A2

Jonas B, Alexander M

October 2022

Part 1

Optimization

Our parallelized solution has a linear speedup with the amount of cores used. To prevent false sharing, each core gets its own histogram which will be added up at the end in a critical section. The true sharing is not a problem because each thread doesn't end at the exact same time and so the faster threads can add their results in a critical section while the other threads finish their work.

Results

Without the optimizations the function ran in about 0.65s, regardless of the number of cores given. With the optimizations the code runs in about 0.65s with 1 core, 0.33s with 2 cores, 0.19 with 4 cores and 0.09s with 8 cores.

Histogram size

The size of the histogram (amount of buckets) doesn't affect the time needed for the computation in both the optimized and unoptimized solutions.

Part 2

Optimization

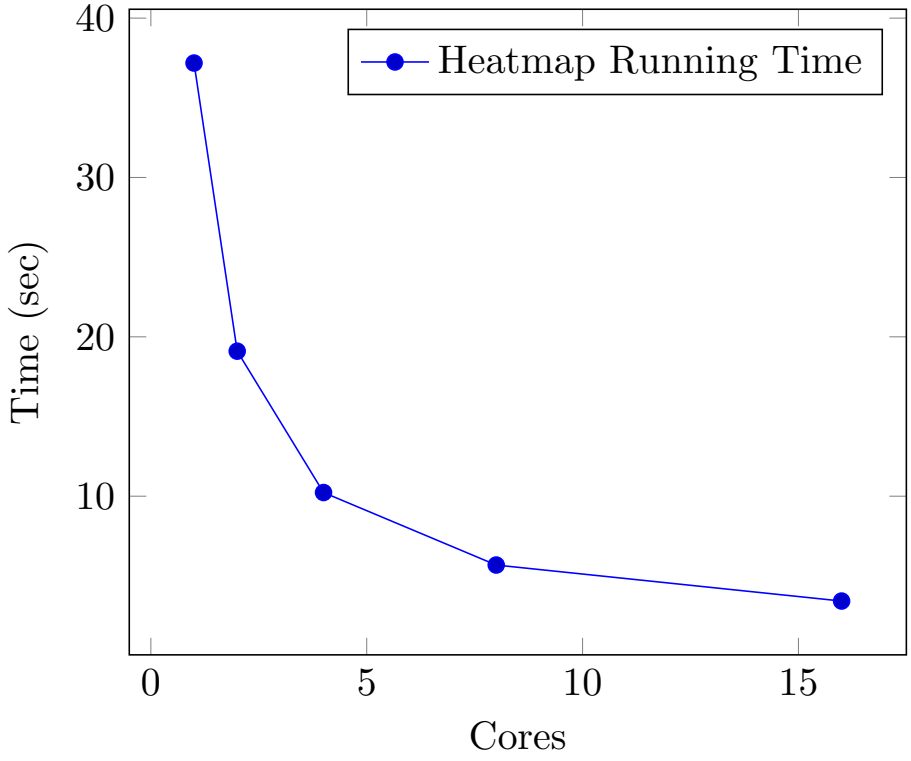
To make it parallel we used again used private arrays for each equivalent row in the 2d array called "output". This lowers the amount of false sharing as it is only possible to have false sharing in the merging loop.

The work could be split into rows. This has good data locality within the cores as the element (I, J) is close to the element (I+1, J) in the array output. The work could be split into columns. This has bad data locality within the cores as the element (I, J) is far away (I, J+1). In fact there is a whole row of data between the two elements.

If the number of cores is bigger than the number of rows in the array, some cores will not get any work to do because the work is not split into small enough parts.

Results

Our program has a linear speed up with respect to the number of cores used. We have only one optimization: reduce false sharing by using private arrays.


Heatmap Running Time
Time (sec)
40
30
20
10
0
5
10
15
Cores