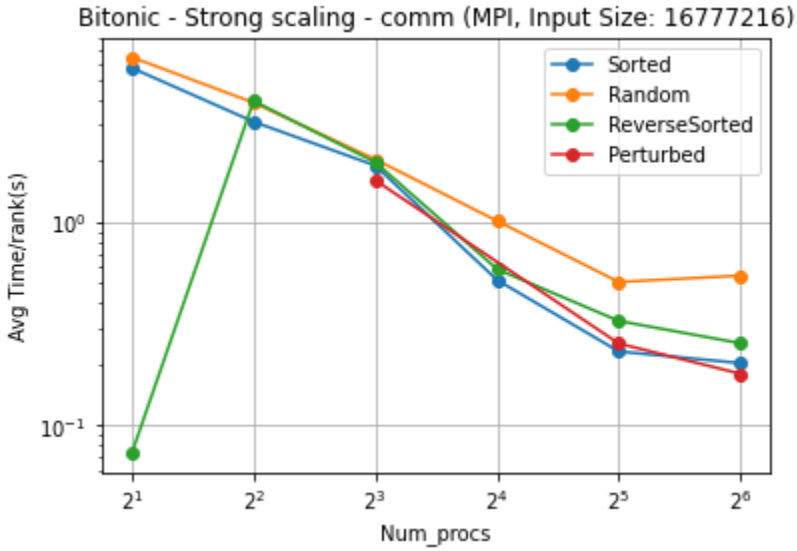

Bitonic - Strong scaling - main (MPI, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
6×10^0
4×10^0
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs

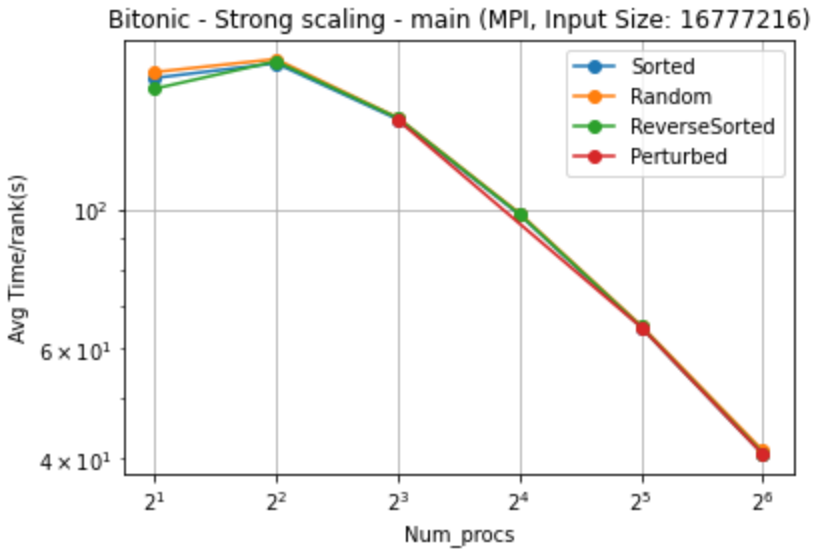

Bitonic - Strong scaling - comp_large (MPI, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^1
6×10^0
4×10^0
3×10^0
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs


Bitonic - Strong scaling - comp_large (MPI, Input Size: 16777216)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
6×10^1
4×10^1
3×10^1
2×10^1
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs

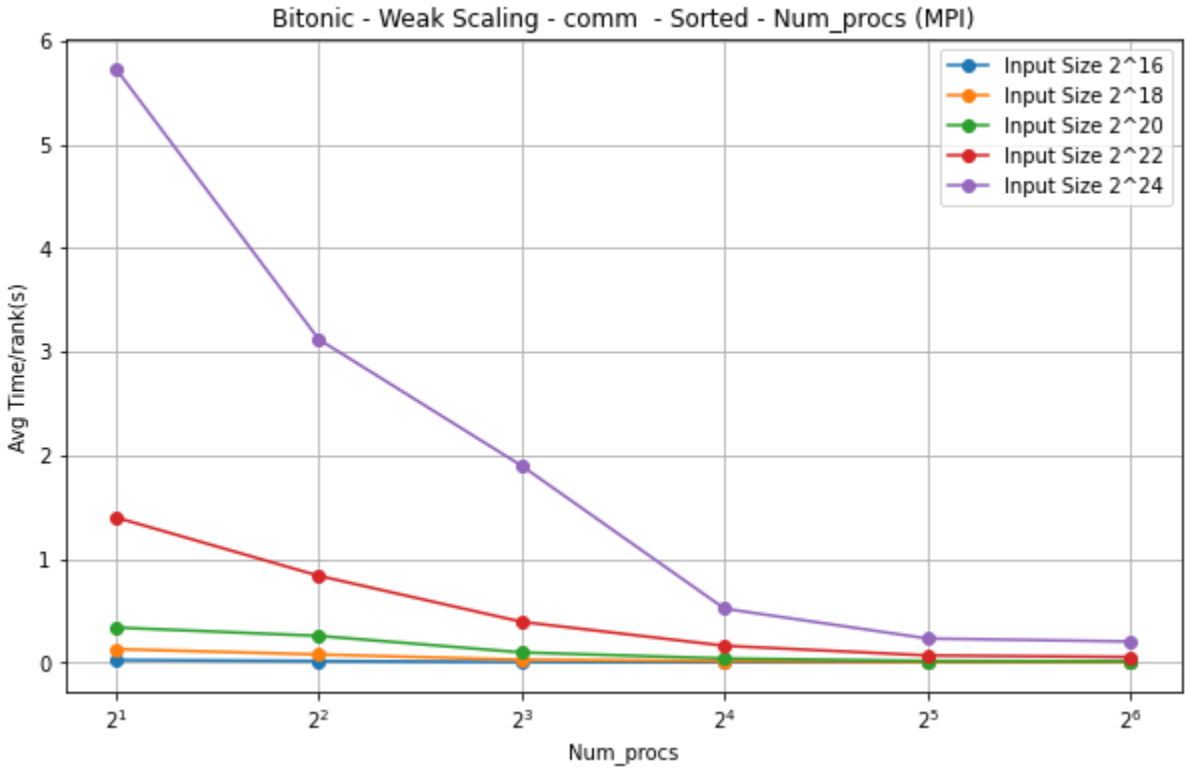

Bitonic - Strong scaling - comm (MPI, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^0
10^{-1}
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs

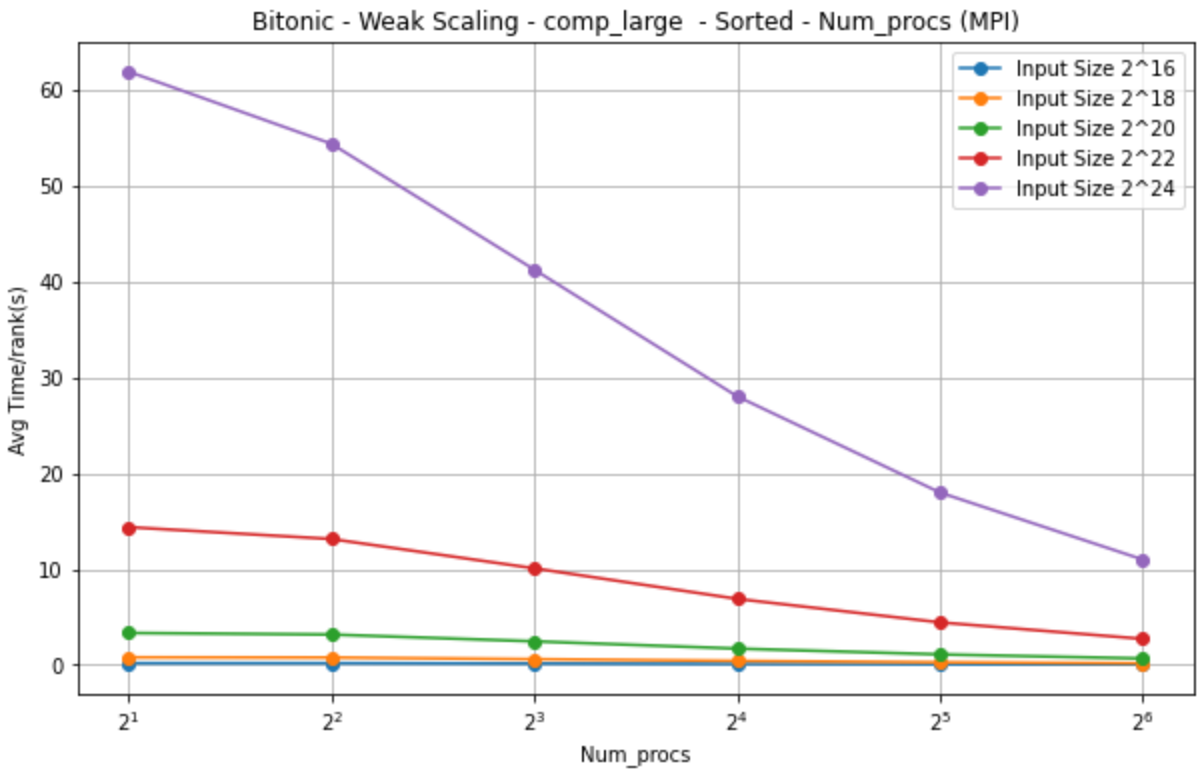

Bitonic - Strong scaling - main (MPI, Input Size: 4194304)
Avg Time/rank(s)
4×10^1
3×10^1
2×10^1
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs
Sorted
Random
ReverseSorted
Perturbed

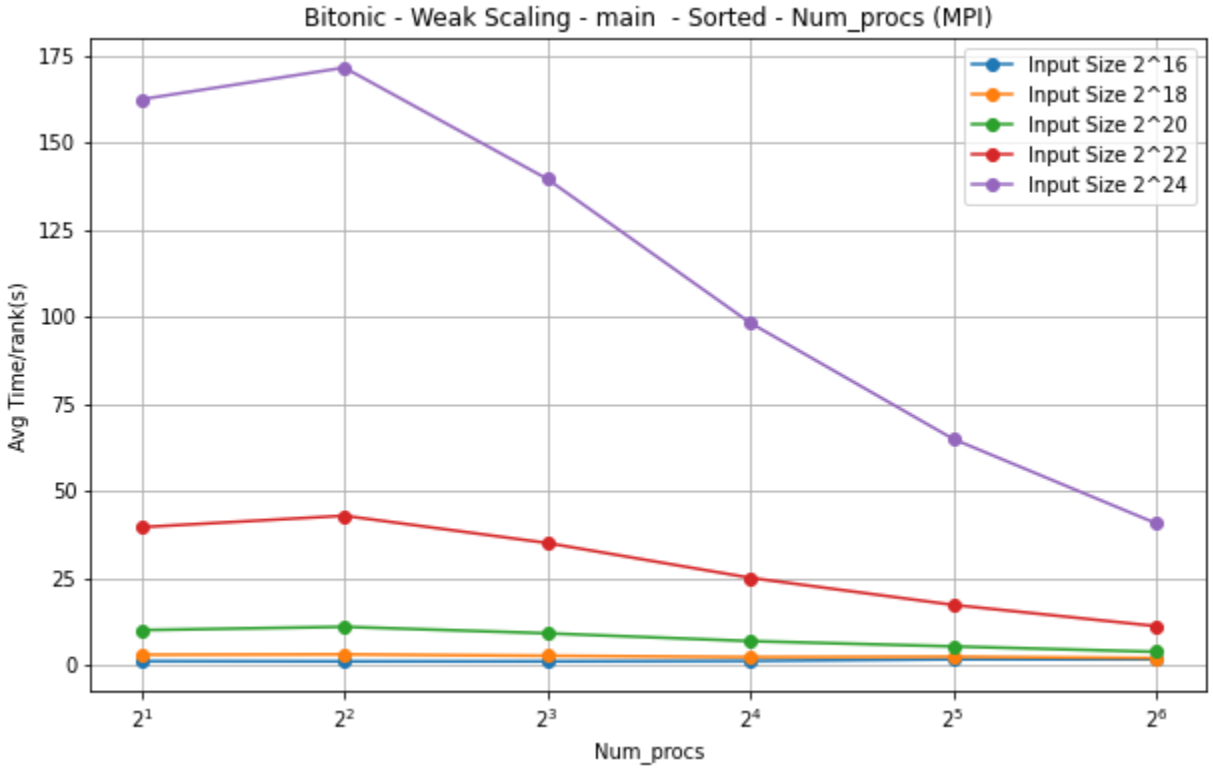

Bitonic - Strong scaling - comm (MPI, Input Size: 16777216)
Avg Time/rank(s)
10^0
10^{-1}
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs
Sorted
Random
ReverseSorted
Perturbed

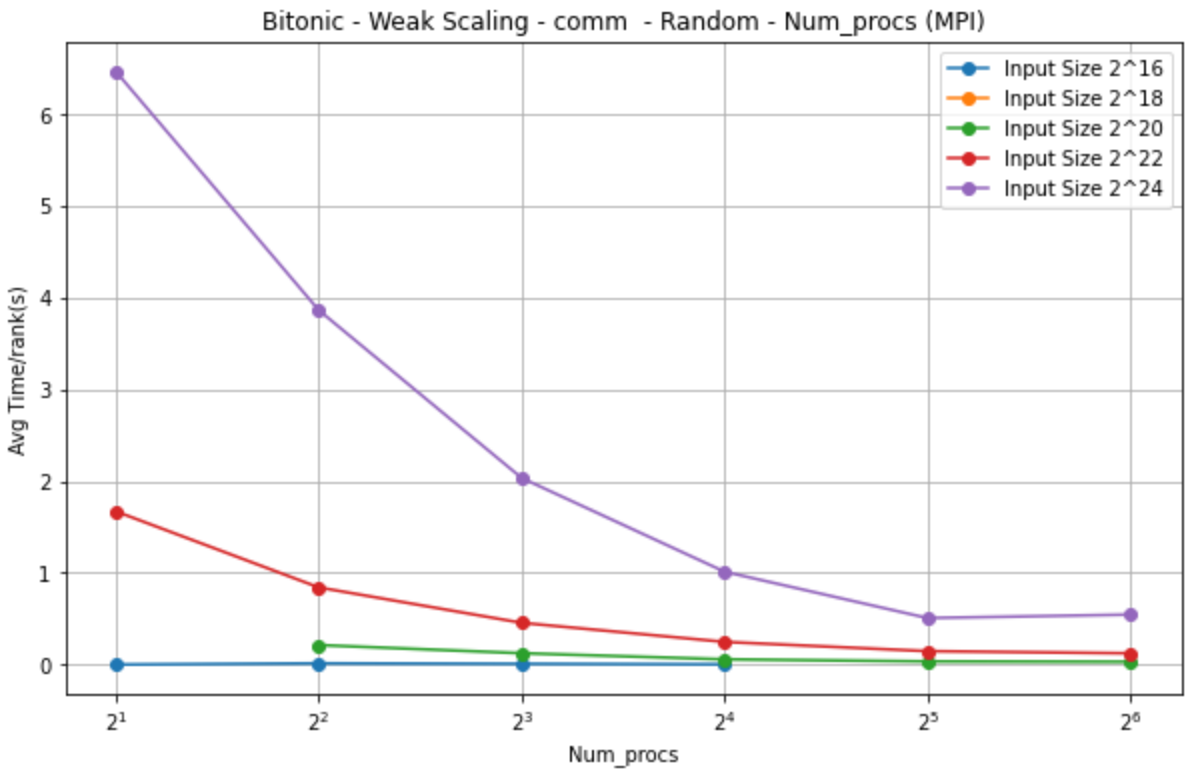


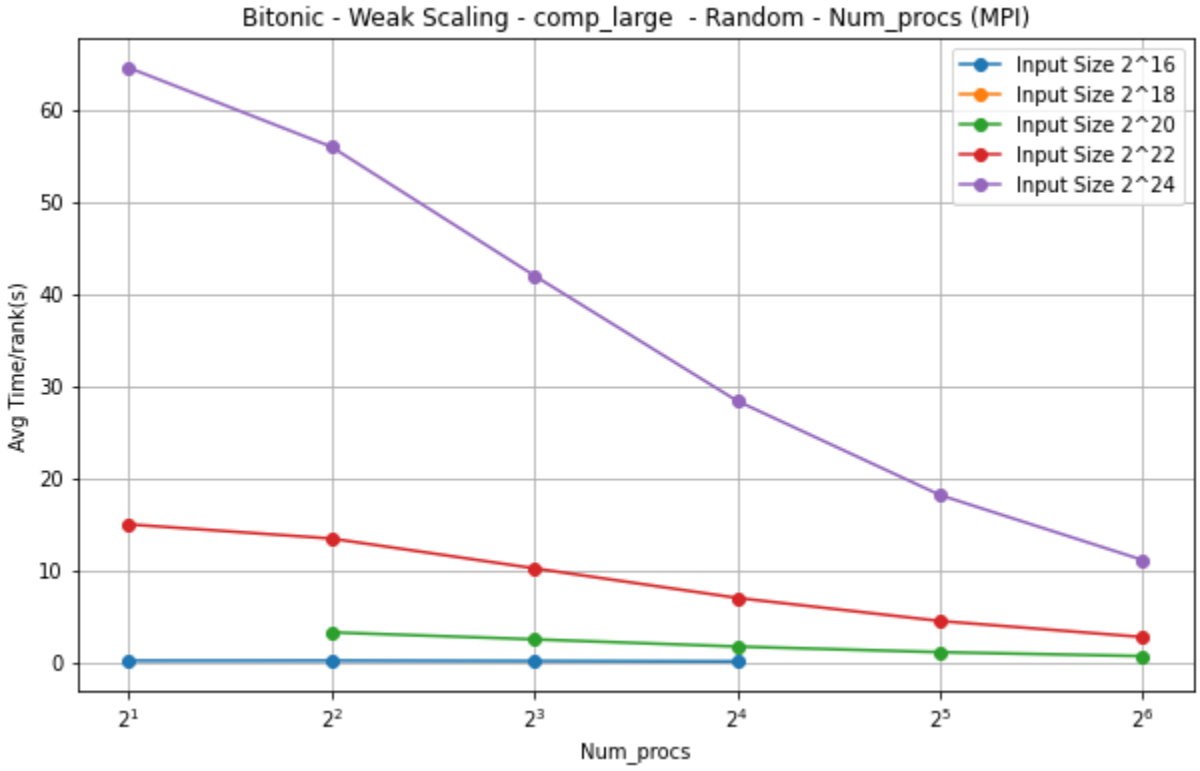
Weak Scaling:

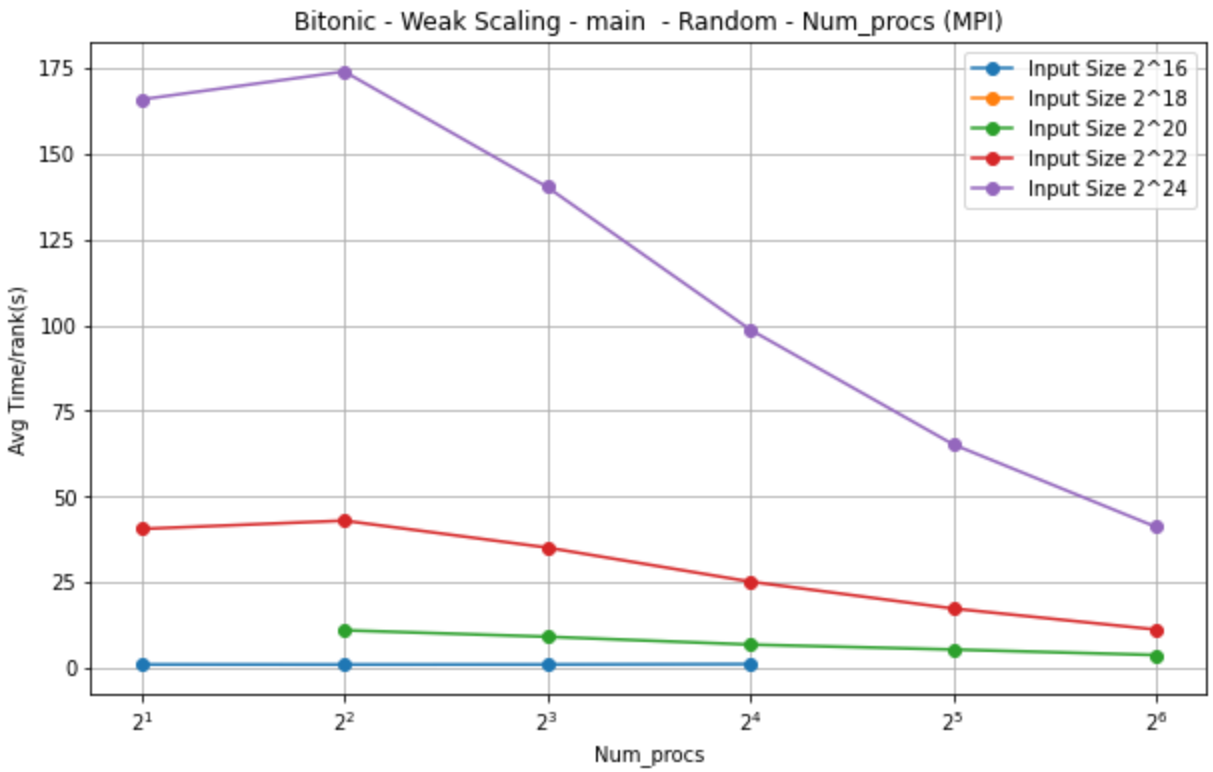


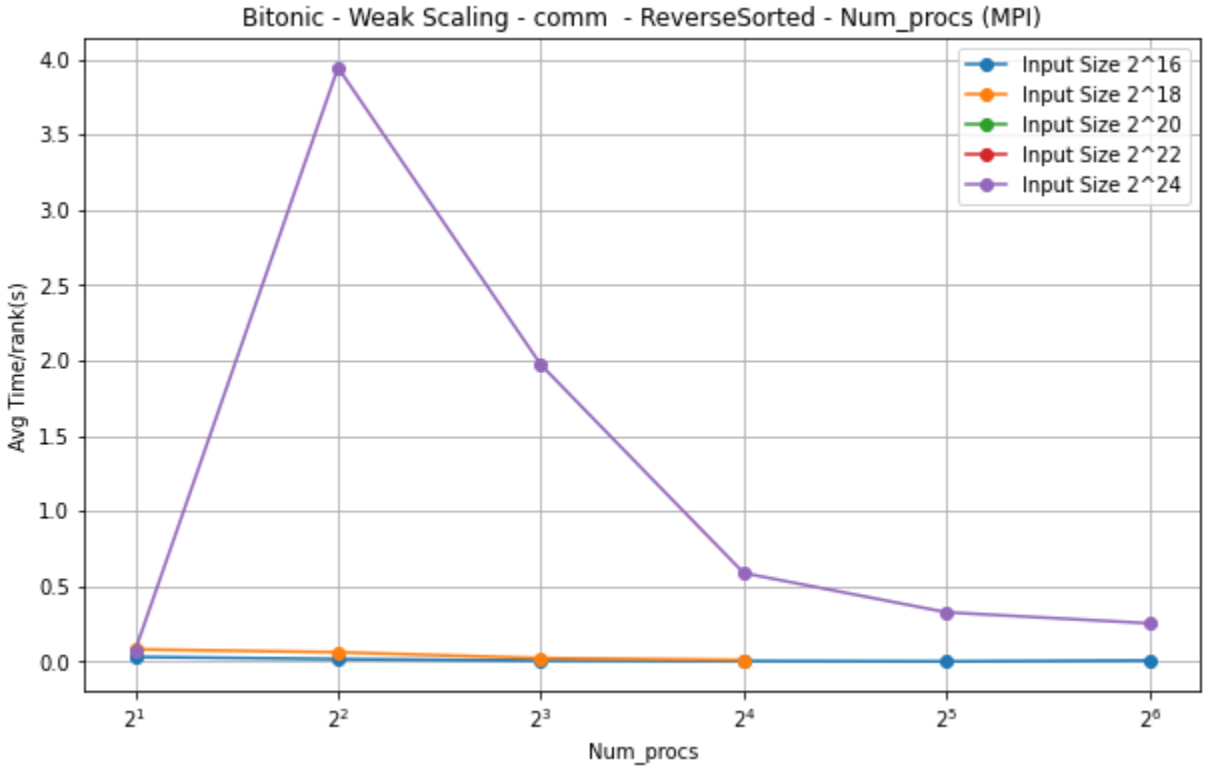

Bitonic - Weak Scaling - comp_large - Sorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
0
10
20
30
40
50
60
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs

