PARALLEL COMPUTING - Challenge 1

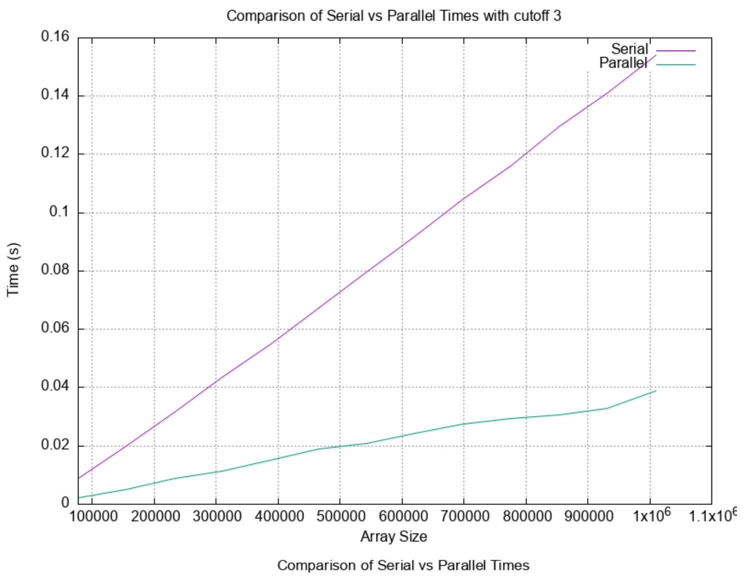
Benedetta Mussini 10794016 - Federico Quartieri 10806848 - Fabio Rossi 10764762

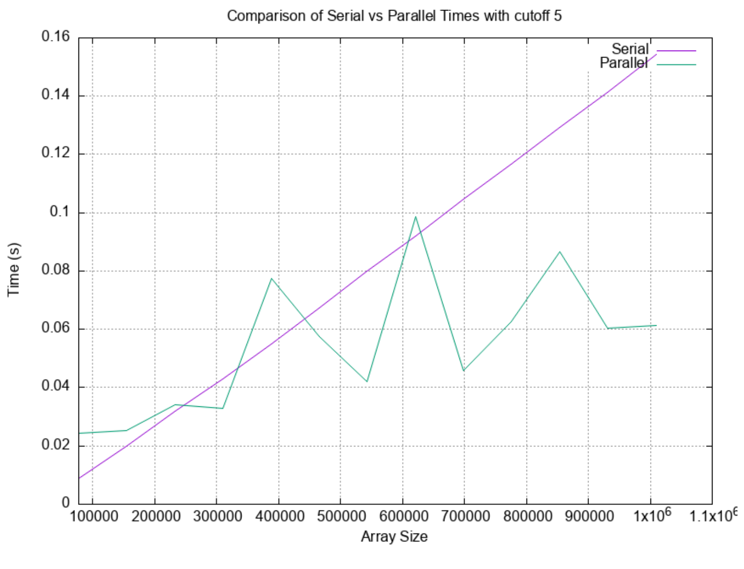
Experimental Setup:

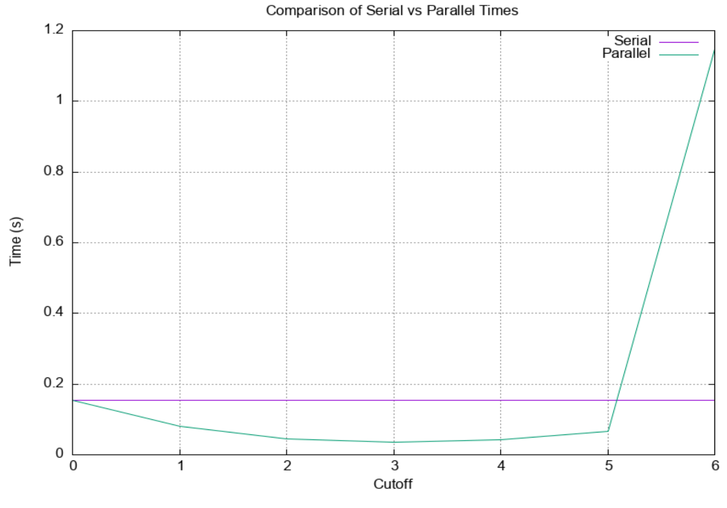
We used the following libraries: omp, which is required to use OpenMP and parallelize the algorithm, and fstream, which is needed for file writing and was used to create CSV files for plotting benchmarks based on array size and cutoff (expressed as the maximum depth of the parallel recursion tree).

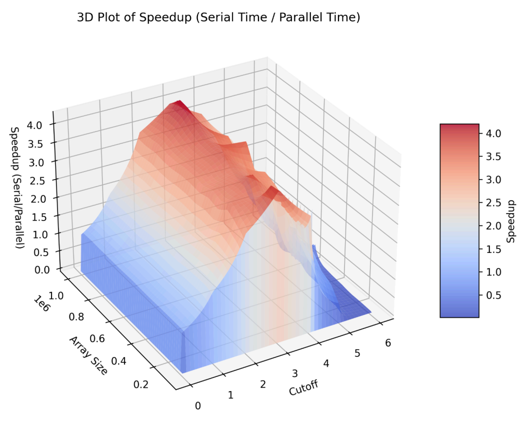
Performance Measurements:

As mentioned before, we compared the results by varying the array size and the number of threads used (and cutoff). Specifically, we decided to compare the results as the array size increases while keeping the cutoff value fixed, and we varied the cutoff value in different graphs. Then, we fixed the array size at the maximum value tested and allowed the cutoff to vary, observing how this affected the elapsed time. We also created a 3d graph with cutoff on the x-axis, array size on the y-axis and the speedup calculated as $\frac{Serial\ Time}{Parallel\ Time}$ on the z-axis, in order to make this graph smoother we performed an interpolation starting from the outputs of the simulations.









From the first graph we can state that with the optimal cutoff at the increasing of array dimension the speedup increases accordingly.

From the second graph we can state that if the cutoff is too big the architecture doesn't manage well the great number of threads and running in parallel starts to become inefficient. In the third graph we can see this properly as with a cutoff of 6 we lose competitivity against the serial algorithm.

In the last graph we can see how the speedup varies in variation to array size and cutoff, and how the optimal cutoff is evident and how the speedup decreases after that point.

Running the code with 1009000 as max size of array, 7 as max cutoff, 13 as number of different sizes and 4 as number of executions to compute the average for a given cutoff and size we obtain this result:

Max cutoff: 3 with speedup: 4.27273 and size: 3 MiB (931380 elements)

Explanation of Design Choices:

We chose to parallelize the division part of the mergesort algorithm. Initially, the master thread running the code starts the mergesort function by passing the entire array as a parameter. It then creates two additional single threads, each of which recursively calls the function with half of the array as input. This continues until the recursion depth reaches the cutoff, which is passed as a parameter to the program. Once this condition is met, the program switches to a sequential merge sort algorithm.

We set the maximum number of threads in the environment to be 2 raised to the power of the cutoff value

We decided to pass as arguments the following parameters:

- max size of array
- max cutoff
- number of sizes
- number of executions to compute the average for a given cutoff and size

We commented out the part of the code which was needed to create the graphs. Gnuplot and python need to be installed to run that part.