Merge Sort

- Project Documentation -

# Project description

This project aims to accelerate the sequential implementation of the well-known sorting

algorithm **MergeSort** by parallelization and multiprocessor execution of the code.

In order to speed up the execution time of the algorithm, we will be using C++ Threads/

For the testing part of this project, multiple sets of data will be generated, with different distribution of numbers and with an increasing sample size. The data will be run multiple times so that an average execution time can be observed and noted.

# Project environment 1

* CPU
  + Model: Intel i9-9900K
  + 8 Cores, 16 Threads
  + Frequency: 5.0 Ghz
  + Cache: 16 Mb
* RAM
  + Capacity: 32 Gb (4 x 8 Gb DIMMs)
  + Speed: 3200 Mhz
  + Channel: Dual Channel
* GPU: Nvidia GeForce 2070 Super 8 Gb
* Operating System: Windows 11 Pro

# Project environment 2

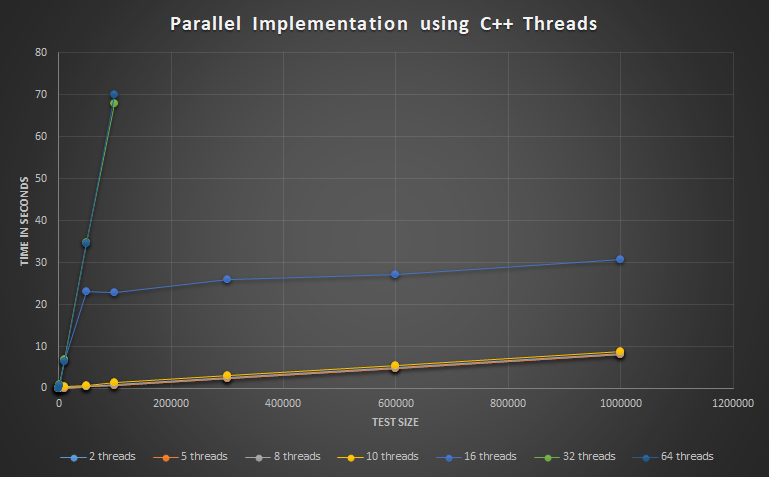
* CPU
  + Model: AMD Ryzen 5 5500U
  + 6 Cores, 12 Threads
  + Frequency: 2.1 Ghz
  + Cache: 8 Mb
* RAM
  + Capacity: 8 Gb (2 x 4 Gb DIMMs)
  + Speed: 3733 Mhz
  + Channel: Dual Channel
* GPU: None
* Operating System: Windows 11 Pro

# Parallel Implementation (**C++ Threads**)

In this implementation, there is a maximum number of threads that will be allocated at a given time. If there are enough available threads to be allocated, one will be delegated to merge a part of a given array from a parent thread, otherwise the sorting will be done locally, contained in the current running thread.

When the thread finishes sorting it’s current chunk of the array, it will exit and the thread pool size will go up.

The same tests were run on this implementation as the previous two. The following chart shows the C++ Threads performance:



As we can see, the results are not quite as expected. Starting with the obvious, the runs with 32 and 64 threads performed poorly, A test with 300.000 numbers on those two configurations took approximately 200 seconds, and the tests on 600.000 numbers did not even finish in a 10 minute time window, so those tests and bigger have been omitted from this chart in order to keep the data shown relevant.

By analyzing the relevant tests, we can actually see that the best time achieved overall on those tests is when running the program with only 2 threads. Adding additional threads seems just to slow down the algorithm gradually.

# Comparisons

Below we can see a comparison between the two systems on their execution time.

We can easily see that the Intel CPU performs better than the AMD one by far on bigger tests.

# References

[GitHub Repository](https://github.com/Ragnarok54/PDA)

[Sequential MergeSort](https://www.geeksforgeeks.org/merge-sort/)

[Parallel STL methods](https://software.intel.com/content/www/us/en/develop/articles/get-started-with-parallel-stl.html)