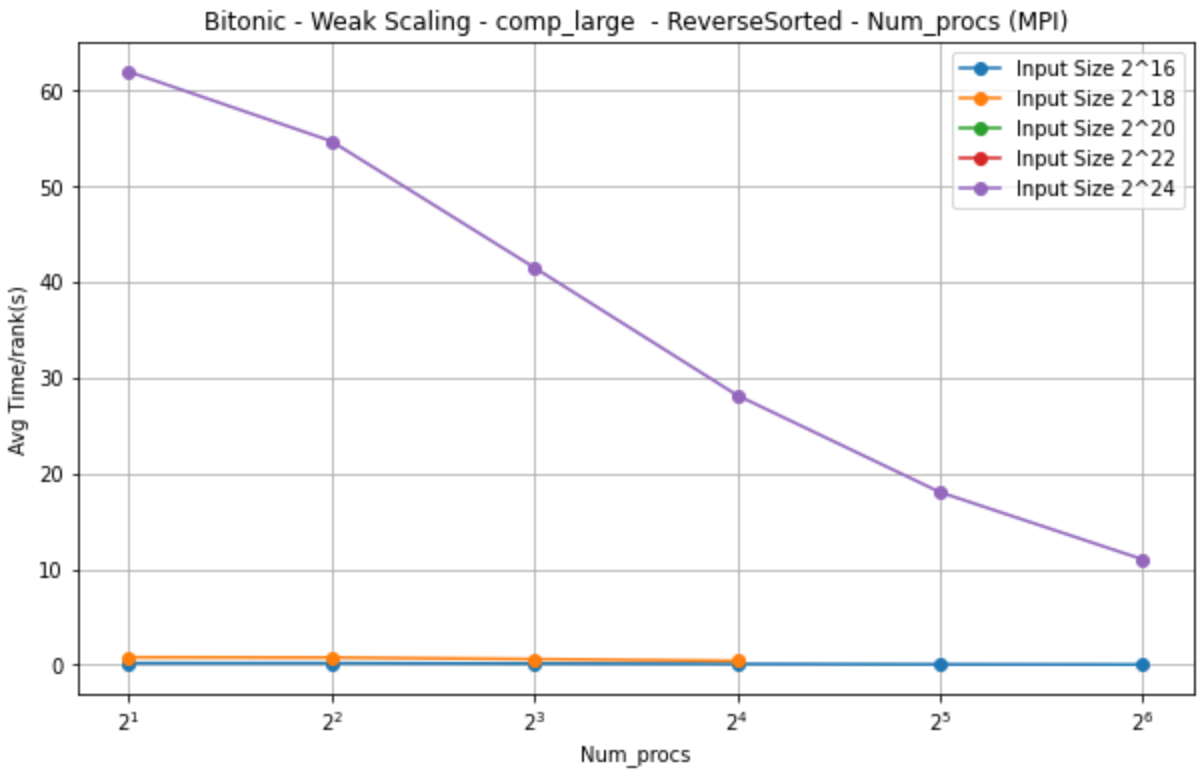

Bitonic - Weak Scaling - main - Sorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
0
25
50
75
100
125
150
175
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs

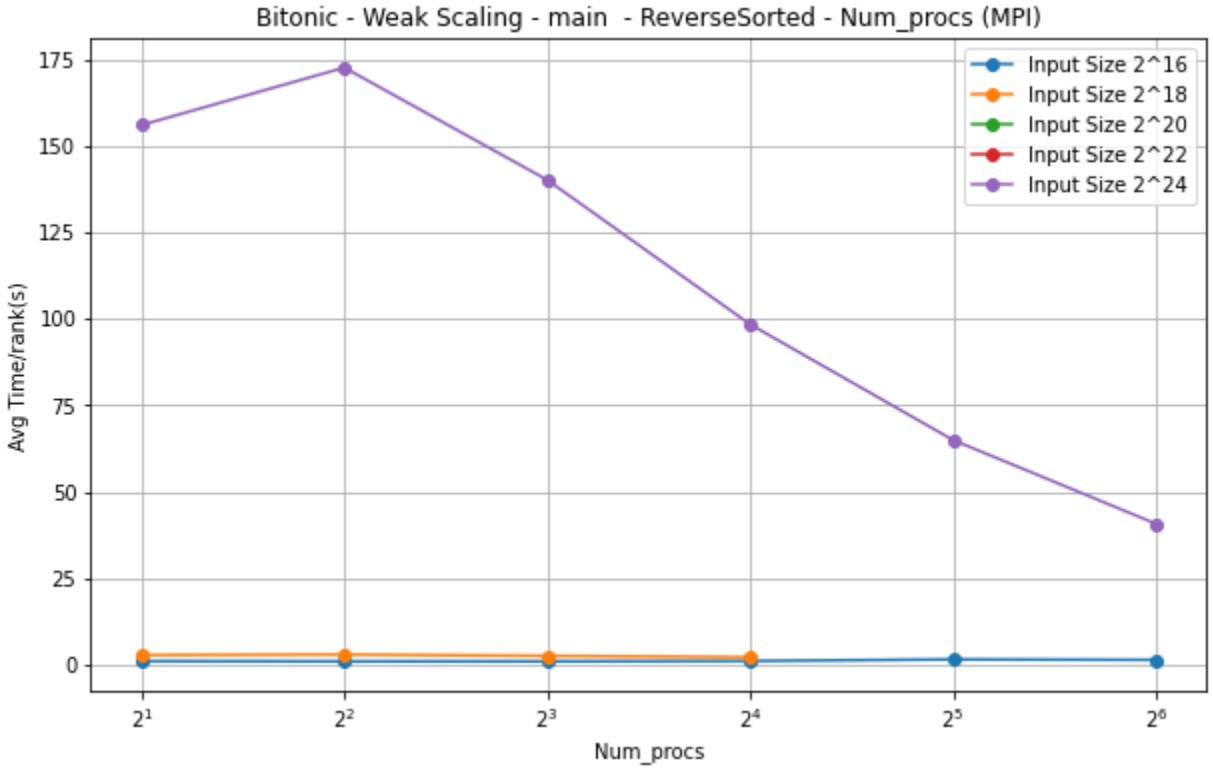

Bitonic - Weak Scaling - comm - Random - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
Num_procs
0
1
2
3
4
5
6
2^1
2^2
2^3
2^4
2^5
2^6

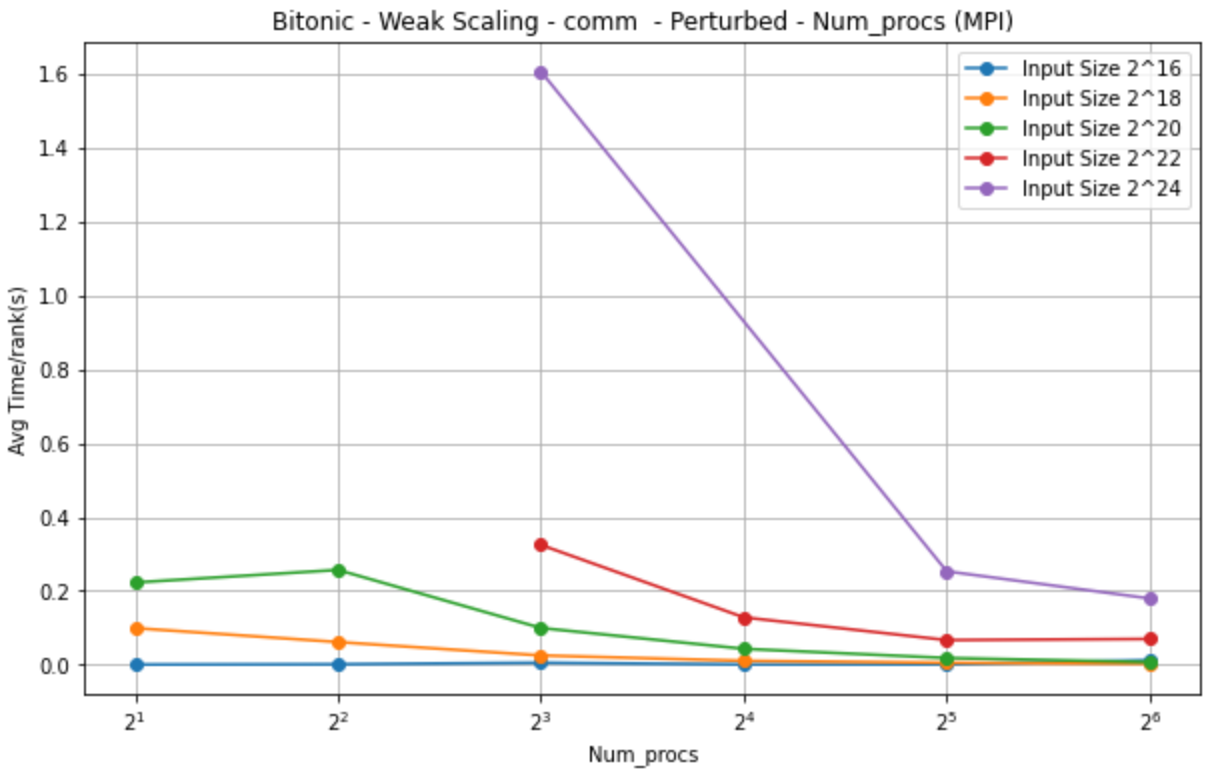

Bitonic - Weak Scaling - comp_large - Random - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
Num_procs
0
10
20
30
40
50
60
2^1
2^2
2^3
2^4
2^5
2^6

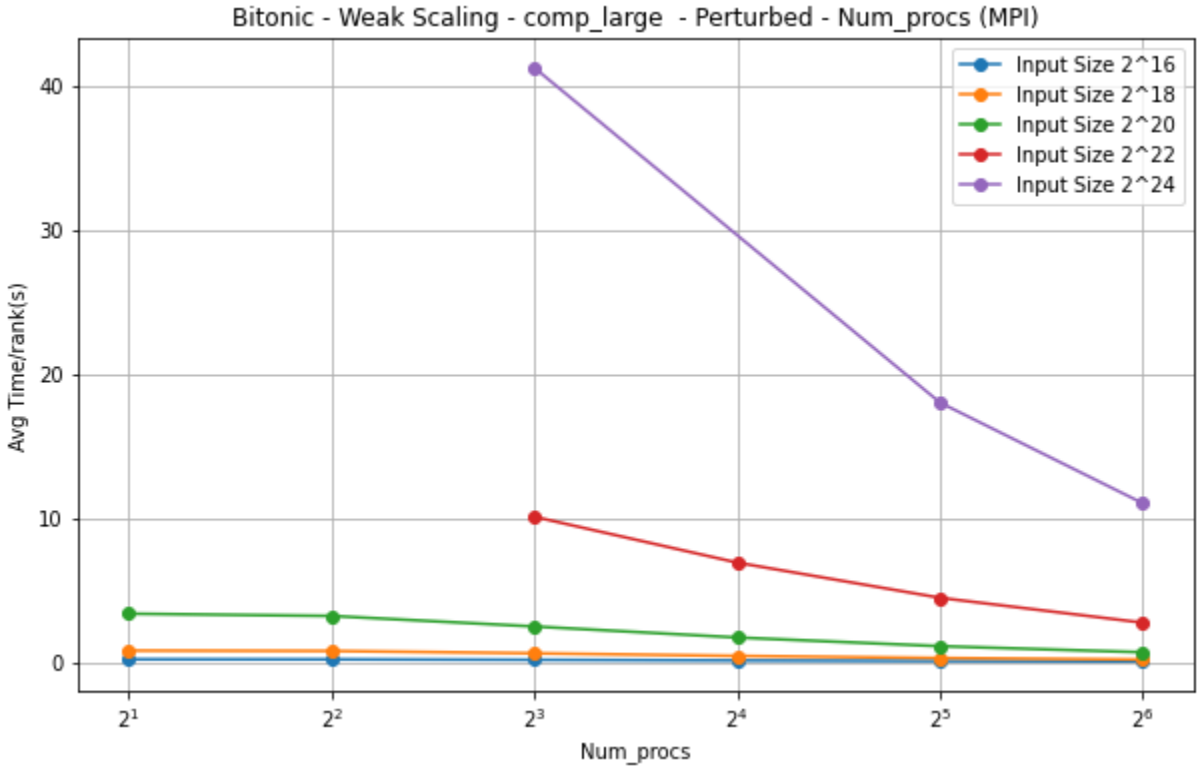

Bitonic - Weak Scaling - main - Random - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
Num_procs
0
25
50
75
100
125
150
175
2^1
2^2
2^3
2^4
2^5
2^6

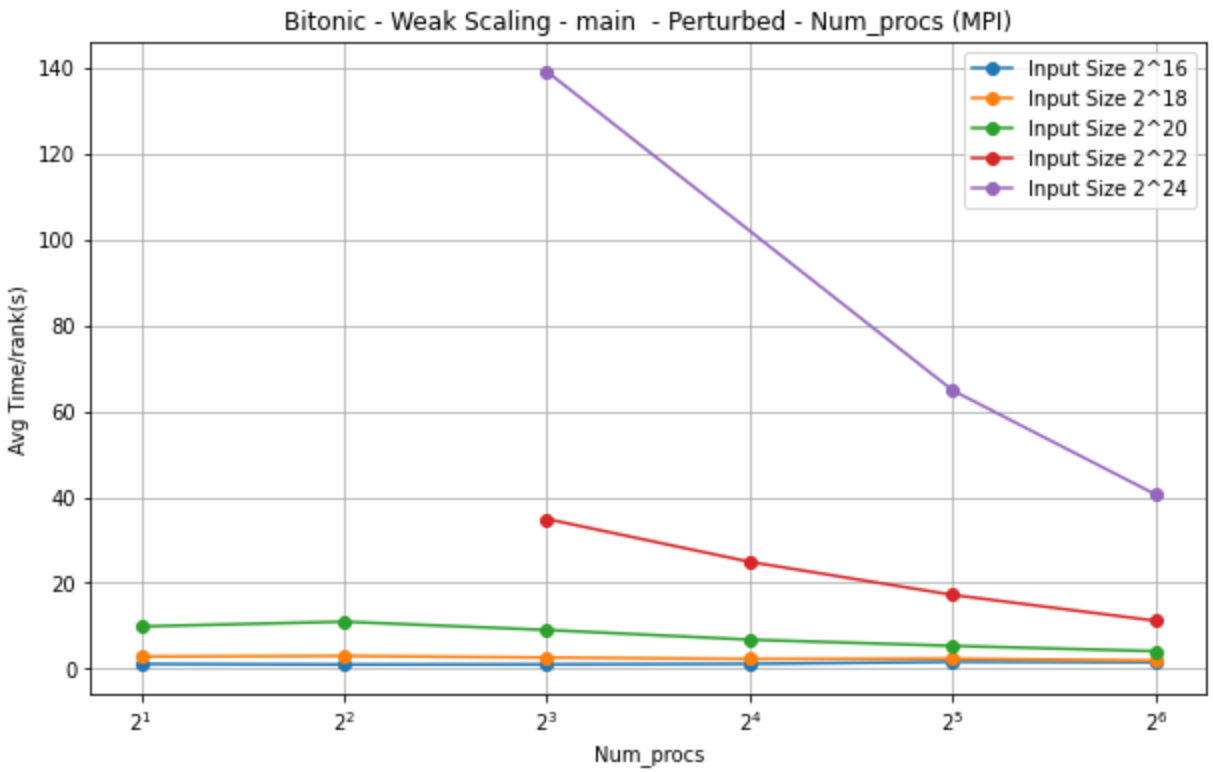

Bitonic - Weak Scaling - comm - ReverseSorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
Num_procs
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
2^1
2^2
2^3
2^4
2^5
2^6


Bitonic - Weak Scaling - comp_large - ReverseSorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
0
10
20
30
40
50
60
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs

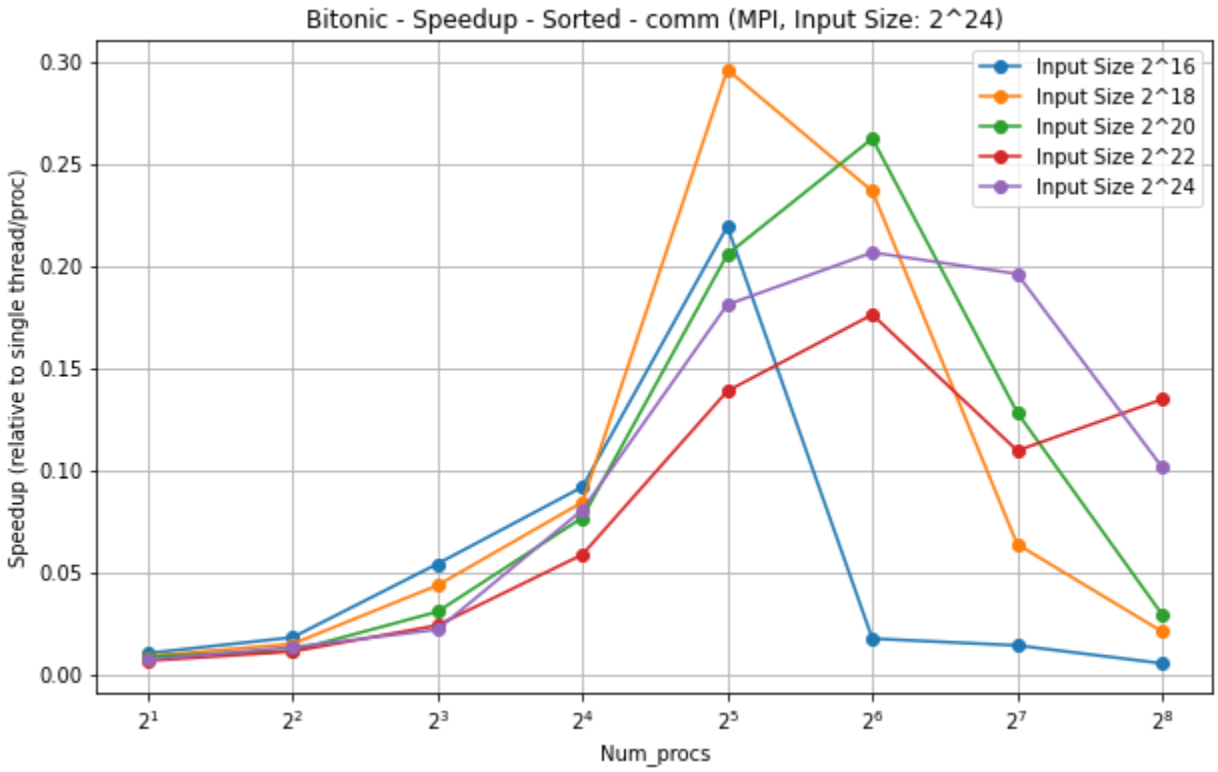

Bitonic - Weak Scaling - main - ReverseSorted - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
0
25
50
75
100
125
150
175
2^1
2^2
2^3
2^4
2^5
2^6
Num_procs

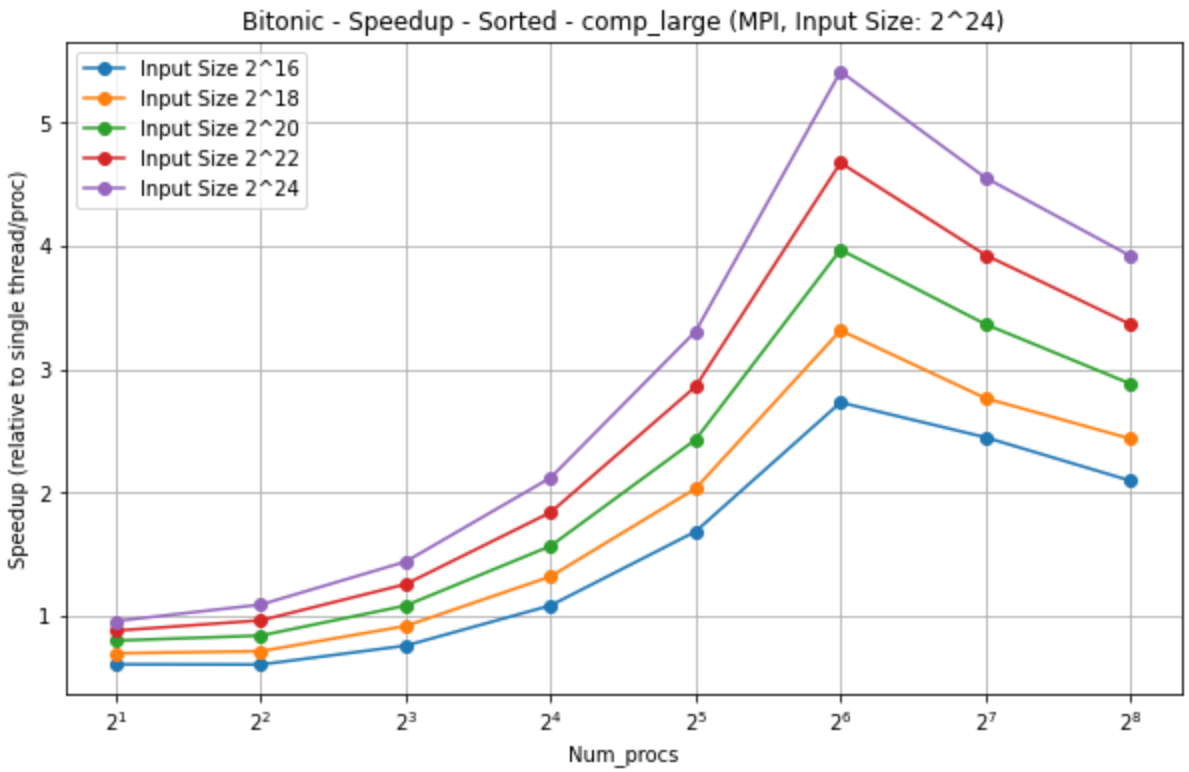

Bitonic - Weak Scaling - comm - Perturbed - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
Num_procs
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
2^1
2^2
2^3
2^4
2^5
2^6

