Merge Sort

- Project Documentation -

I. 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, the following synchronizing and parallel mechanisms will be used:

- MPI
- Pthreads
- Parallel STL

At least two of the mentioned mechanisms will be used for the parallel implementation of the MergeSort algorithm. The coding language used will be C++ so all the parallelization methods will be available

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.

In total, three implementations will be developed, a sequential one and two parallel. This is done in such a way that the results can have a better interpretation in relation to the methods used.

In order to quantify whether the parallel implementations have any actual benefit over the sequential one, a baseline needs to be established. The sequential implementation of the algorithm that will be developed is similar to the one linked in the references.

II. Project environment

All the implementations will be run on a PC with the following specifications:

- CPU
  - Model: Intel i9-9900K
  - 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 10 Pro

III. Sequential Implementation

The sequential implementation follows the basic logic for the **Merge Sort**, by dividing the input vector into two halves, and recursively calling itself on the two newly created halves.

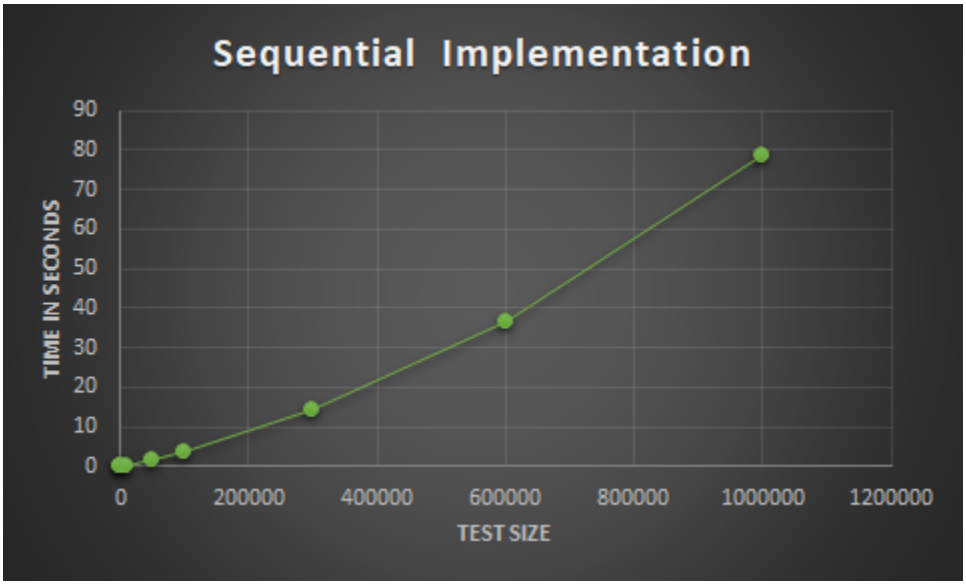
The sequential implementation can be seen at the **GitHub** repository linked in the references.
A quick explanation of the functions used can be found below:

- **merge(left, right)**
  This function does the merging between two sorted vectors and returns a single sorted vector. An improvement of note here, would be to use the merge function found in the “algorithm” header.
- **mergeSort(vector)**
  This function is the recursive part of the algorithm, that calls itself after splitting the initial vector in two halves.

For testing the sequential implementation, a smaller application was developed with the sole purpose of generating random sets of data. Those inputs are stored in the **“Input Data”** folder of the repository. In order to save space and time, 8 tests were generated, with varying sizes from 100 numbers to 1000000, with numbers between 1 and 1000000000 uniformly distributed.

The program was designed in such a way that it takes all the input defined and sorts them, then prints the time it took to sort the input to the console.

The execution time of the sequential algorithm was stored and compiled in the following graph.



As we can see, the execution time increases proportionally with the input size. This is expected, and for now, until we have the parallel implementations, we can not say anything more about this graph.

IV. References

GitHub Repository
Sequential MergeSort
MPI Tutorials
Pthreads Information
Parallel STL methods