Report

Parallel Systems: Sequential Algorithms with OMP

Auer Thomas

January 7, 2023

Contents

1. Introduction

The following report shows performance analytics of several popular algorithms including matrix multiplication and the following sorting algorithms: bubble sort, bucket sort, counting sort, insertion sort, brick sort, and selection sort. The main goal was to review the time complexity of the algorithms when changing the dimensionality of the input data and the amount of threads computing the programs.

2. Methods and Resources

The algorithms were performed on a AMD EPYC 7702P 64-Core Processor via the programming language C and the library OpenMP. The program implementations are available on Github.com.

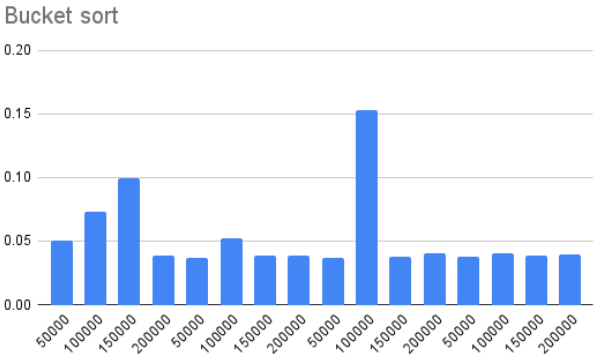
The methodical approach was to run each program execution via different input parameters, ranging from different element dimension to varying amounts of threads utilized.

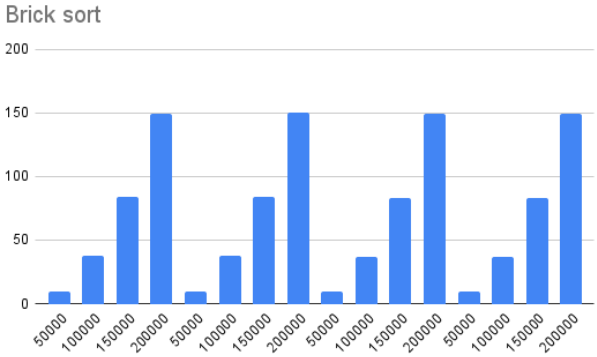
Algorithm	Elements	Threads
Matrix multiplication	500 1000 1500 2000	1 2 4 8
Brick sort	50000 100000 150000 200000	1 2 4 8
Bucket sort	50000 100000 150000 200000	1 2 4 8
Counting sort	50000 100000 150000 200000	1 2 4 8
Insertion sort	50000 100000 150000 200000	1 2 4 8
Selection sort	50000 100000 150000 200000	1 2 4 8

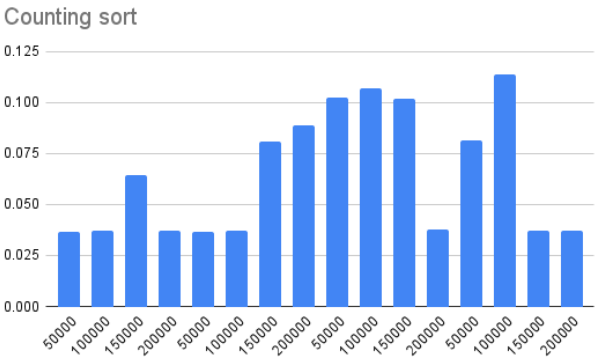
Table 2.1: The parametrization of each algorithm. Each thread would run a task with each of the stated amount of elements.

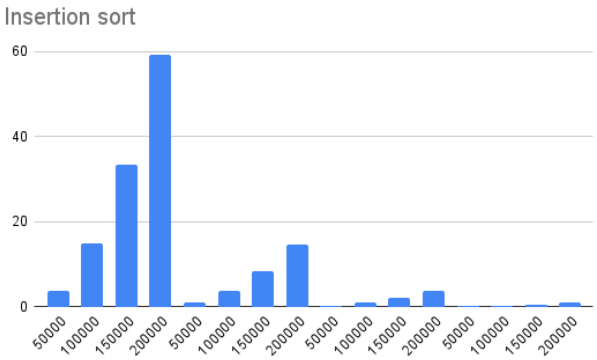
3. Results

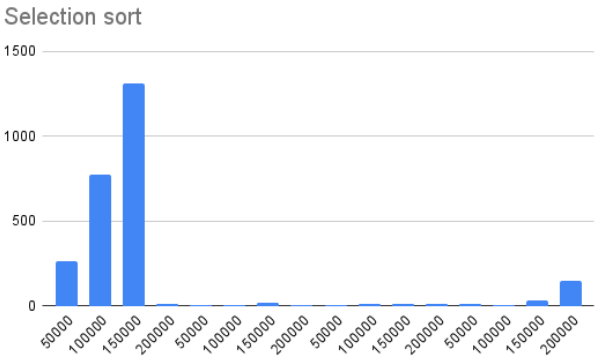
Each diagram provides results for each thread parametrization running the amount of stated elements. The diagrams order of 4 sections each defined by the number of threads utilized (1,2,4,8).

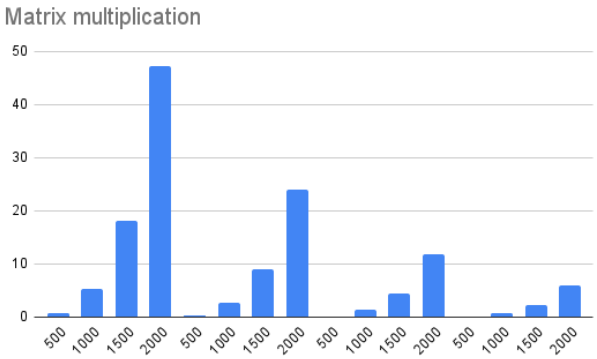












4. Discussion

Most of the algorithms have returned satisfiable results, while some data points are reasons of concern.

Selection sort performed horrendously on just 1 thread. Bucket sort did not significantly improve in execution time for $(n > 2)$ threads. Brick sort did not improve at all. Counting sort worsened with multi-threading.

5. Conclusion

The use of OpenMP has severely boosted time performances, for little effort. In comparison, POSIX threads would be significantly more complex to implement, but would have allowed deeper insights into the program's code and execution.