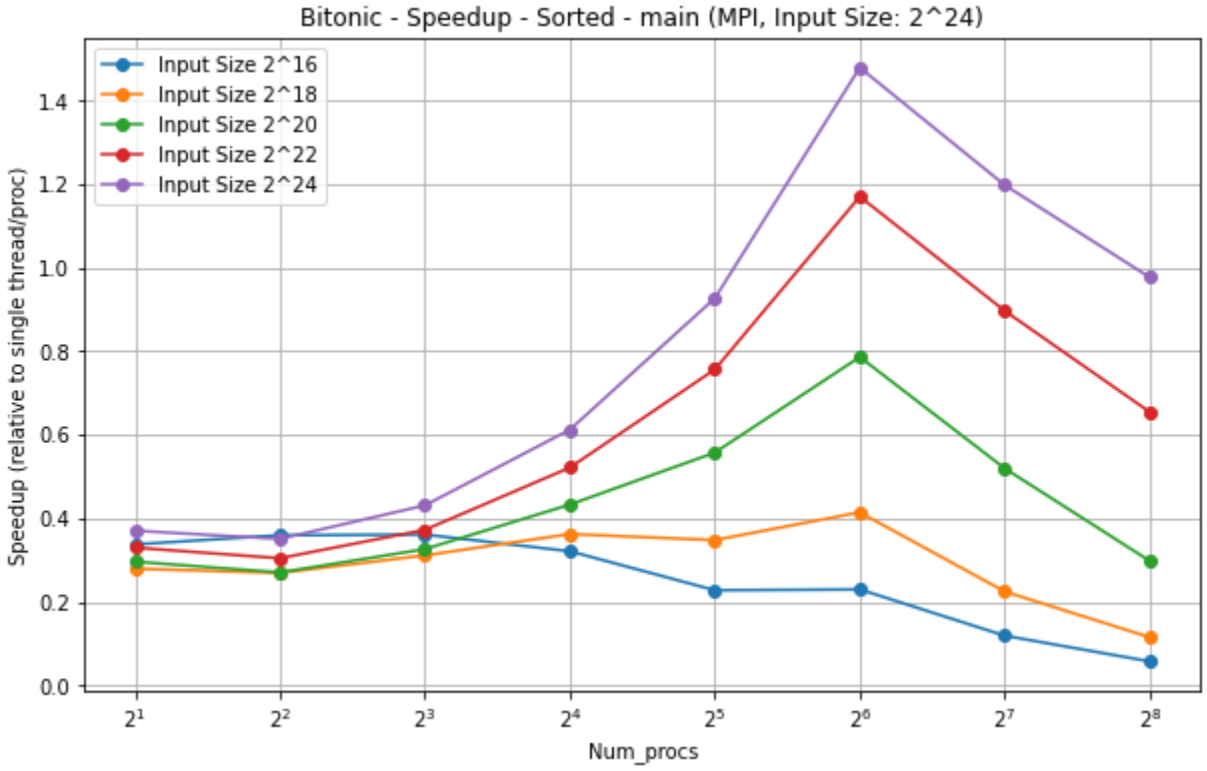

Bitonic - Weak Scaling - comp_large - Perturbed - Num_procs (MPI)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Avg Time/rank(s)
Num_procs
0
10
20
30
40
2^1
2^2
2^3
2^4
2^5
2^6

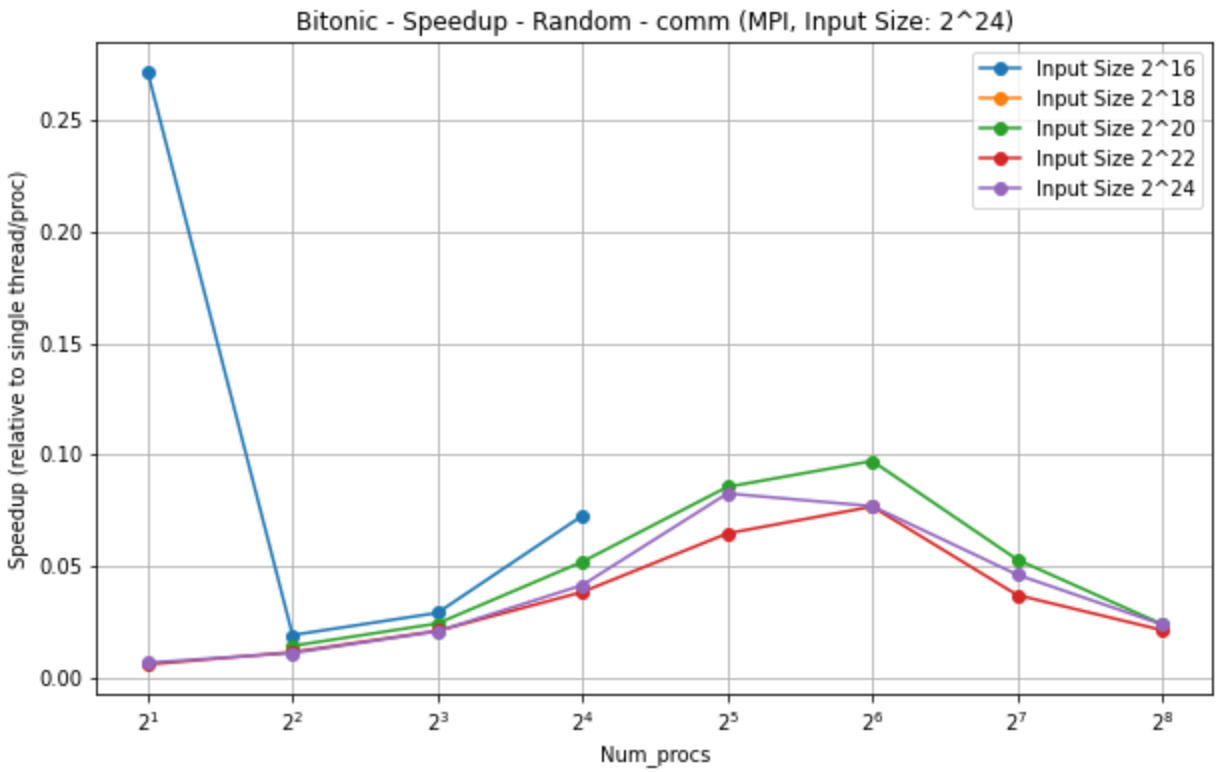

Bitonic - Weak Scaling - main - Perturbed - Num_procs (MPI)
Avg Time/rank(s)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24

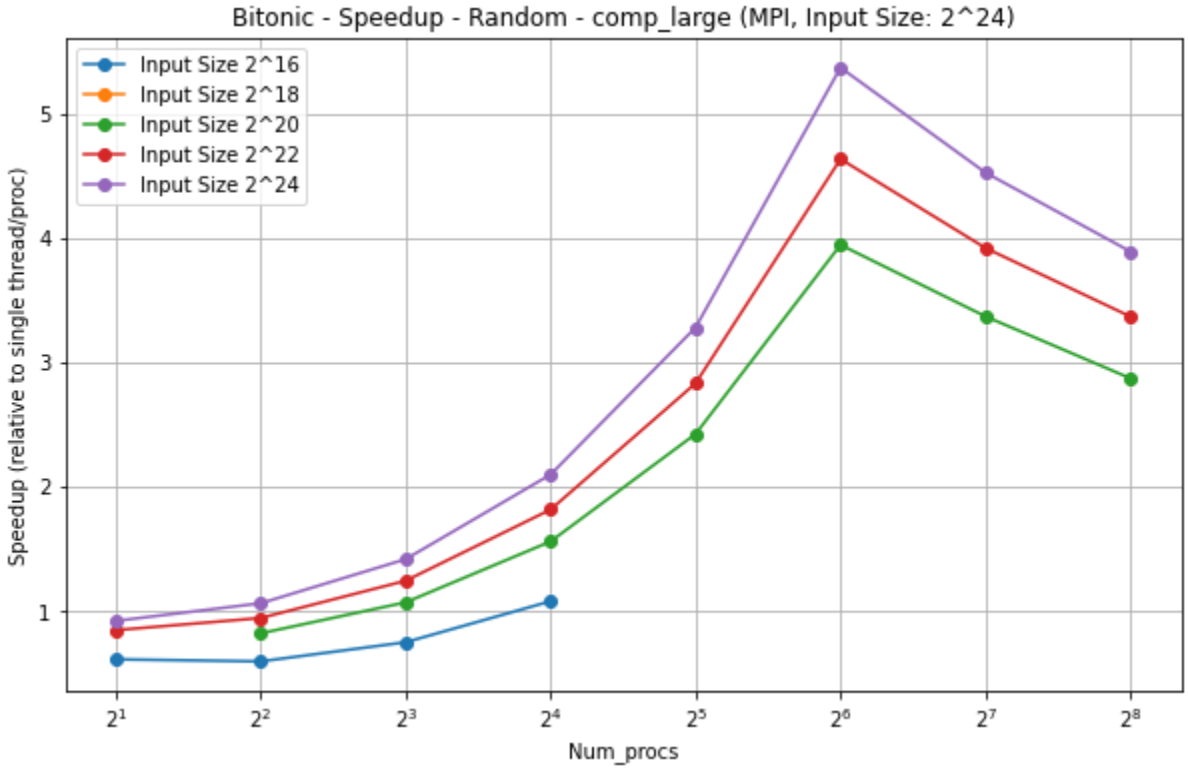
Speedup:

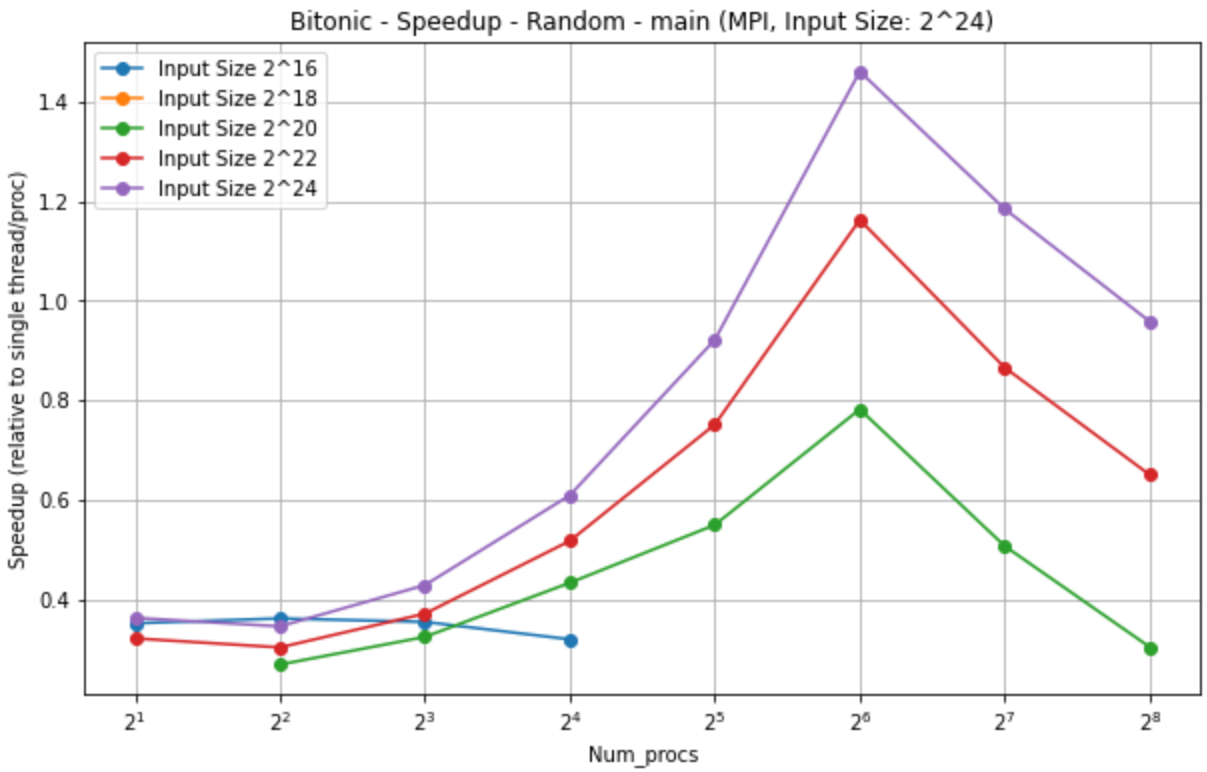

Bitonic - Speedup - Sorted - comm (MPI, Input Size: 2^24)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24

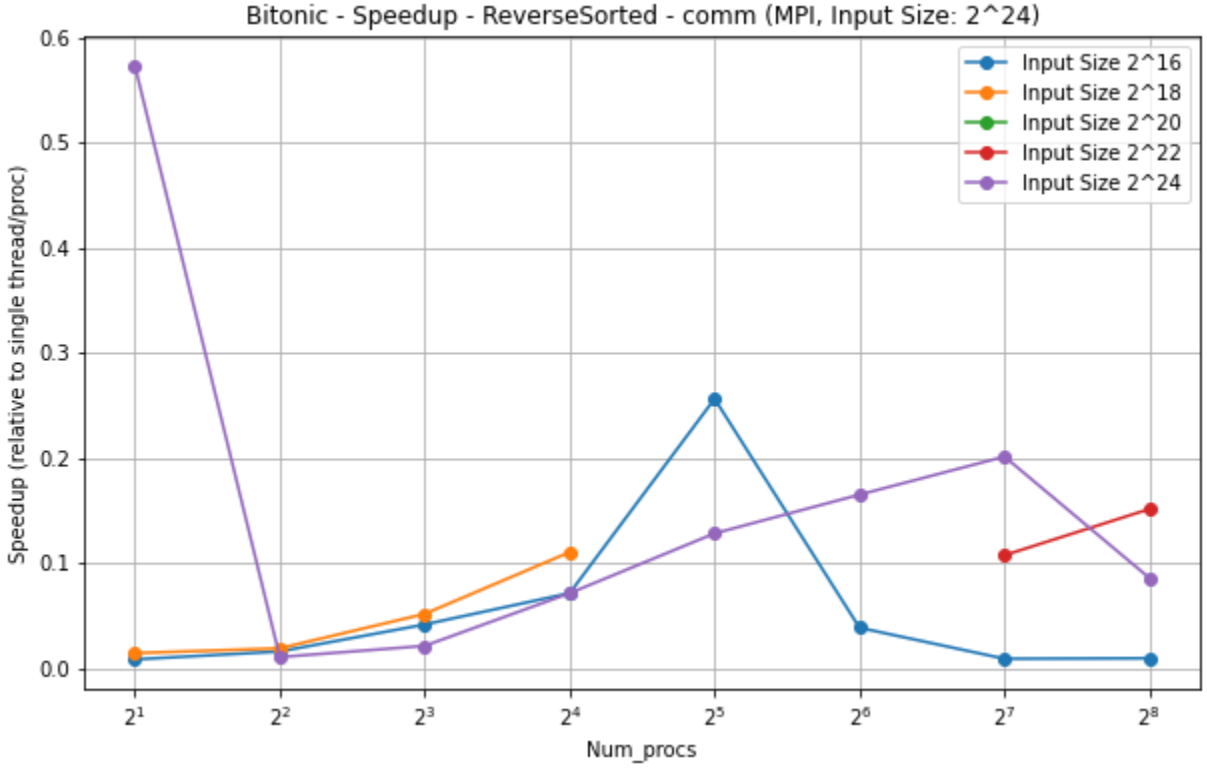

Bitonic - Speedup - Sorted - comp_large (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
Num_procs

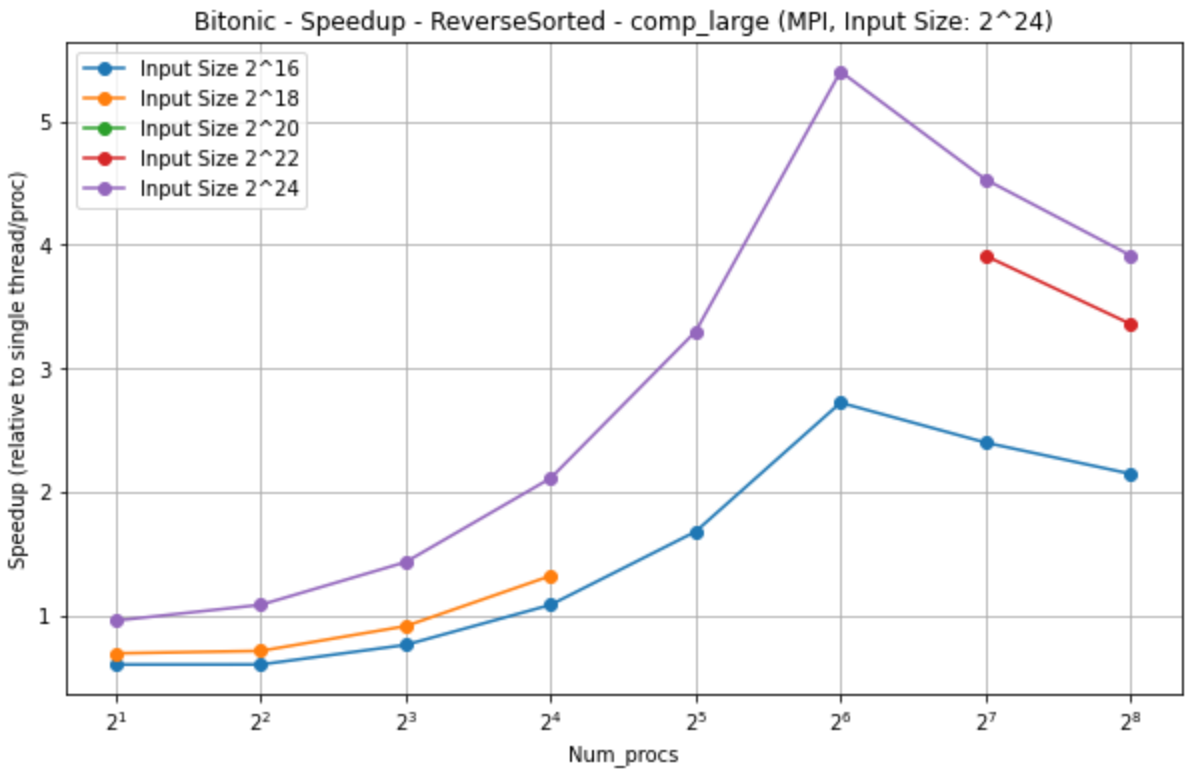

Bitonic - Speedup - Sorted - main (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
Num_procs

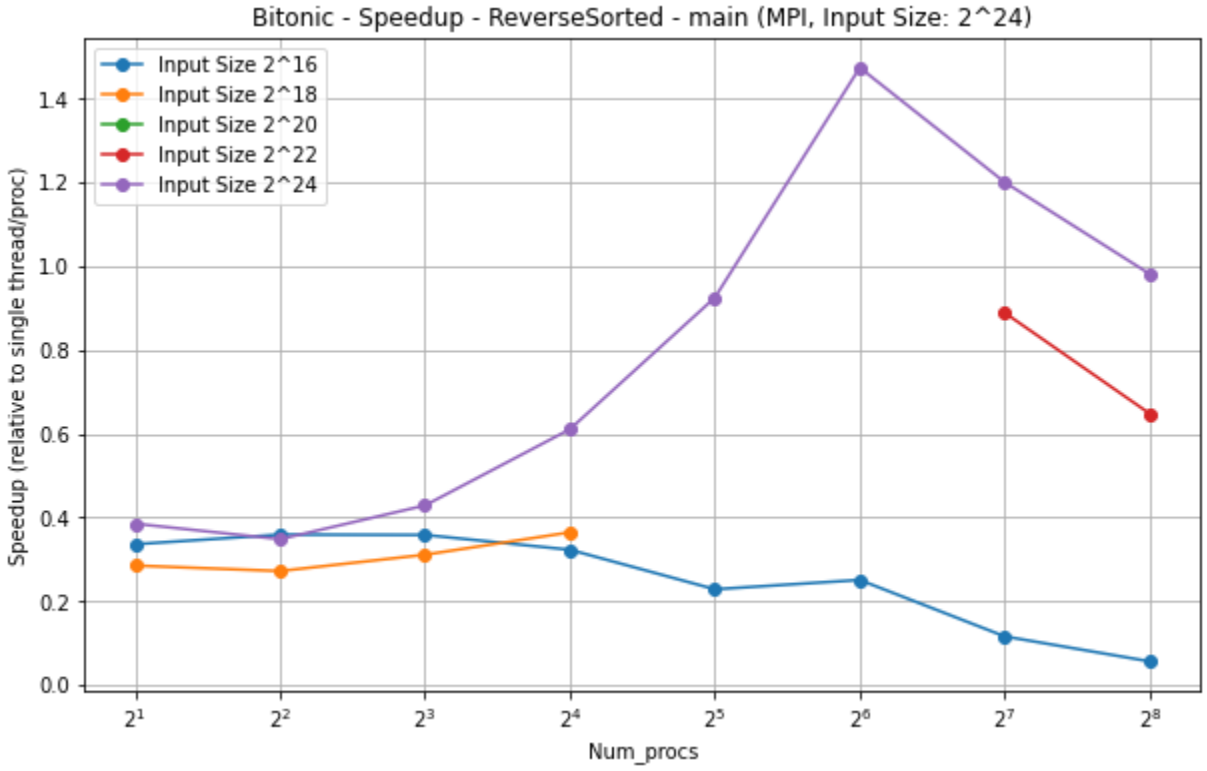

Bitonic - Speedup - Random - comm (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
0.25
0.20
0.15
0.10
0.05
0.00
2^1
2^2
2^3
2^4
2^5
2^6
2^7
2^8
Num_procs

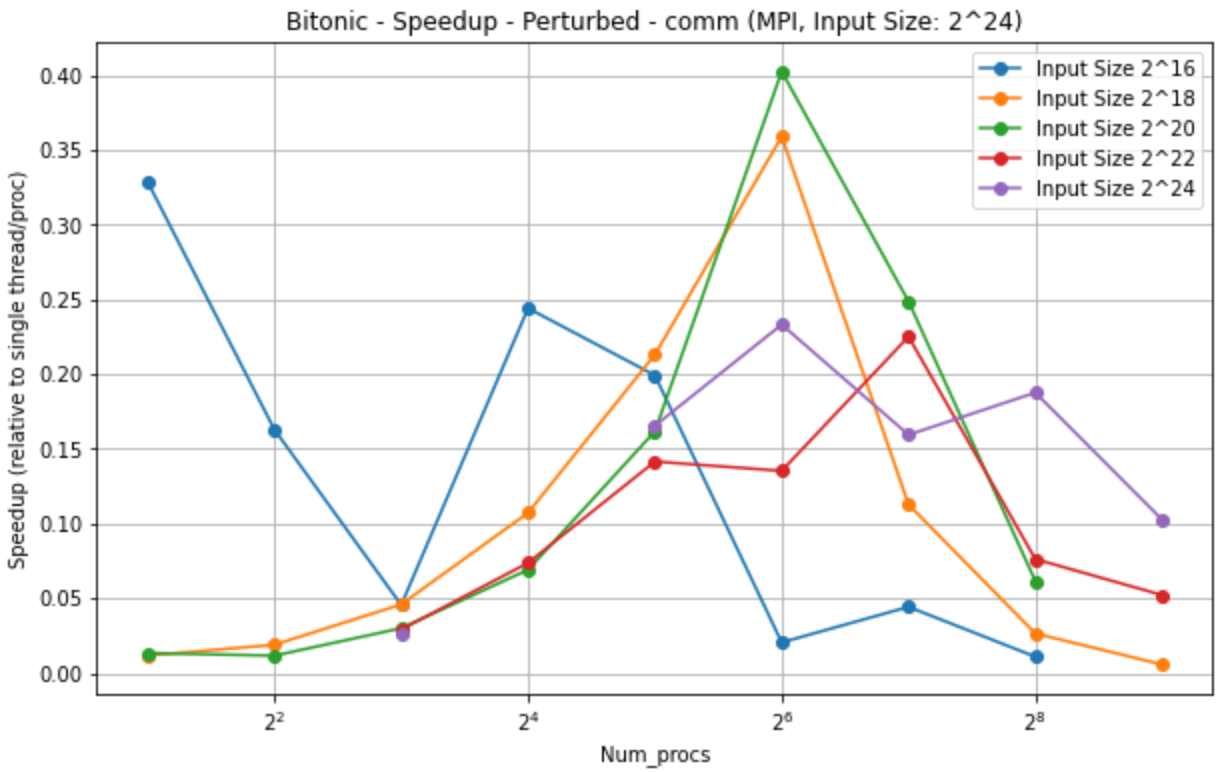

Bitonic - Speedup - Random - comp_large (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
5
4
3
2
1
2^1
2^2
2^3
2^4
2^5
2^6
2^7
2^8
Num_procs

