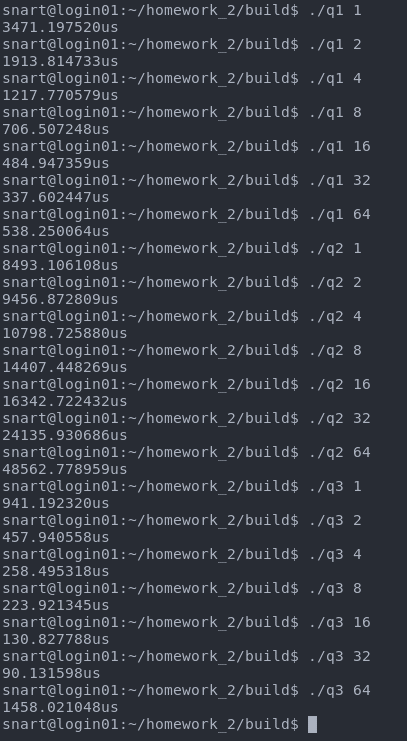
**Parallel Architectures – Homework 2**

Şamil NART - 110510129

Executing the executables in AGÜ HPC node:

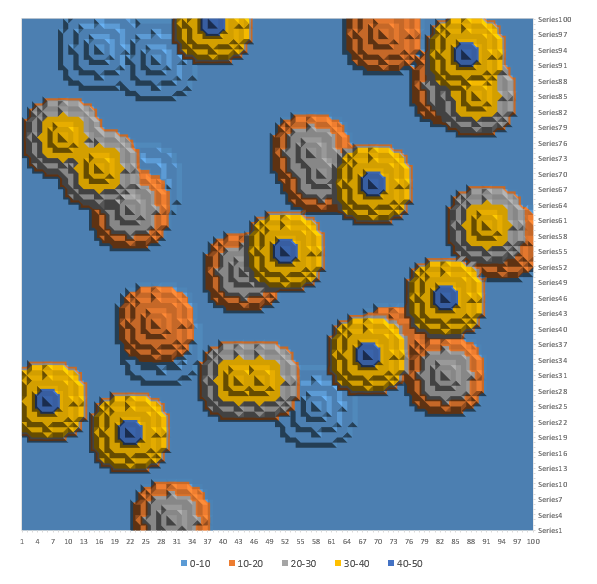


Results:

Q1) 48000.00

Q2) [sorted list]

Q3)



Methodology:

Each question’s algorithm was run 512 times and the average execution time was taken.

Q1)

Calculation of the integration was done according to the description in the assignment, the workload was spread among threads by calculating each rectangle in a separate thread and accumulating results in a `sum` variable.

Q2)

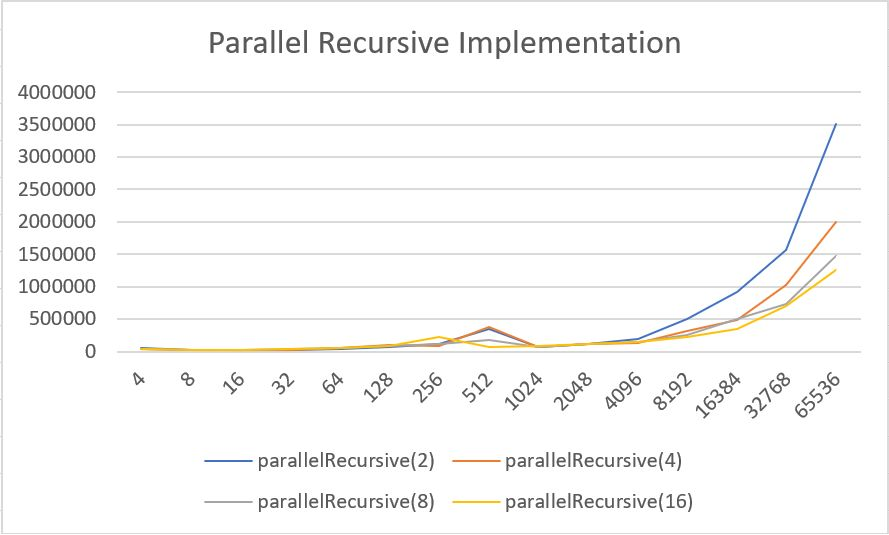
To sort the given array in a parallel manner, recursive merge sort was picked. Merging sub-arrays was done in a single thread, however work was divided across threads. I haven’t been able to get the expected results through OpenMP due to recursive nature of merge sort and its interference with proper task scheduling, however I also attached results I got from previous semester’s “Algorithm Analysis” course. This other result was gathered using Java’s `ForkJoinPool` thread pool and scheduling implementation.

Q3)

For this question the output map was divided into sub-regions and each sub-region was divided among threads. For each point in the map, closest heat-source was found and the heat on that point was calculated accordingly.

Execution Times

As it can be seen from the graphs, as thread count increases the execution time decreases, from this data we can conclude that using 8 CPU cores is the sweet spot. The sudden spike with 64 threads can be explained with the fact that the processing node used has 2x18 cpu cores, using more threads than cpu cores available will result in a lot of scheduling overhead, affecting overall performance.



(Previous work done on implementing parallel merge sort, x axis is for array size and y axis is execution time in nanoseconds, as it can be seen here, we get expected speed ups as thread count increases (from 2 to 16))