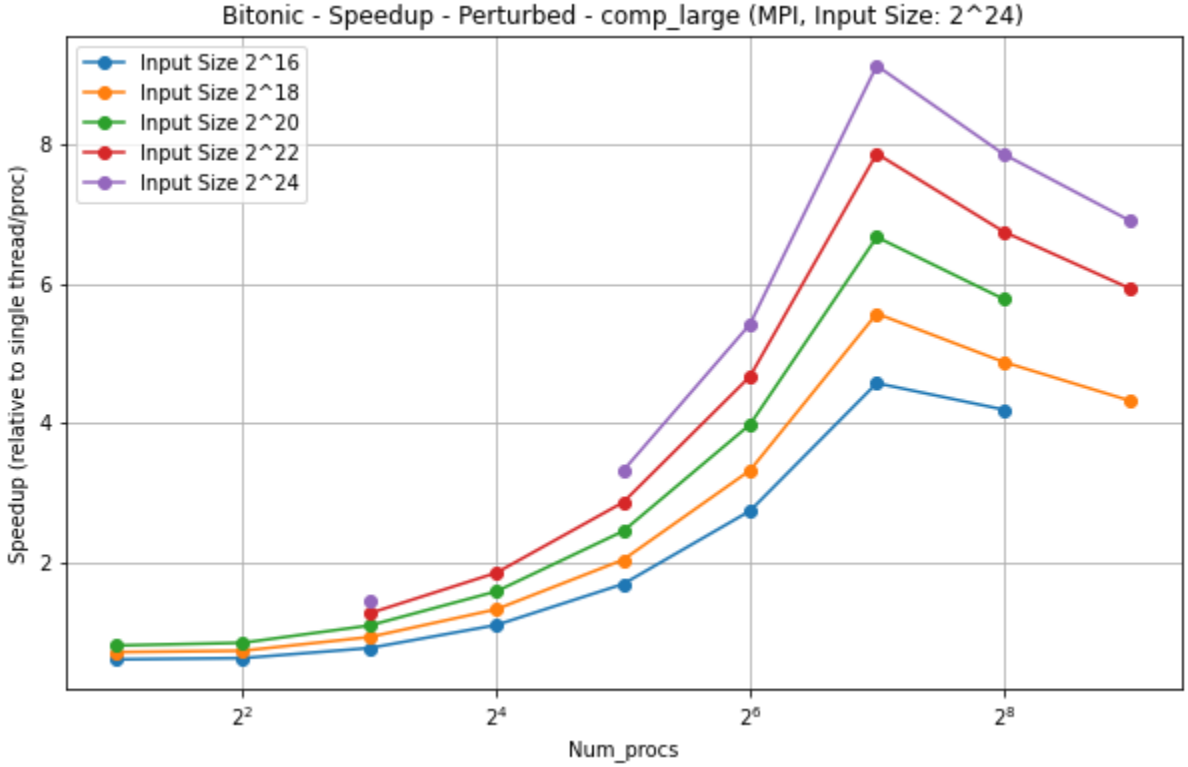

Bitonic - Speedup - Random - main (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
1.4
1.2
1.0
0.8
0.6
0.4
2^1
2^2
2^3
2^4
2^5
2^6
2^7
2^8
Num_procs

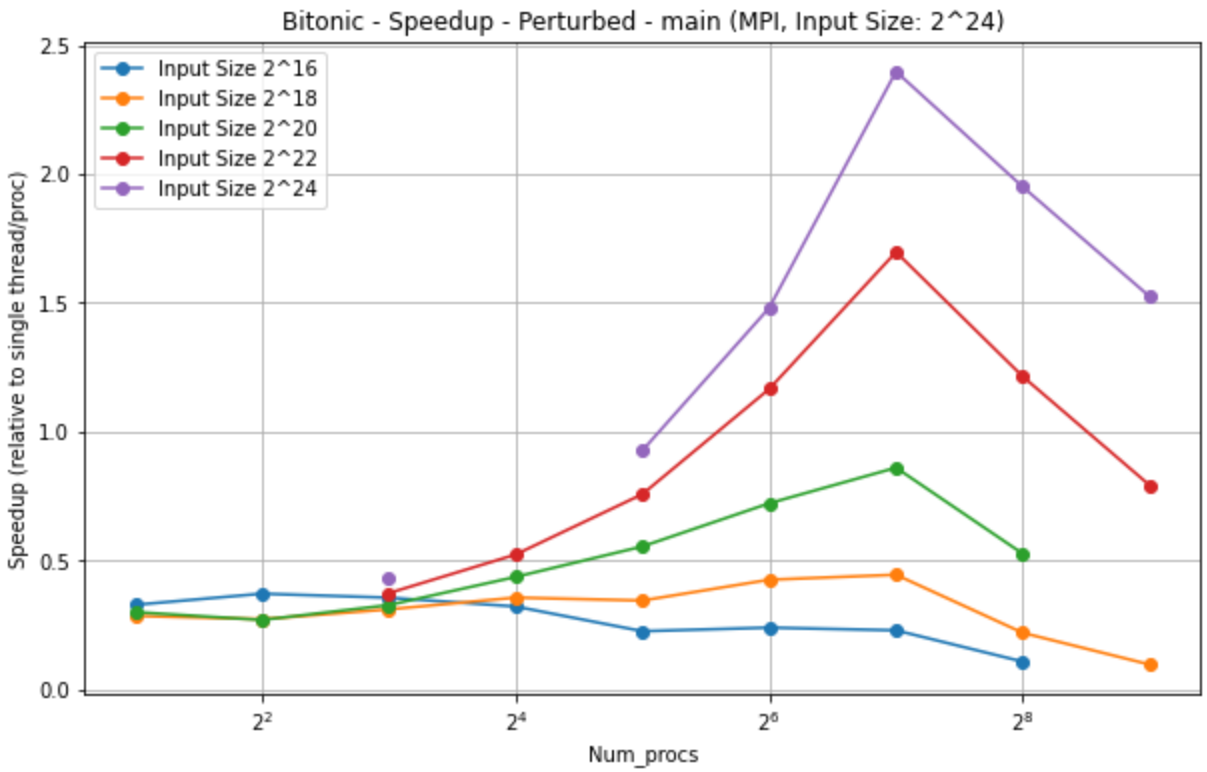

Bitonic - Speedup - ReverseSorted - comm (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2^1
2^2
2^3
2^4
2^5
2^6
2^7
2^8
Num_procs


Bitonic - Speedup - ReverseSorted - comp_large (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
Num_procs

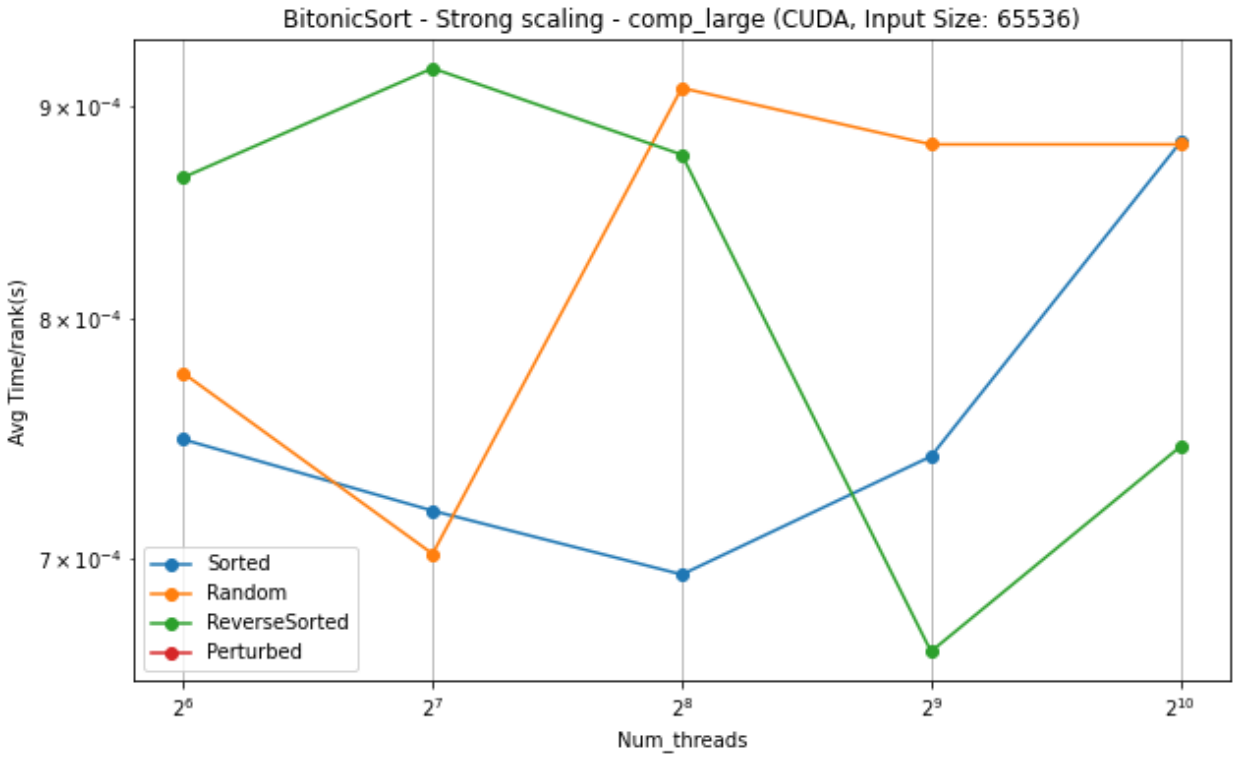

Bitonic - Speedup - ReverseSorted - main (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
Num_procs

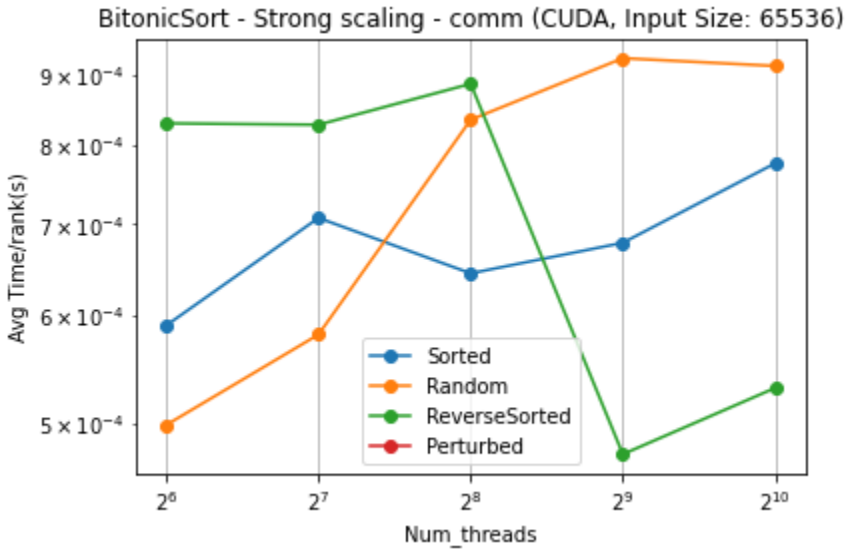

Bitonic - Speedup - Perturbed - comm (MPI, Input Size: 2^24)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
2^2
2^4
2^6
2^8

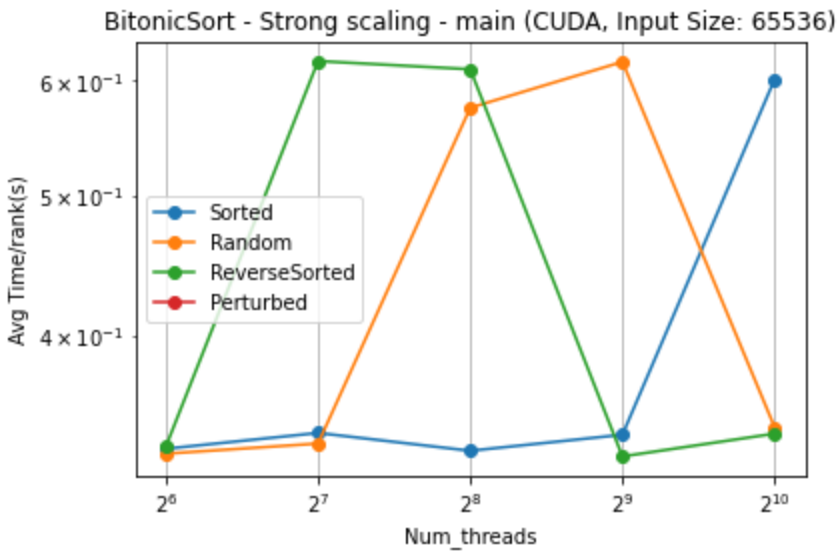

Bitonic - Speedup - Perturbed - comp_large (MPI, Input Size: 2^24)
Speedup (relative to single thread/proc)
Num_procs
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
2
4
6
8
2^2
2^4
2^6
2^8

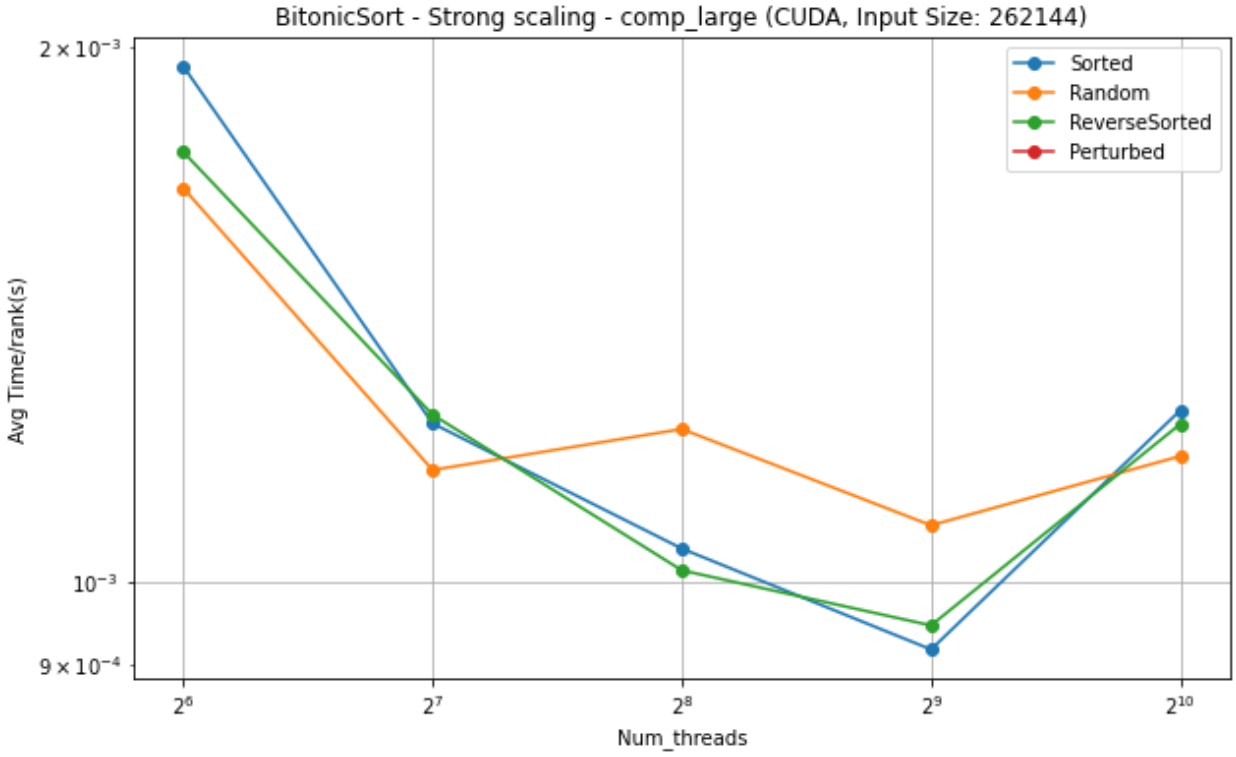

Bitonic - Speedup - Perturbed - main (MPI, Input Size: 2^24)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Speedup (relative to single thread/proc)
Num_procs
0.0
0.5
1.0
1.5
2.0
2.5
2^2
2^4
2^6
2^8

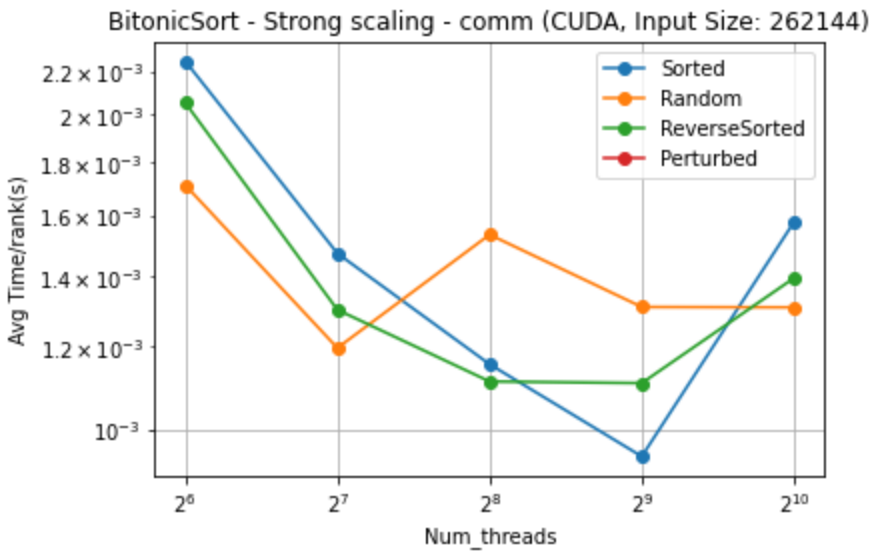
CUDA:
Strong Scaling:

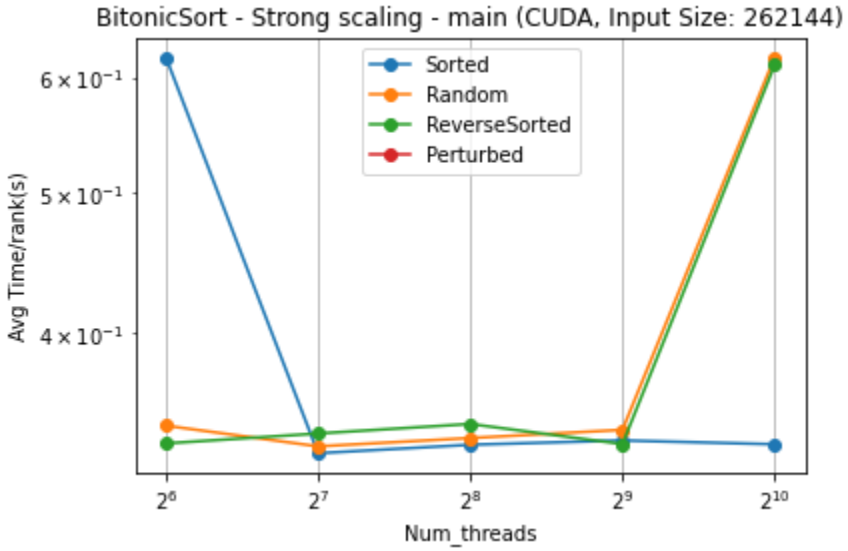


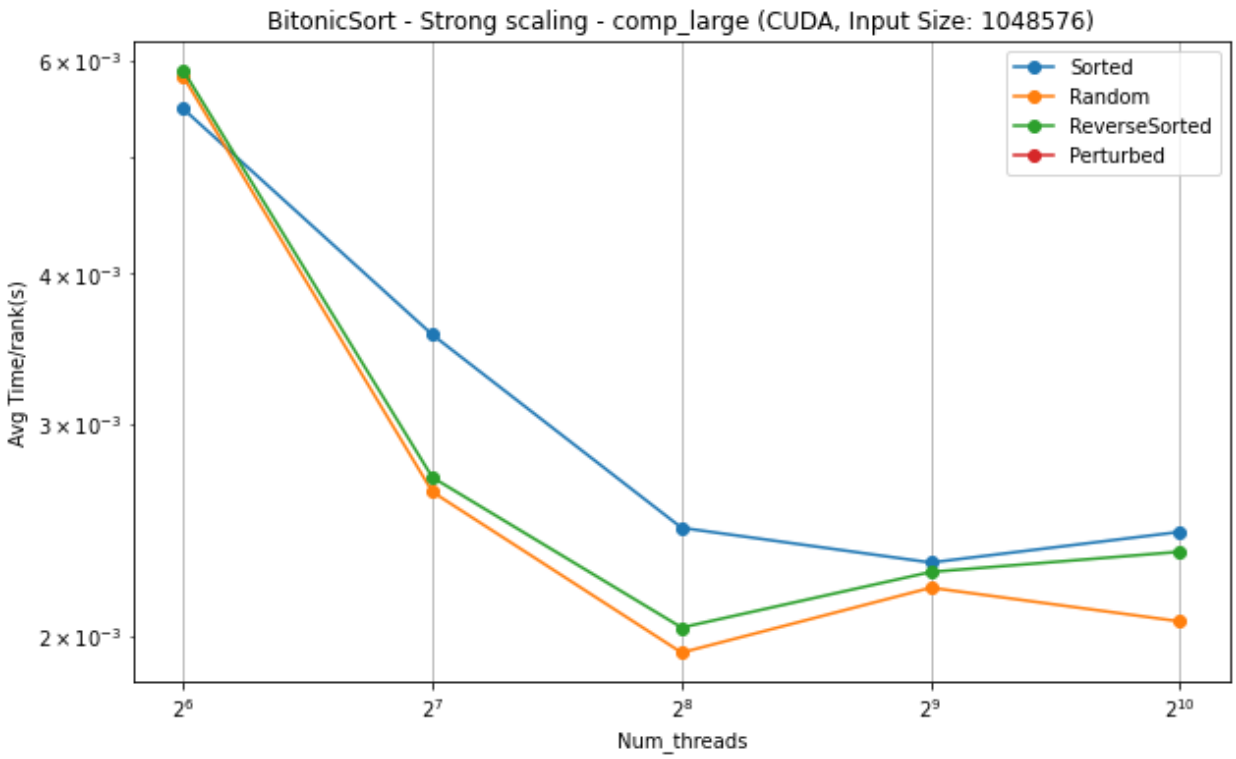


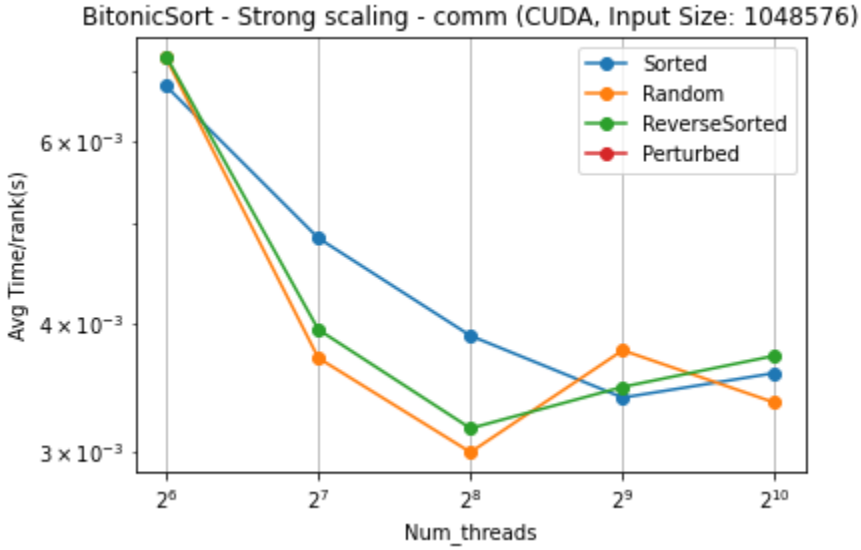

BitonicSort - Strong scaling - main (CUDA, Input Size: 65536)
Avg Time/rank(s)
6×10^{-1}
5×10^{-1}
4×10^{-1}
Sorted
Random
ReverseSorted
Perturbed
2^6
2^7
2^8
2^9
2^{10}
Num_threads

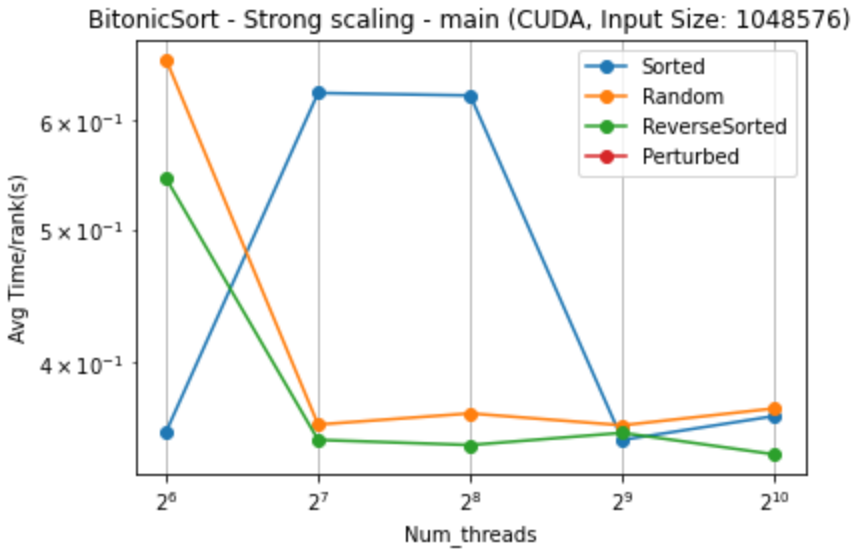

BitonicSort - Strong scaling - comp_large (CUDA, Input Size: 262144)
Avg Time/rank(s)
2×10^{-3}
10^{-3}
9×10^{-4}
Sorted
Random
ReverseSorted
Perturbed
2^6
2^7
2^8
2^9
2^{10}
Num_threads

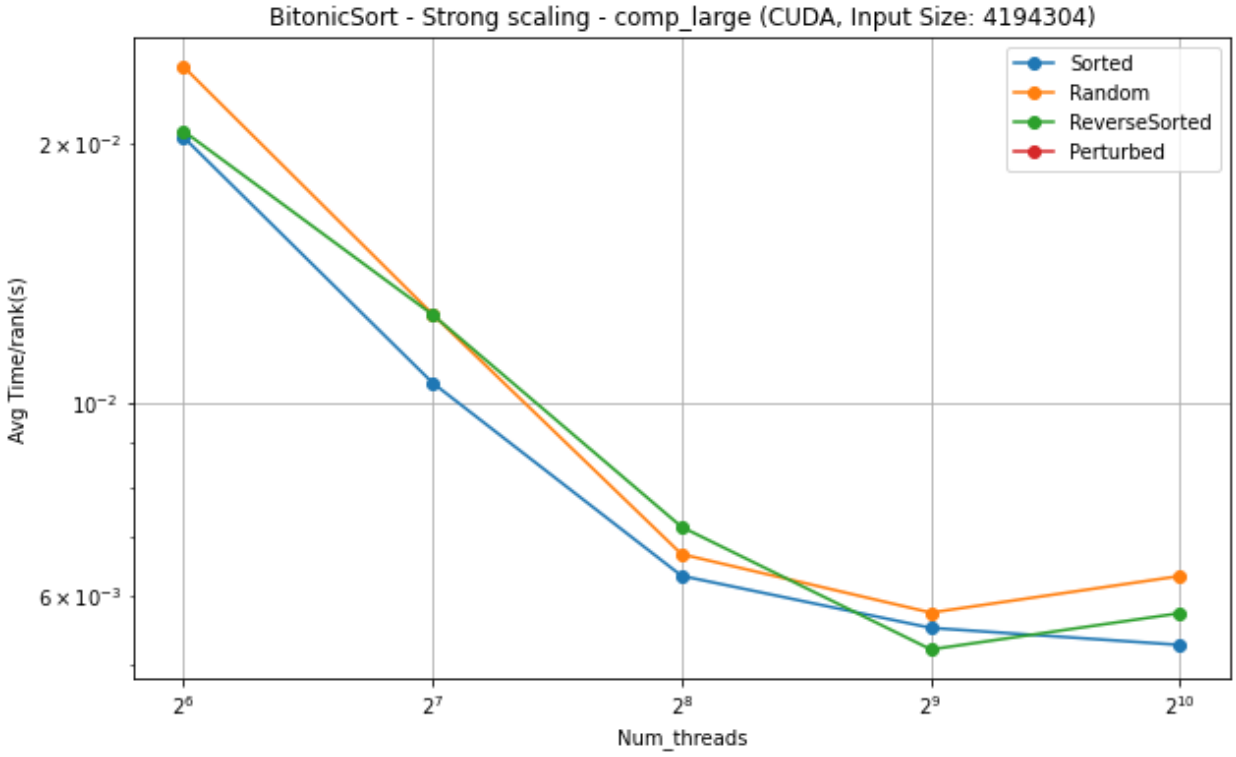

BitonicSort - Strong scaling - comm (CUDA, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_threads

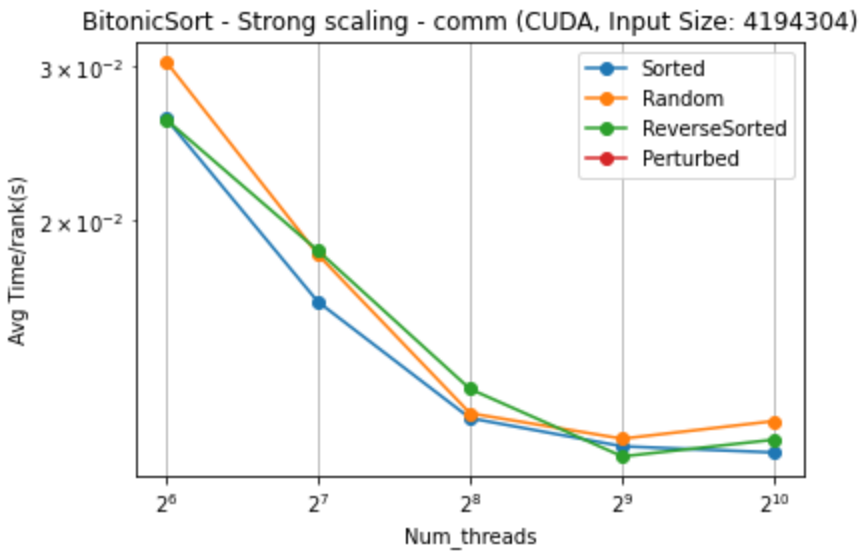

BitonicSort - Strong scaling - main (CUDA, Input Size: 262144)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
Num_threads

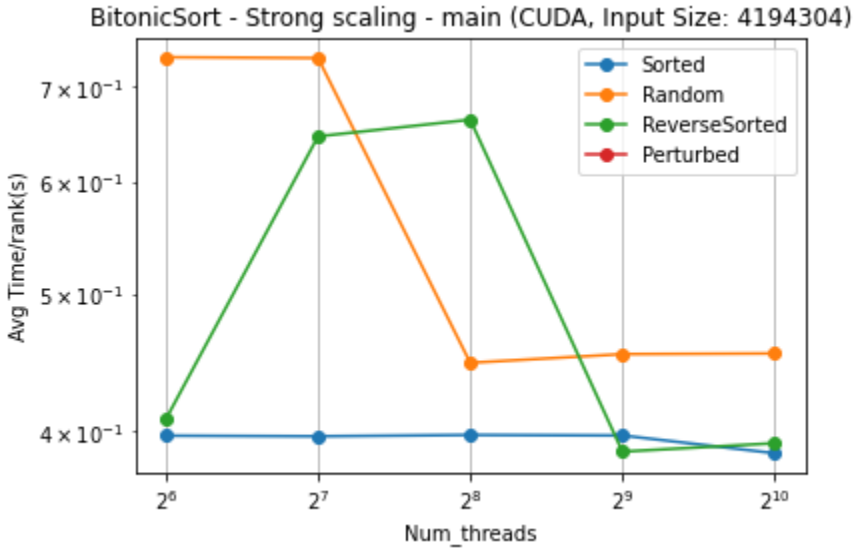

BitonicSort - Strong scaling - comp_large (CUDA, Input Size: 1048576)
Avg Time/rank(s)
Num_threads
Sorted
Random
ReverseSorted
Perturbed

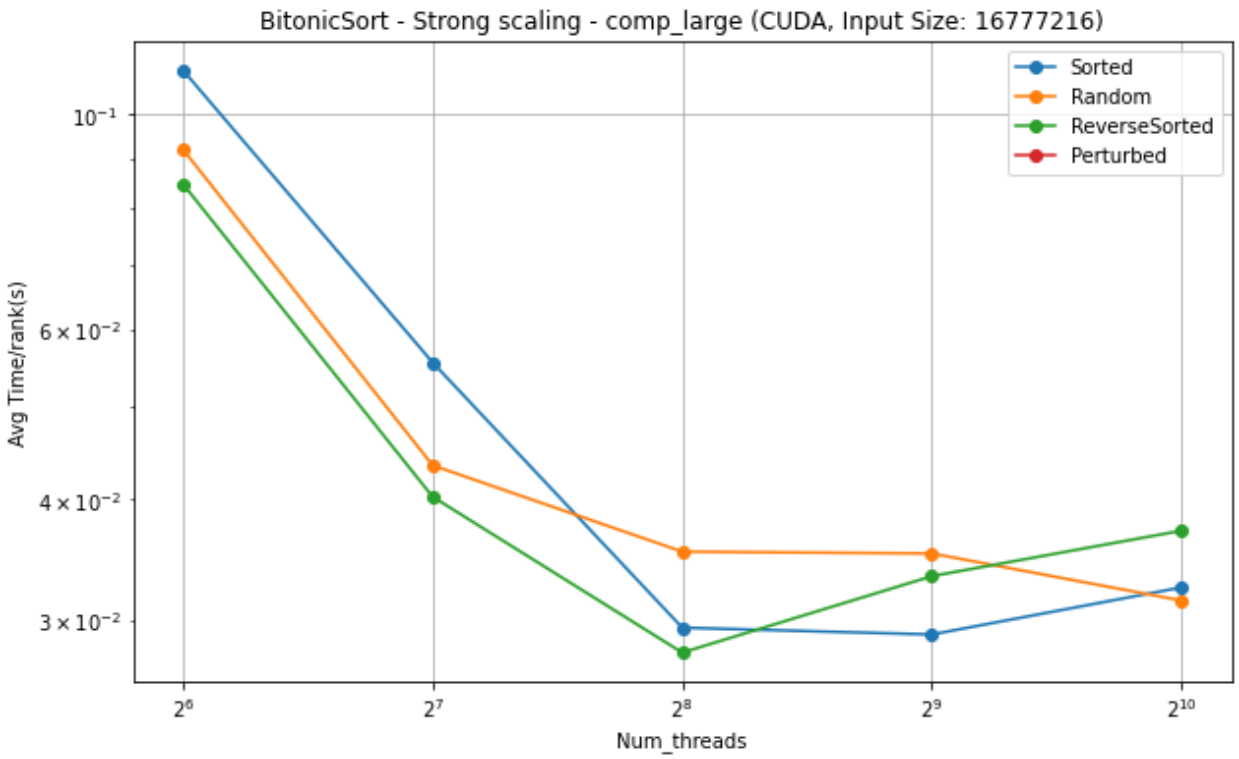

BitonicSort - Strong scaling - comm (CUDA, Input Size: 1048576)
Avg Time/rank(s)
Num_threads
Sorted
Random
ReverseSorted
Perturbed

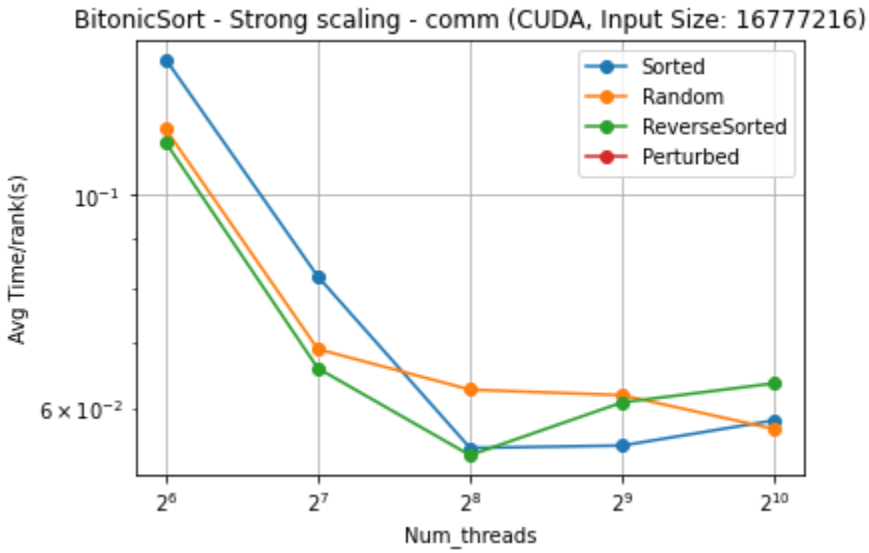

BitonicSort - Strong scaling - main (CUDA, Input Size: 1048576)
Sorted
Random
ReverseSorted
Perturbed
6×10^{-1}
5×10^{-1}
4×10^{-1}
Avg Time/rank(s)
2^6
2^7
2^8
2^9
2^{10}
Num_threads

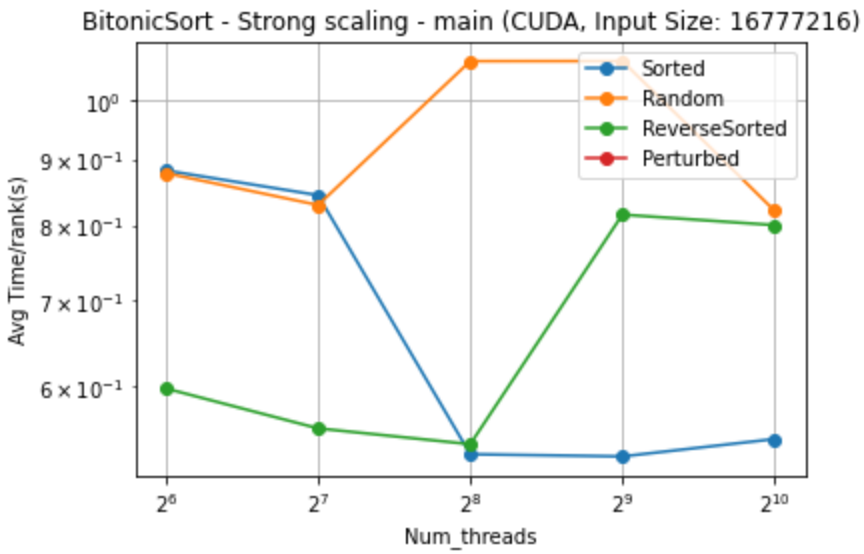

BitonicSort - Strong scaling - comp_large (CUDA, Input Size: 4194304)
Sorted
Random
ReverseSorted
Perturbed
2×10^{-2}
10^{-2}
6×10^{-3}
Avg Time/rank(s)
2^6
2^7
2^8
2^9
2^{10}
Num_threads

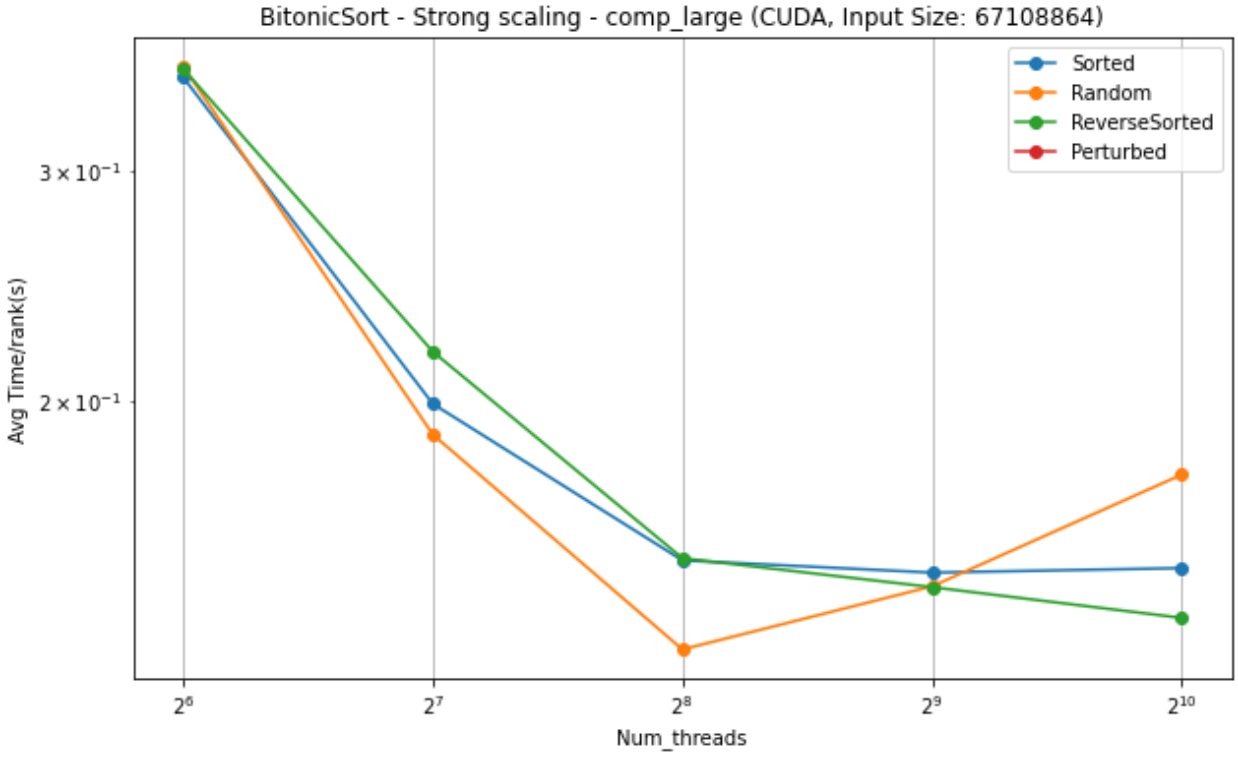

BitonicSort - Strong scaling - comm (CUDA, Input Size: 4194304)
Avg Time/rank(s)
3×10^{-2}
2×10^{-2}
Sorted
Random
ReverseSorted
Perturbed
2^6
2^7
2^8
2^9
2^{10}
Num_threads

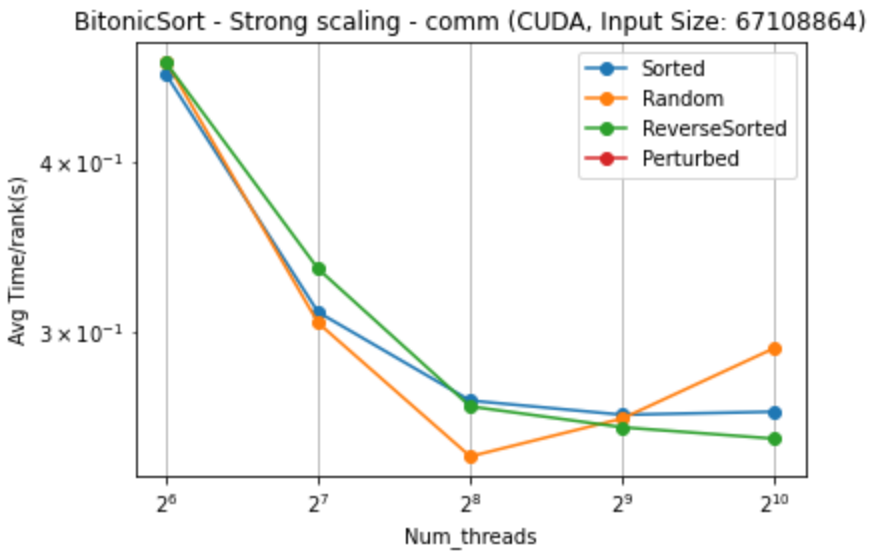

BitonicSort - Strong scaling - main (CUDA, Input Size: 4194304)
Avg Time/rank(s)
7×10^{-1}
6×10^{-1}
5×10^{-1}
4×10^{-1}
Sorted
Random
ReverseSorted
Perturbed
2^6
2^7
2^8
2^9
2^{10}
Num_threads

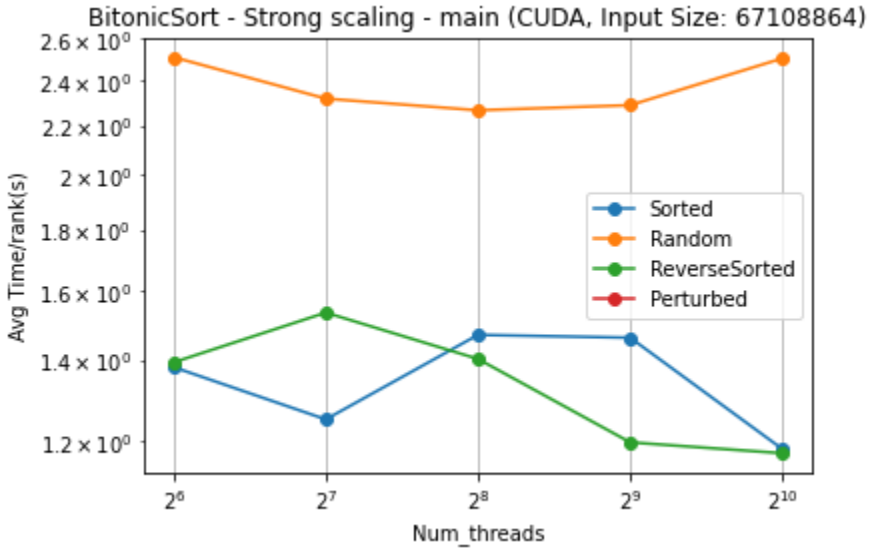

BitonicSort - Strong scaling - comp_large (CUDA, Input Size: 16777216)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{-1}
6×10^{-2}
4×10^{-2}
3×10^{-2}
2^6
2^7
2^8
2^9
2^{10}
Num_threads

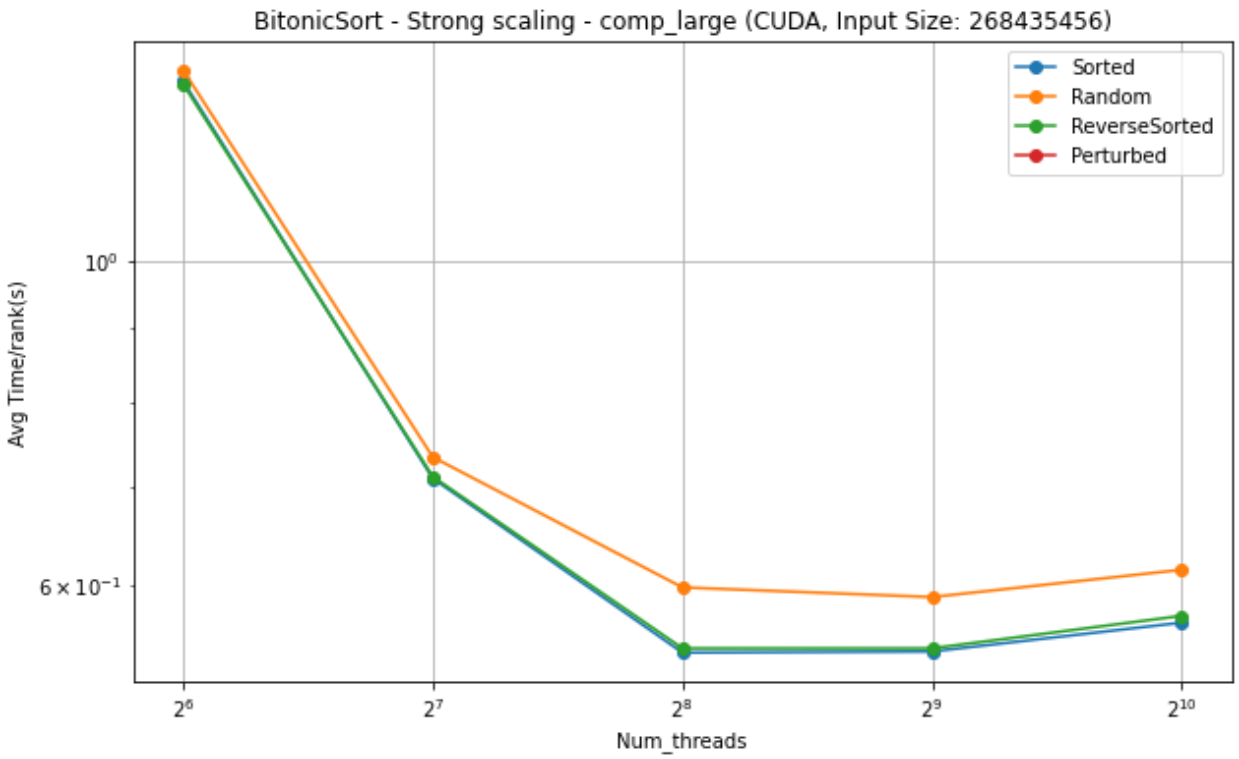

BitonicSort - Strong scaling - comm (CUDA, Input Size: 16777216)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^{-1}
6×10^{-2}
2^6
2^7
2^8
2^9
2^{10}
Num_threads

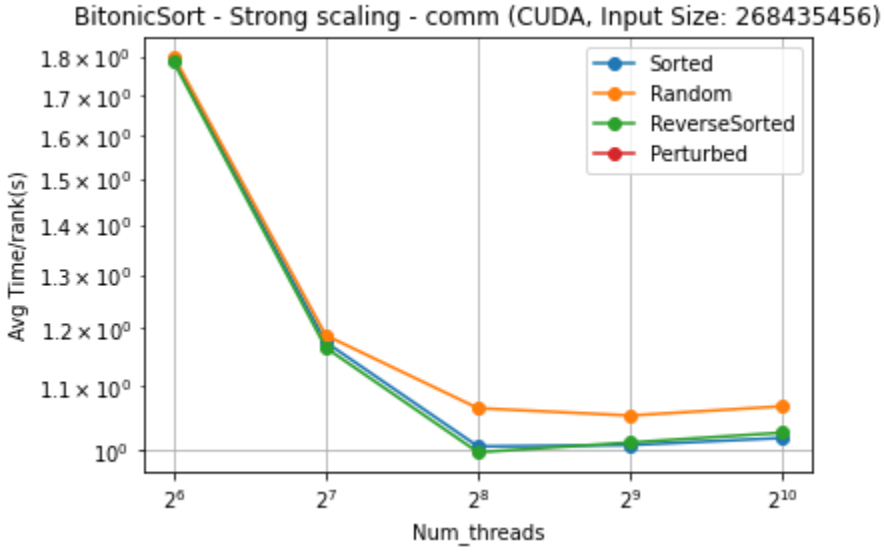

BitonicSort - Strong scaling - main (CUDA, Input Size: 16777216)
Sorted
Random
ReverseSorted
Perturbed
10^{0}
9×10^{-1}
8×10^{-1}
7×10^{-1}
6×10^{-1}
Avg Time/rank(s)
2^{6}
2^{7}
2^{8}
2^{9}
2^{10}
Num_threads

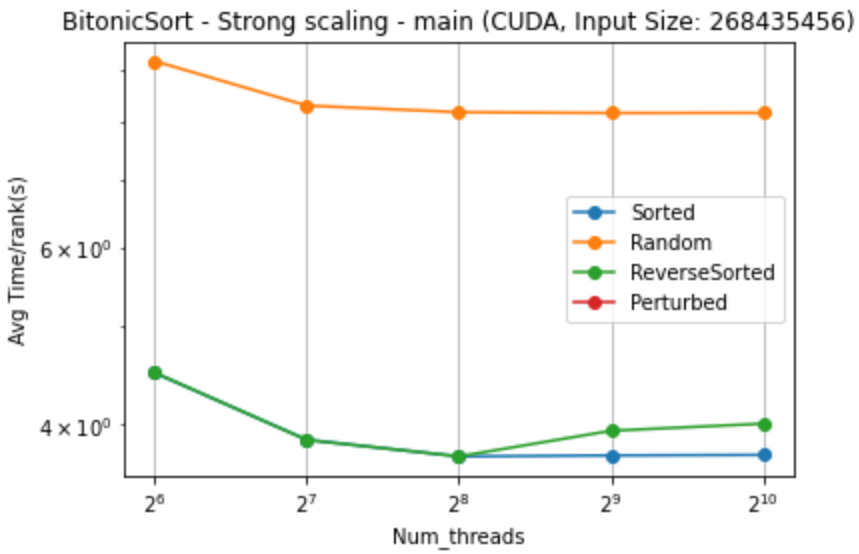

BitonicSort - Strong scaling - comp_large (CUDA, Input Size: 67108864)
Sorted
Random
ReverseSorted
Perturbed
3×10^{-1}
2×10^{-1}
Avg Time/rank(s)
2^{6}
2^{7}
2^{8}
2^{9}
2^{10}
Num_threads


BitonicSort - Strong scaling - comm (CUDA, Input Size: 67108864)
Avg Time/rank(s)
4×10^{-1}
3×10^{-1}
2^6
2^7
2^8
2^9
2^{10}
Num_threads
Sorted
Random
ReverseSorted
Perturbed

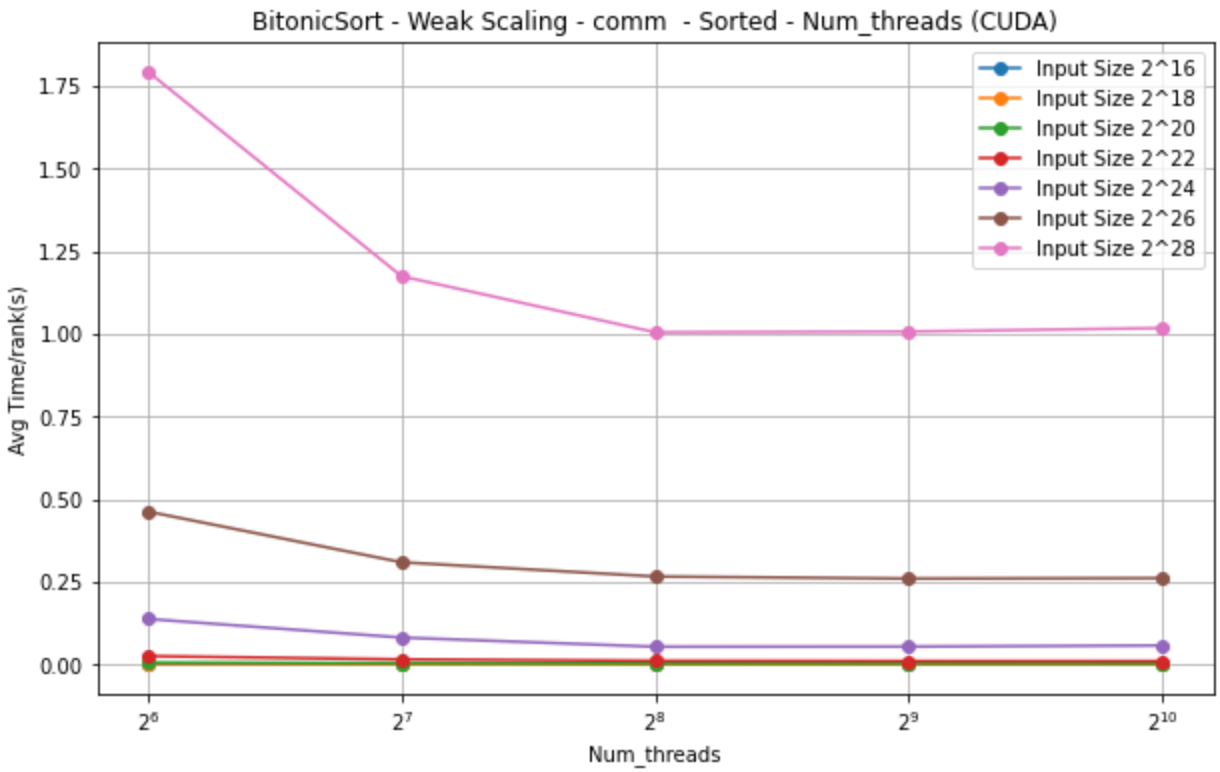

BitonicSort - Strong scaling - main (CUDA, Input Size: 67108864)
Avg Time/rank(s)
2.6×10^0
2.4×10^0
2.2×10^0
2×10^0
1.8×10^0
1.6×10^0
1.4×10^0
1.2×10^0
2^6
2^7
2^8
2^9
2^{10}
Num_threads
Sorted
Random
ReverseSorted
Perturbed

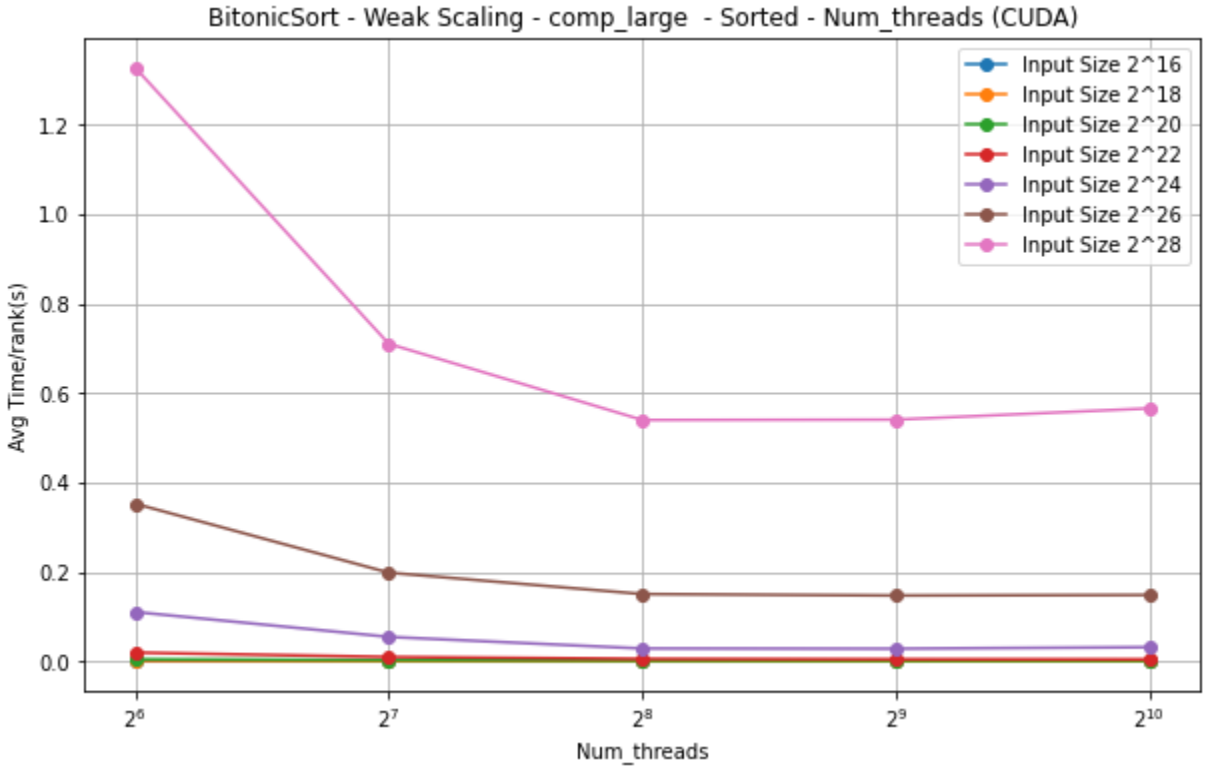

BitonicSort - Strong scaling - comp_large (CUDA, Input Size: 268435456)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
10^0
6×10^{-1}
2^6
2^7
2^8
2^9
2^{10}
Num_threads

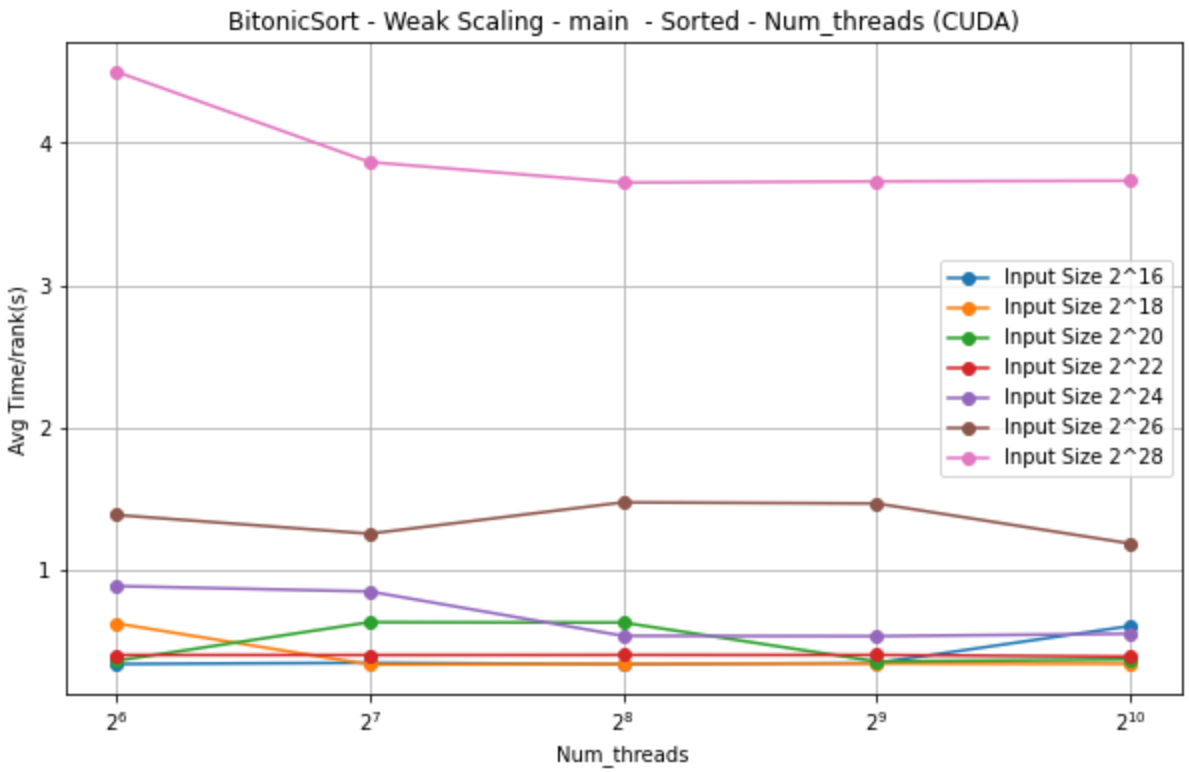

BitonicSort - Strong scaling - comm (CUDA, Input Size: 268435456)
Sorted
Random
ReverseSorted
Perturbed
Avg Time/rank(s)
1.8×10^0
1.7×10^0
1.6×10^0
1.5×10^0
1.4×10^0
1.3×10^0
1.2×10^0
1.1×10^0
10^0
2^6
2^7
2^8
2^9
2^{10}
Num_threads

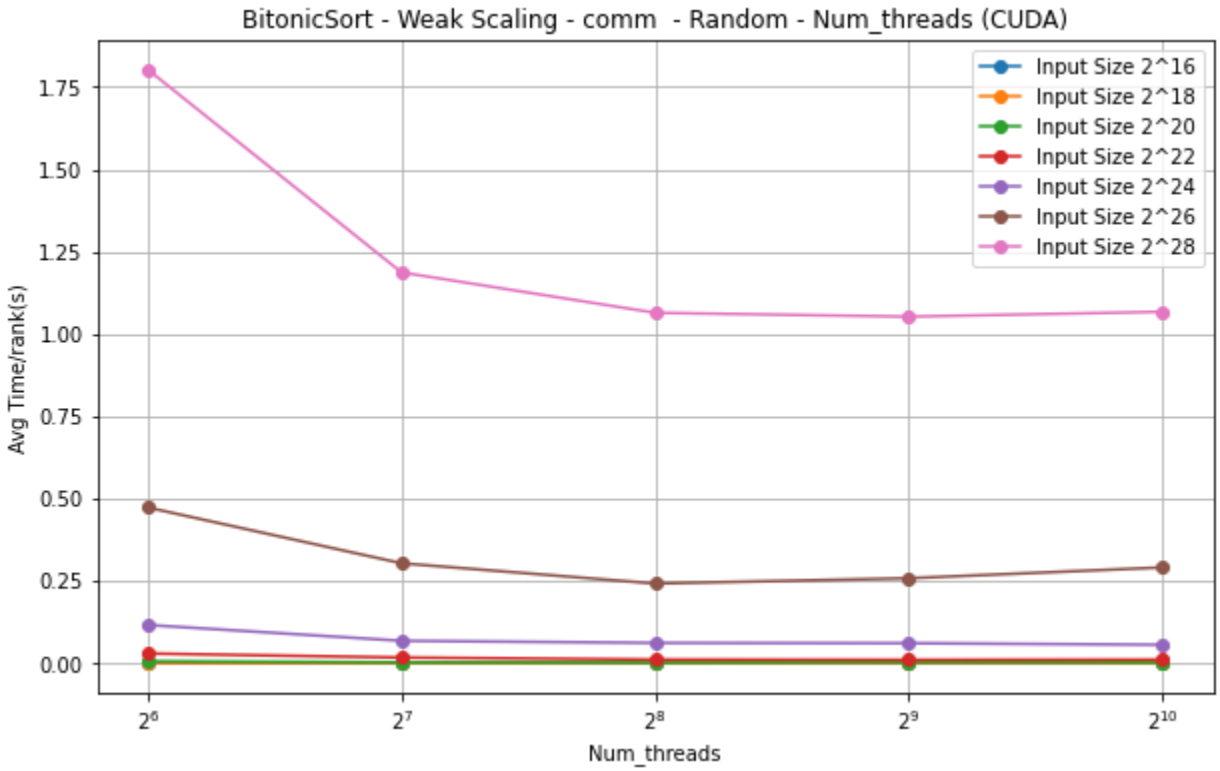


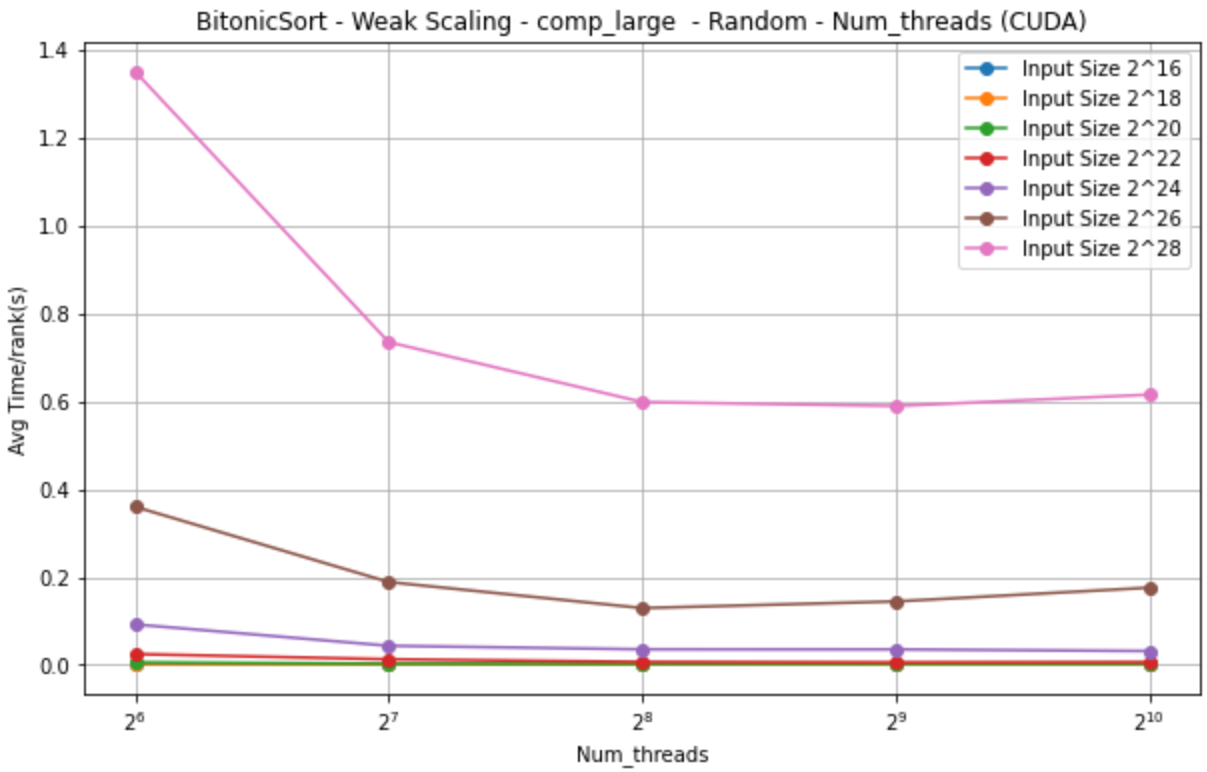
Weak Scaling:

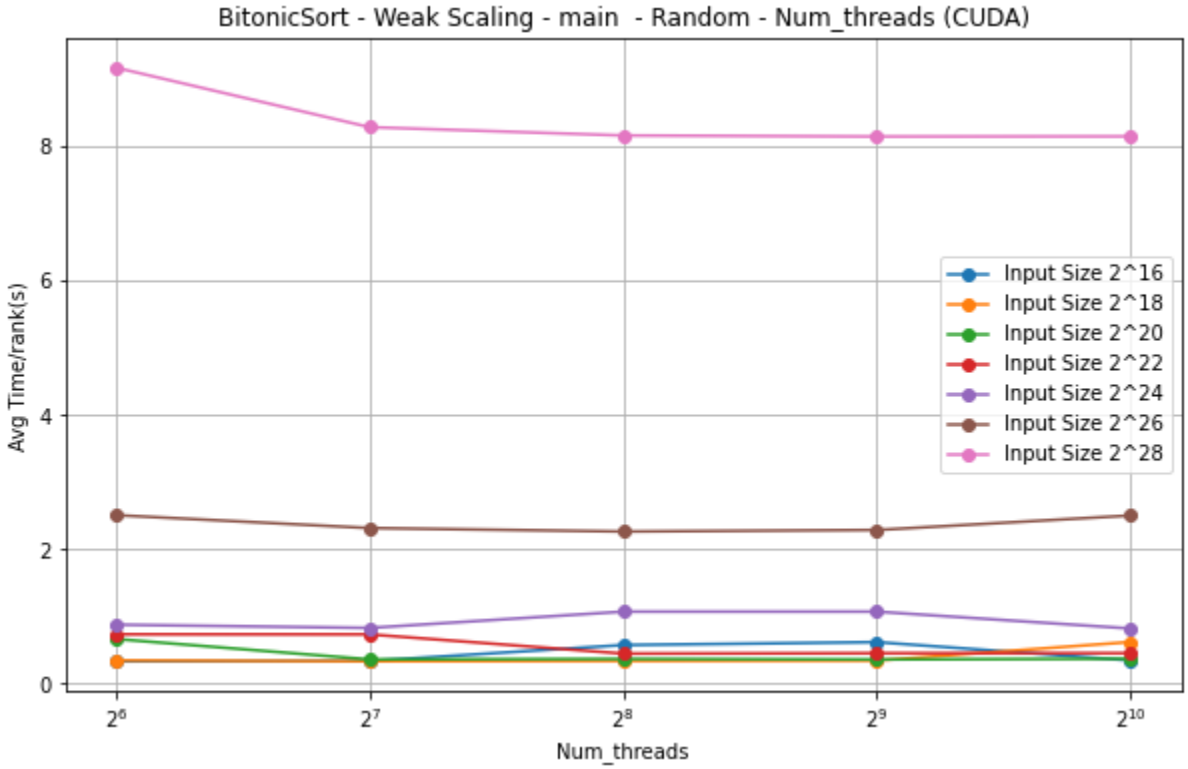

BitonicSort - Weak Scaling - comm - Sorted - Num_threads (CUDA)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2^6
2^7
2^8
2^9
2^{10}
Num_threads

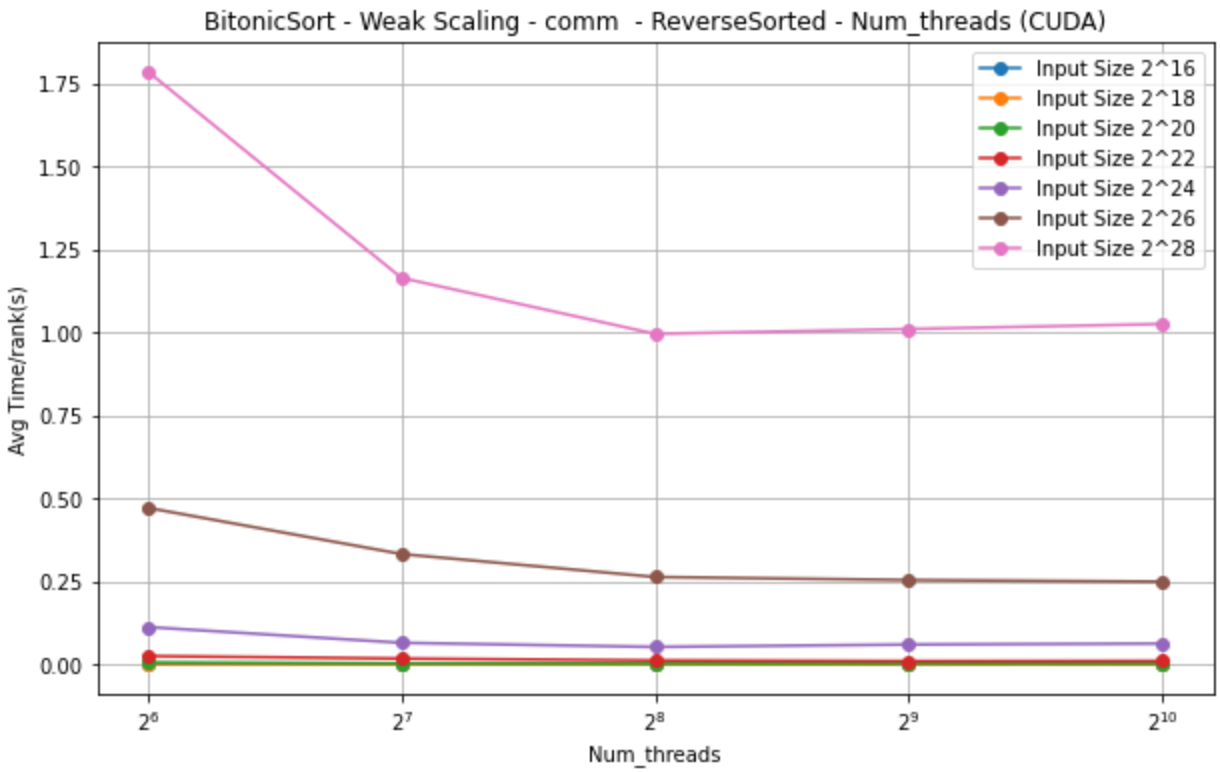

BitonicSort - Weak Scaling - comp_large - Sorted - Num_threads (CUDA)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2^6
2^7
2^8
2^9
2^{10}
Num_threads

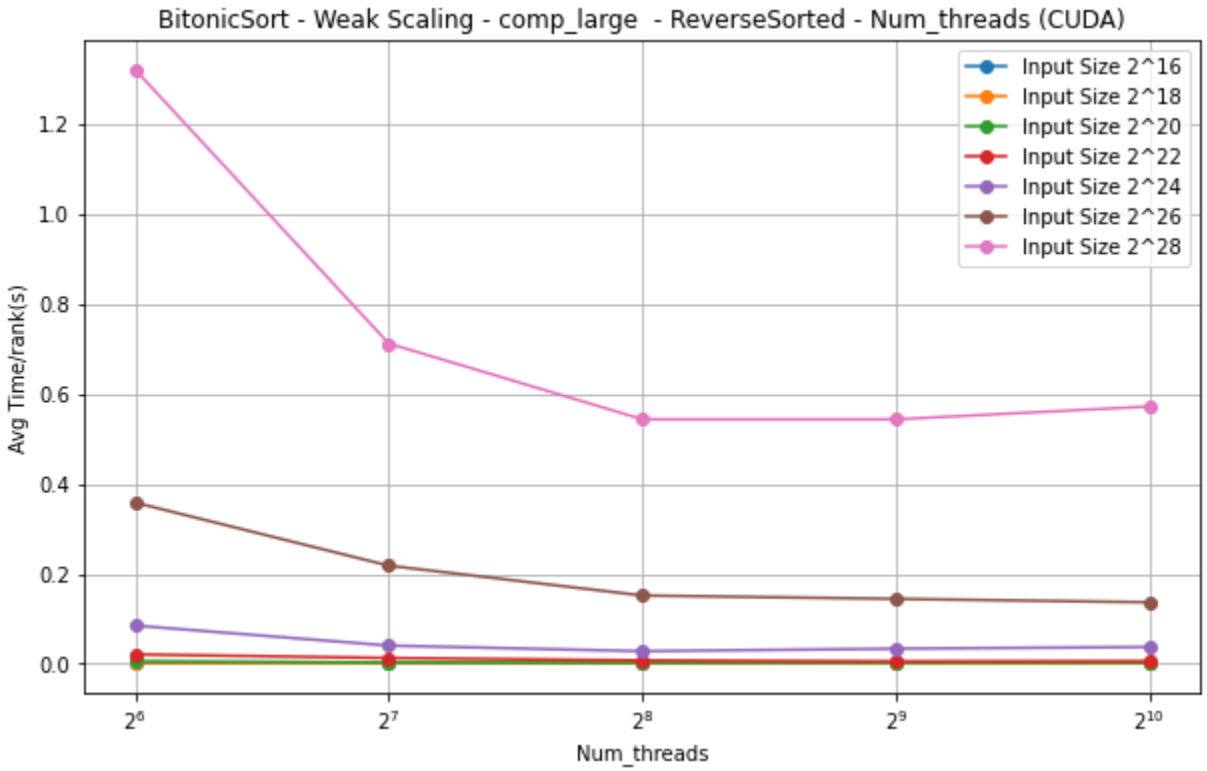

BitonicSort - Weak Scaling - main - Sorted - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

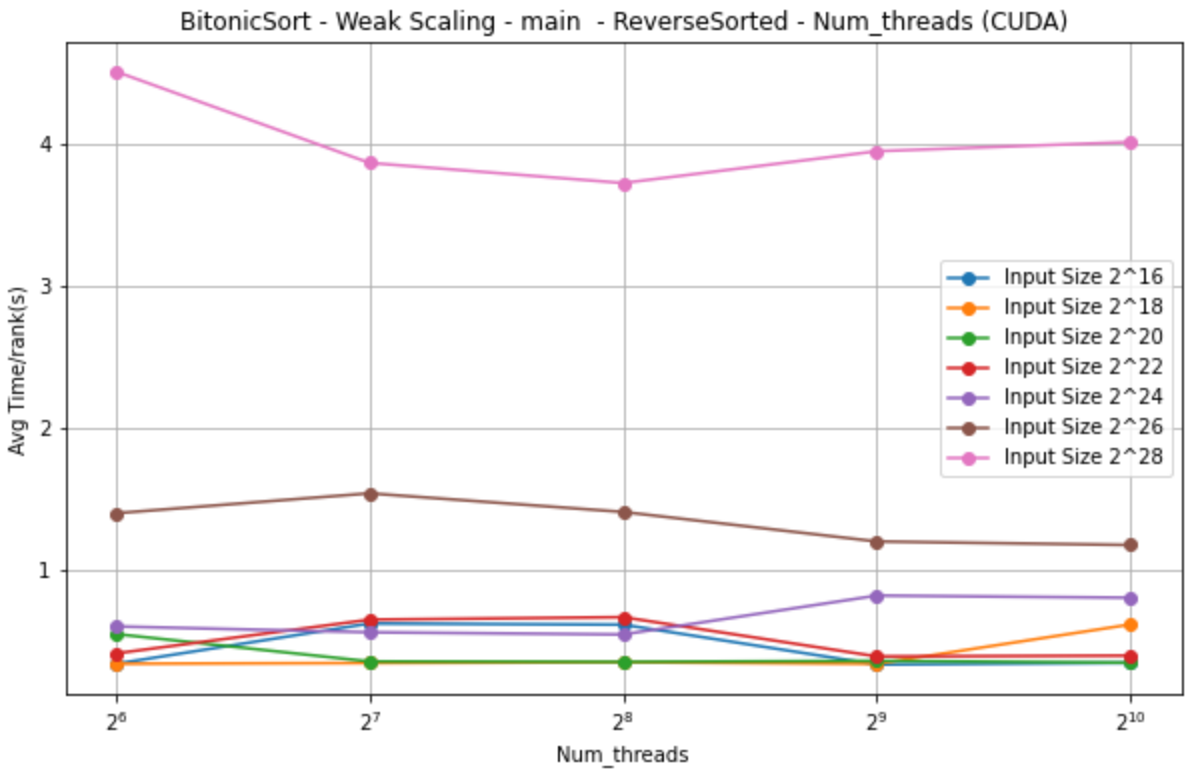

BitonicSort - Weak Scaling - comm - Random - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28


BitonicSort - Weak Scaling - comp_large - Random - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

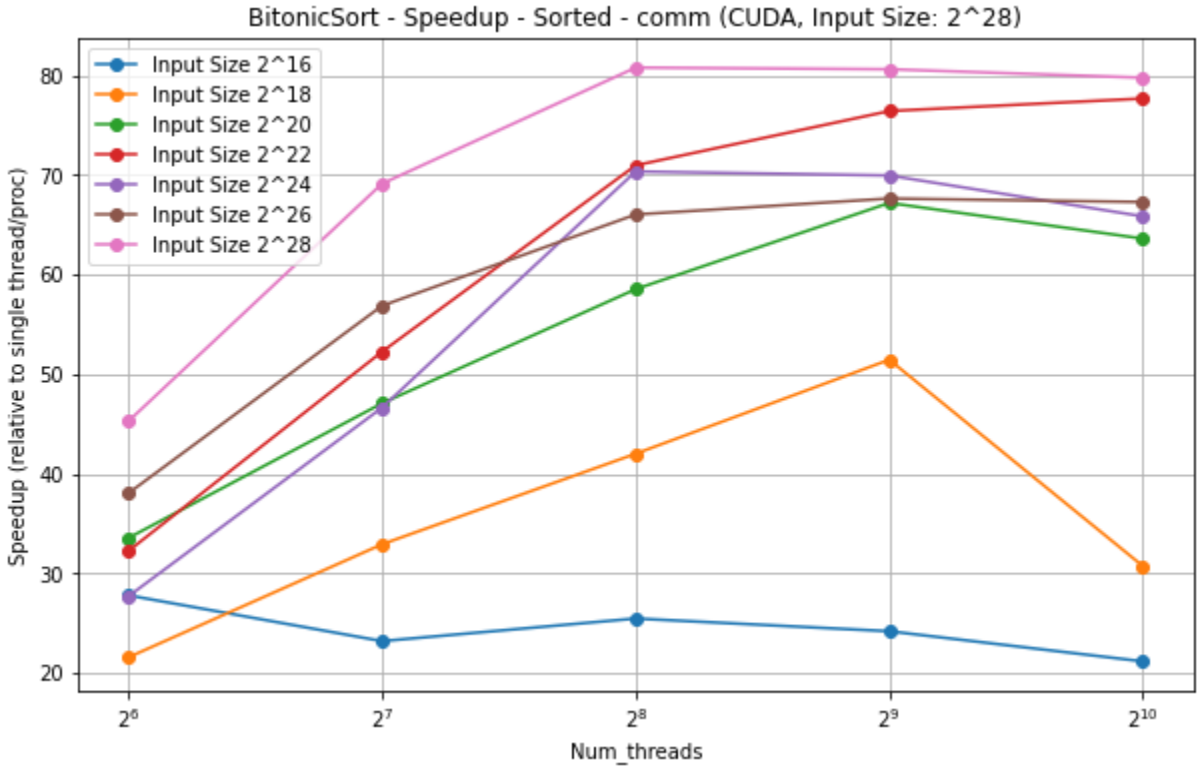

BitonicSort - Weak Scaling - main - Random - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

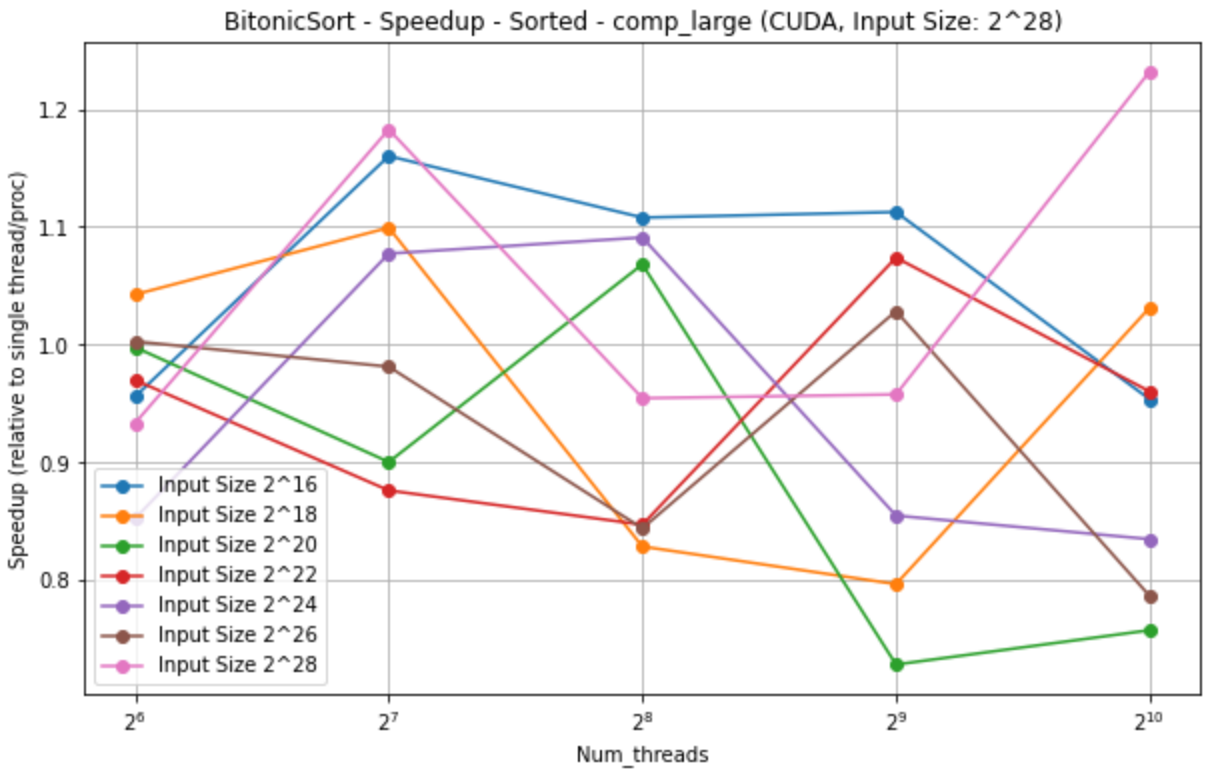

BitonicSort - Weak Scaling - comm - ReverseSorted - Num_threads (CUDA)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2^6
2^7
2^8
2^9
2^{10}
Num_threads

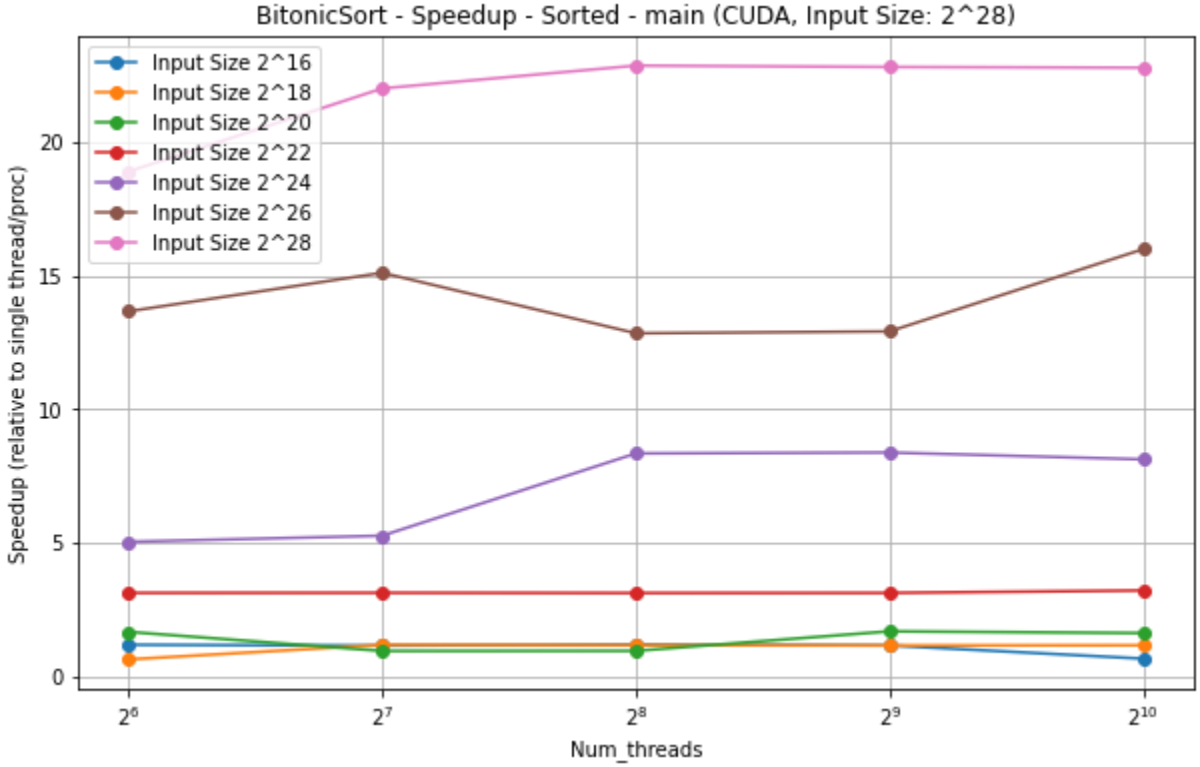

BitonicSort - Weak Scaling - comp_large - ReverseSorted - Num_threads (CUDA)
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
Avg Time/rank(s)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2^6
2^7
2^8
2^9
2^{10}
Num_threads

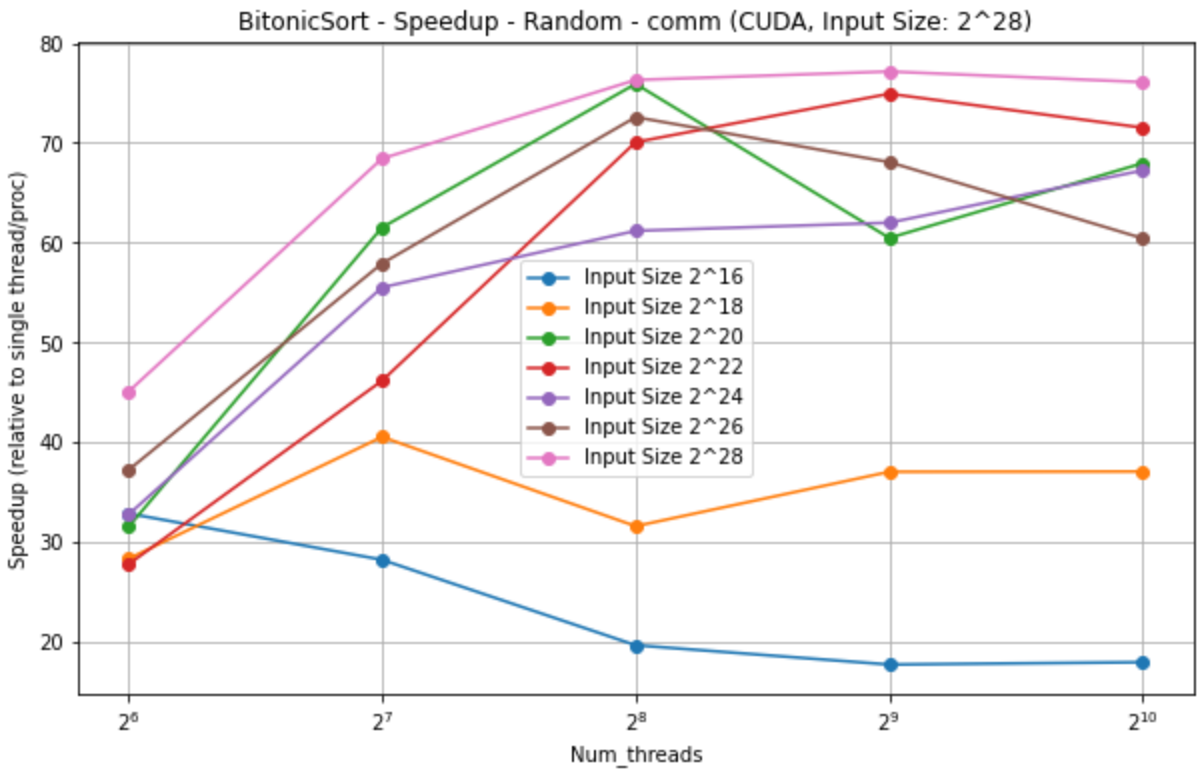

BitonicSort - Weak Scaling - main - ReverseSorted - Num_threads (CUDA)
Avg Time/rank(s)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

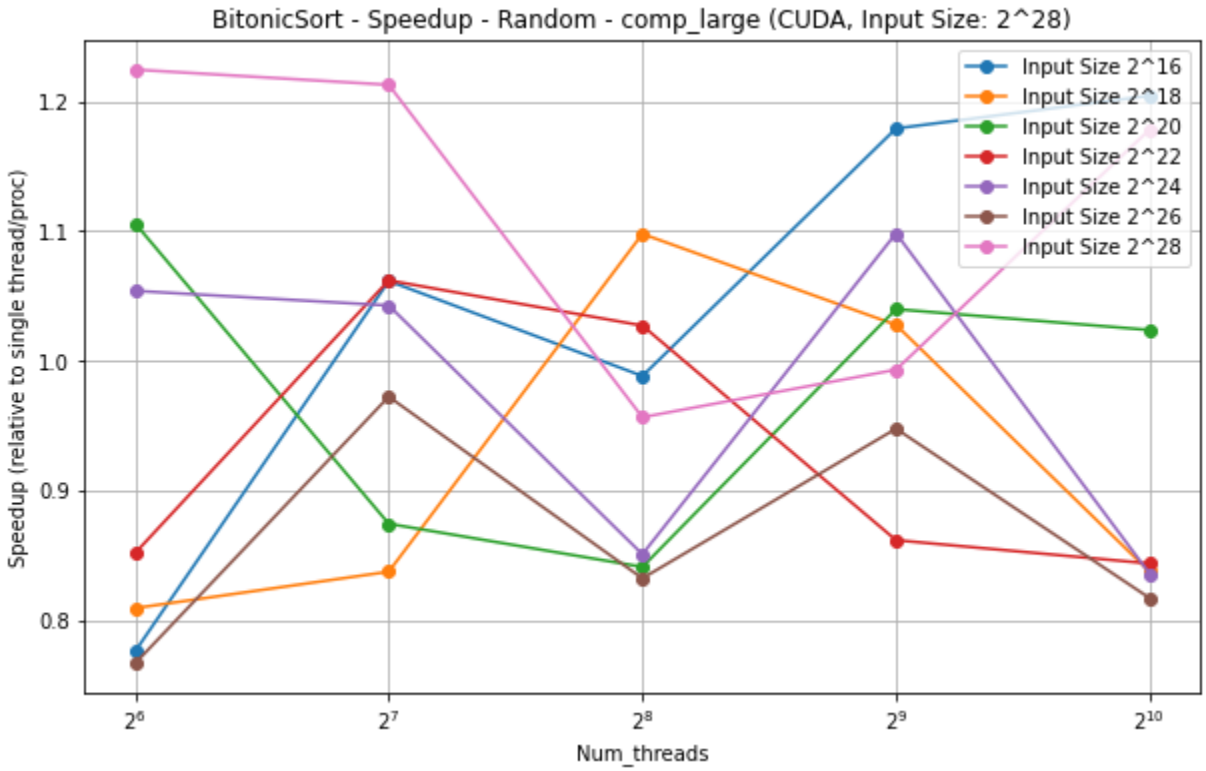
Speedup:

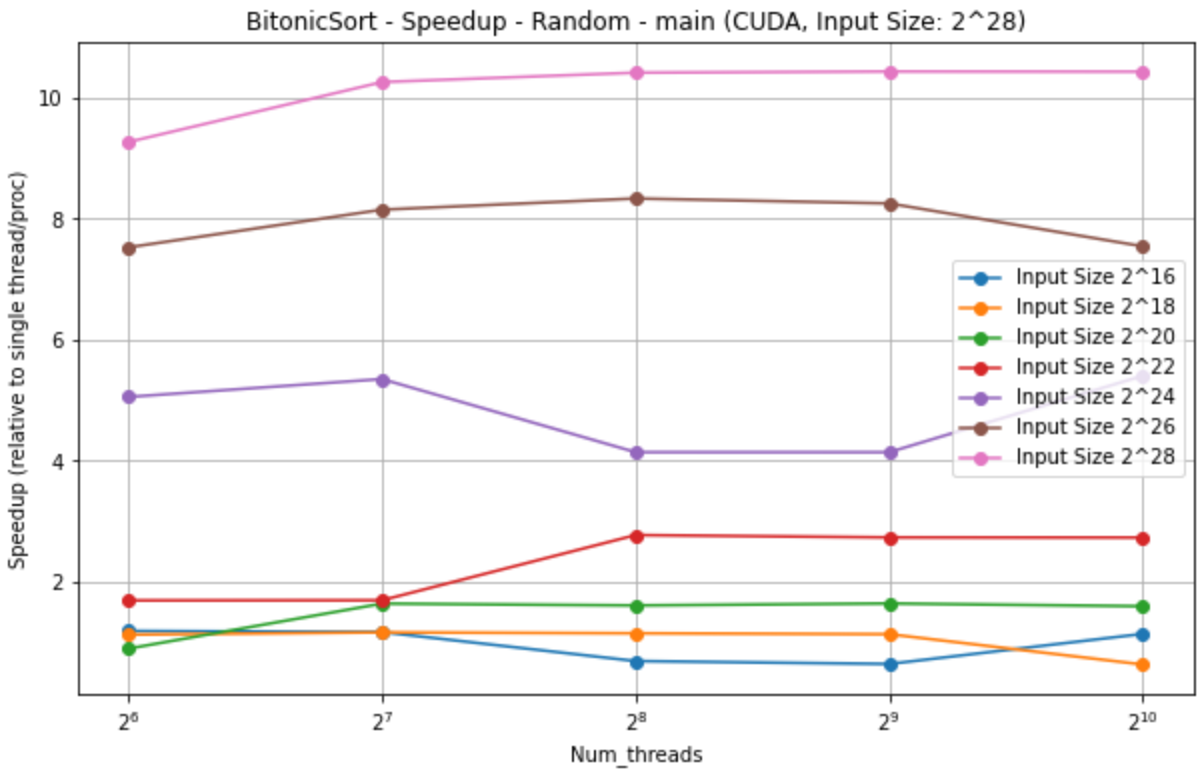

BitonicSort - Speedup - Sorted - comm (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

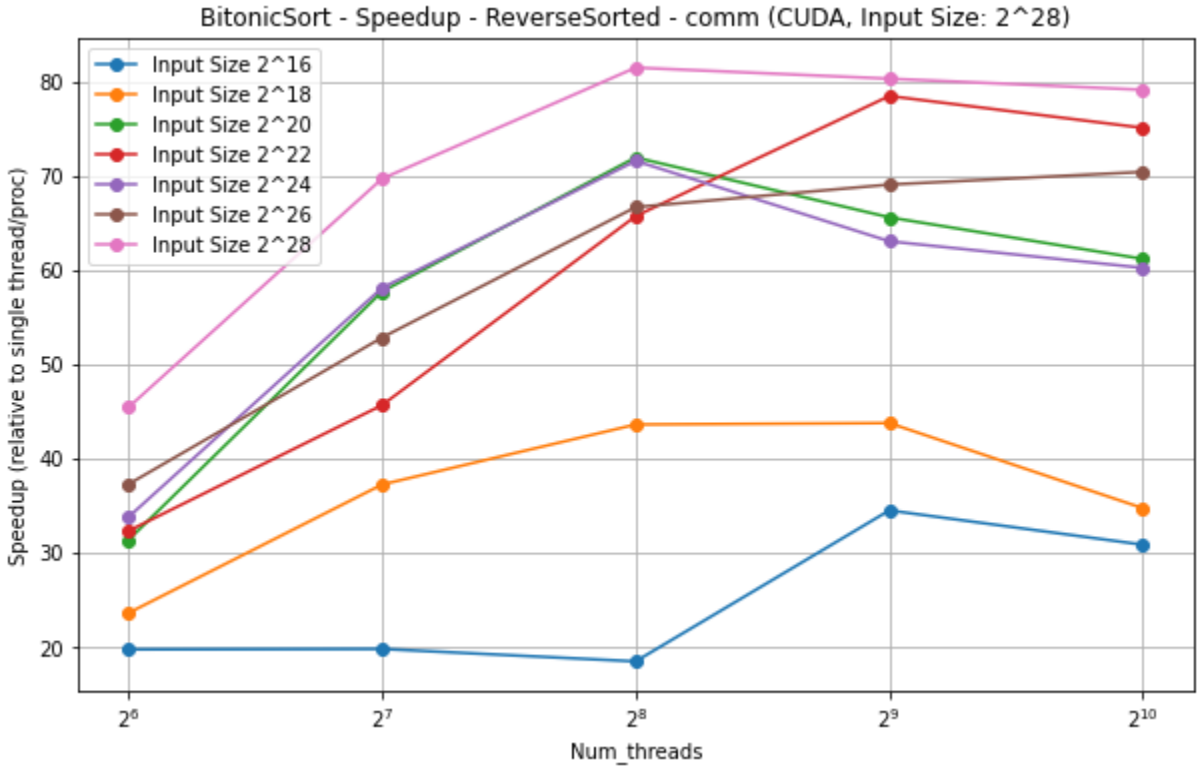

BitonicSort - Speedup - Sorted - comp_large (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
1.2
1.1
1.0
0.9
0.8
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
2^6
2^7
2^8
2^9
2^{10}
Num_threads

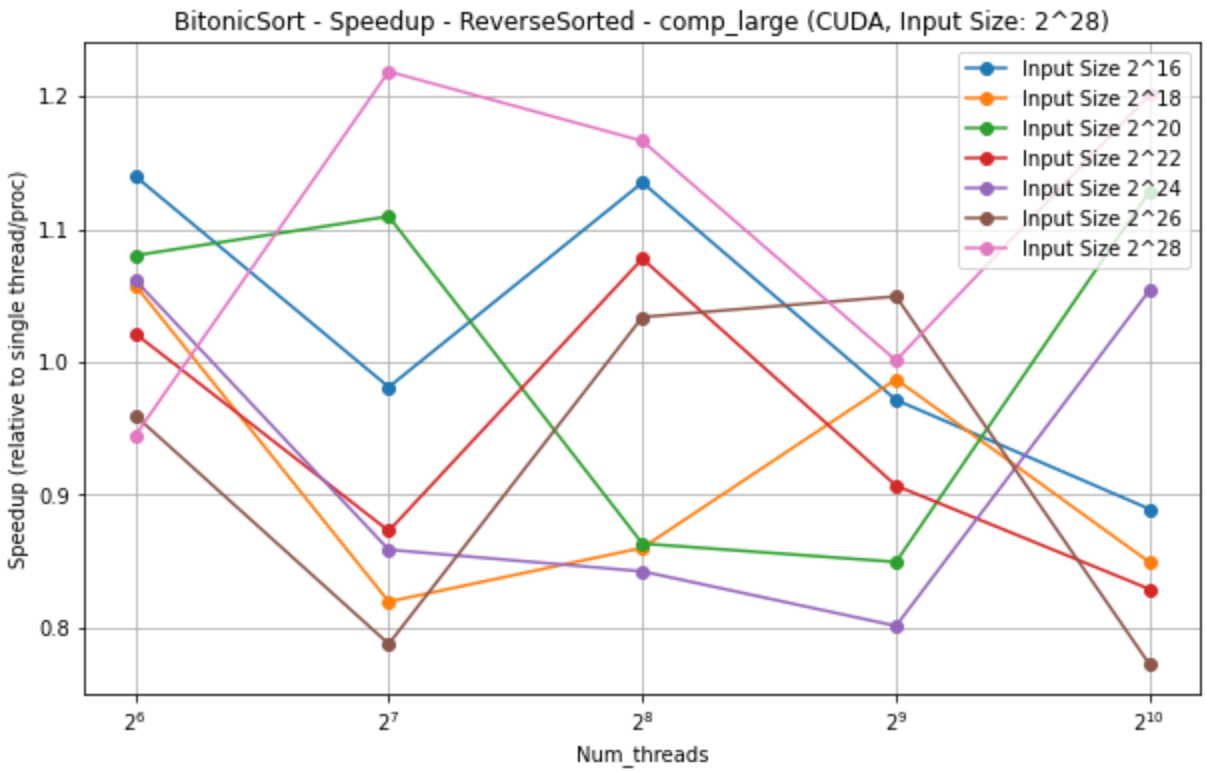

BitonicSort - Speedup - Sorted - main (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
20
15
10
5
0
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28
2^6
2^7
2^8
2^9
2^{10}
Num_threads

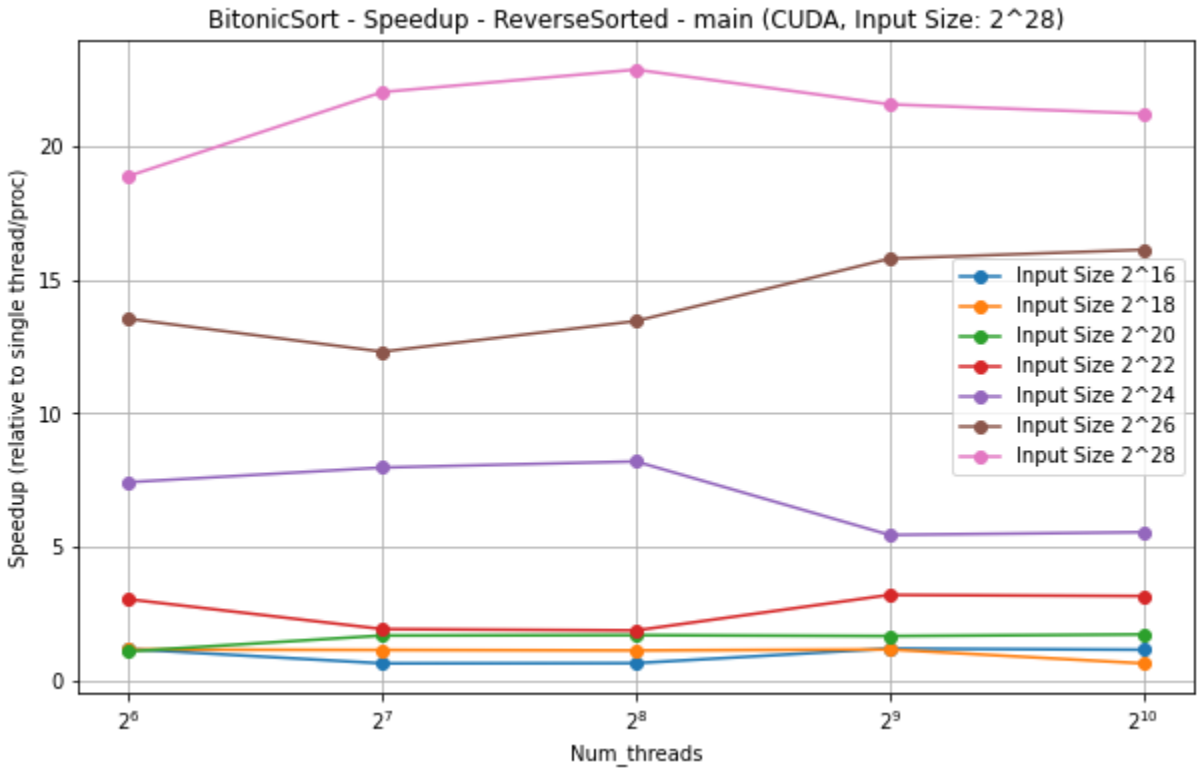

BitonicSort - Speedup - Random - comm (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28


BitonicSort - Speedup - Random - comp_large (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

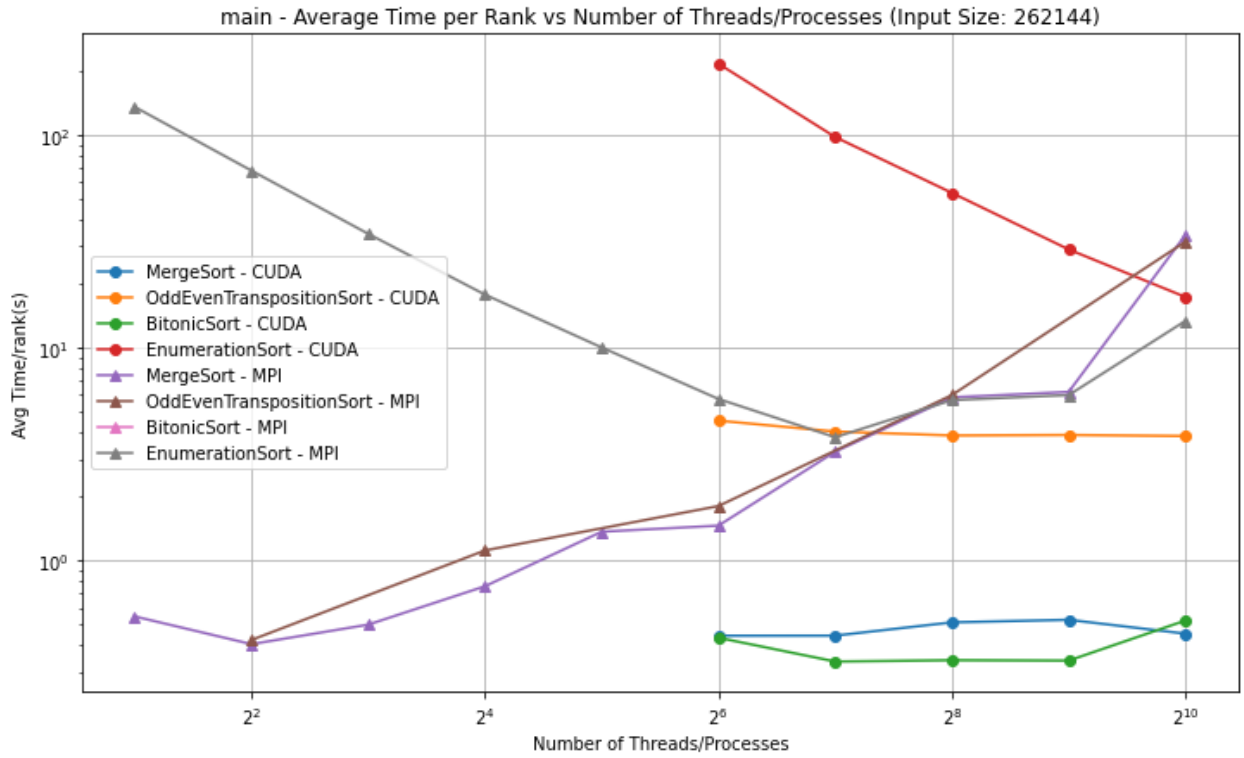

BitonicSort - Speedup - Random - main (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

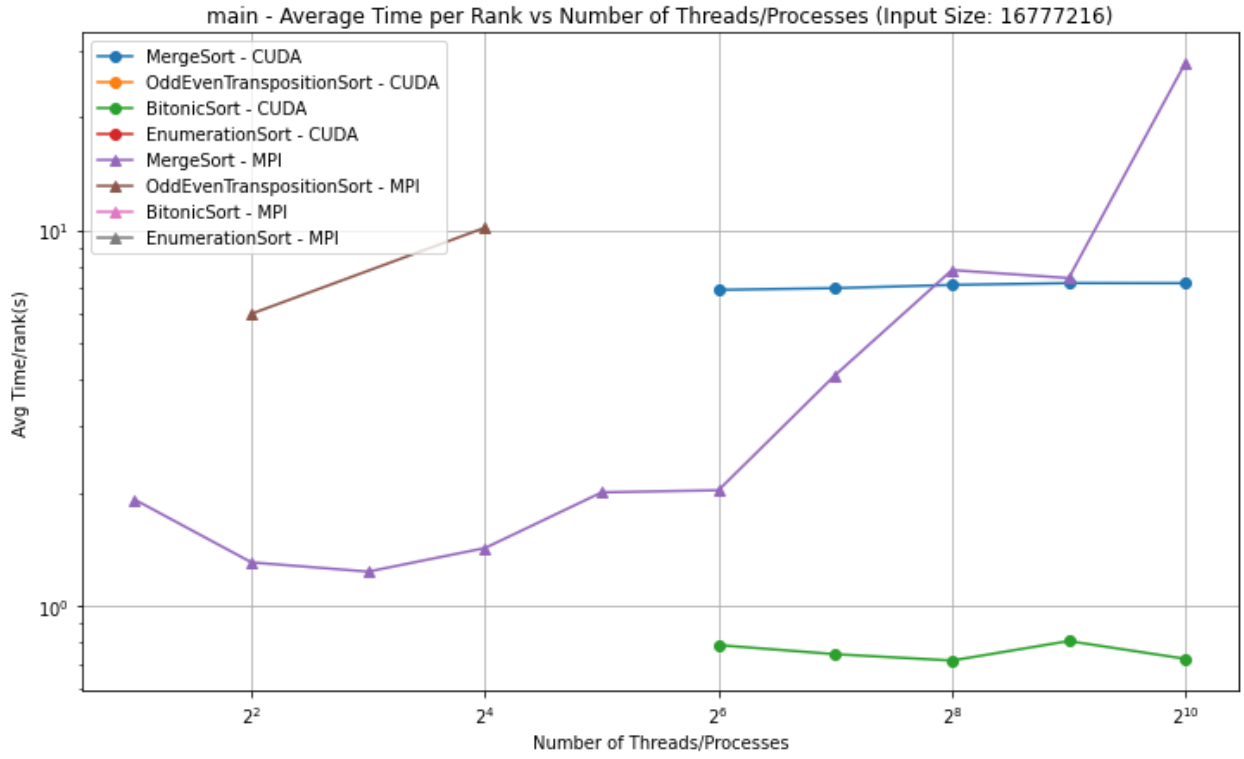

BitonicSort - Speedup - ReverseSorted - comm (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28


BitonicSort - Speedup - ReverseSorted - comp_large (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28


BitonicSort - Speedup - ReverseSorted - main (CUDA, Input Size: 2^28)
Speedup (relative to single thread/proc)
Num_threads
Input Size 2^16
Input Size 2^18
Input Size 2^20
Input Size 2^22
Input Size 2^24
Input Size 2^26
Input Size 2^28

Algorithm Comparison





For smaller input sizes (262144) CUDA implementations are generally faster than MPI implementations because communication overhead is not as big in CUDA algorithms. Some algorithms are able to scale well even on a small input size, while others aren't as visible on the plot. For example, based on previous plots we know that Merge MPI would scale better on inputs on a magnitude of 2^28, but on this specific input size.

For large input sizes(16777216) there are multiple algorithms that timed out for the cali file generation, such as some implementations of Enumeration and OddEven Sort as well as Bitonic MPI. Looking at the algorithms that are present in the plot, Bitonic CUDA is the fastest by far, and merge CUDA while doesn’t get slower as more threads, it is quite slower

The algorithm that benefits from a large number of threads and processes are Bitonic CUDA which has a low average sorting time with a number of threads and processes higher than 2 to the 6th. Other algorithms such as the Merge sort MPI and odd even transposition sort MPI have a fast sorting time however, as the number of threads and processes increases these algorithms will become more inefficient.