Homework 2 - ID1217

Youssef Taoudi, yousseft@kth.se

February 12, 2019

Abstract

Performance report for implementation of Matrix and Palindromes using OpenMP.

1 Matrix

I made two matrix programs, one that used the openMP critical section functionality and one that used reduction compare with customized min and max functions.

Figure 1: Speedup times for matrix using reduction with size 10

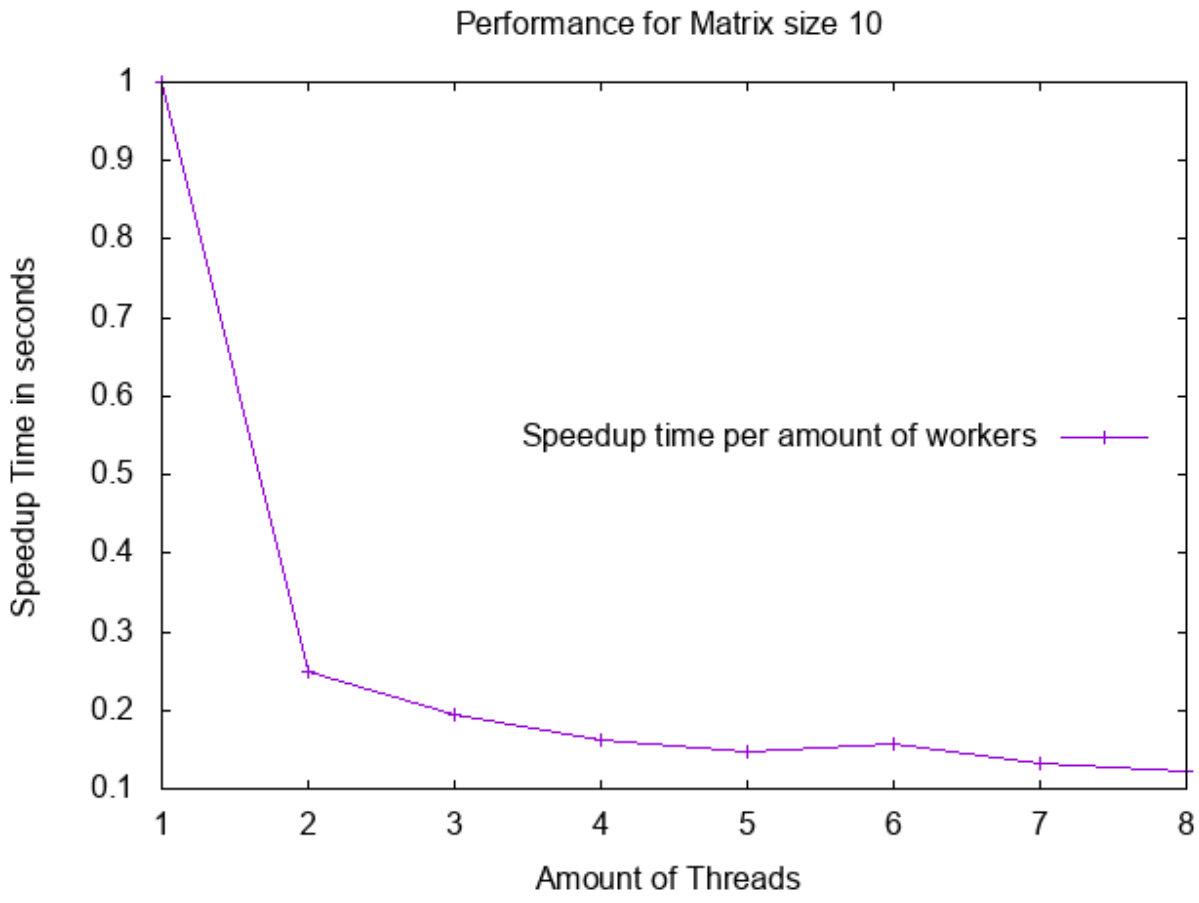

Figure 2: Speedup times for matrix using reduction with size 500

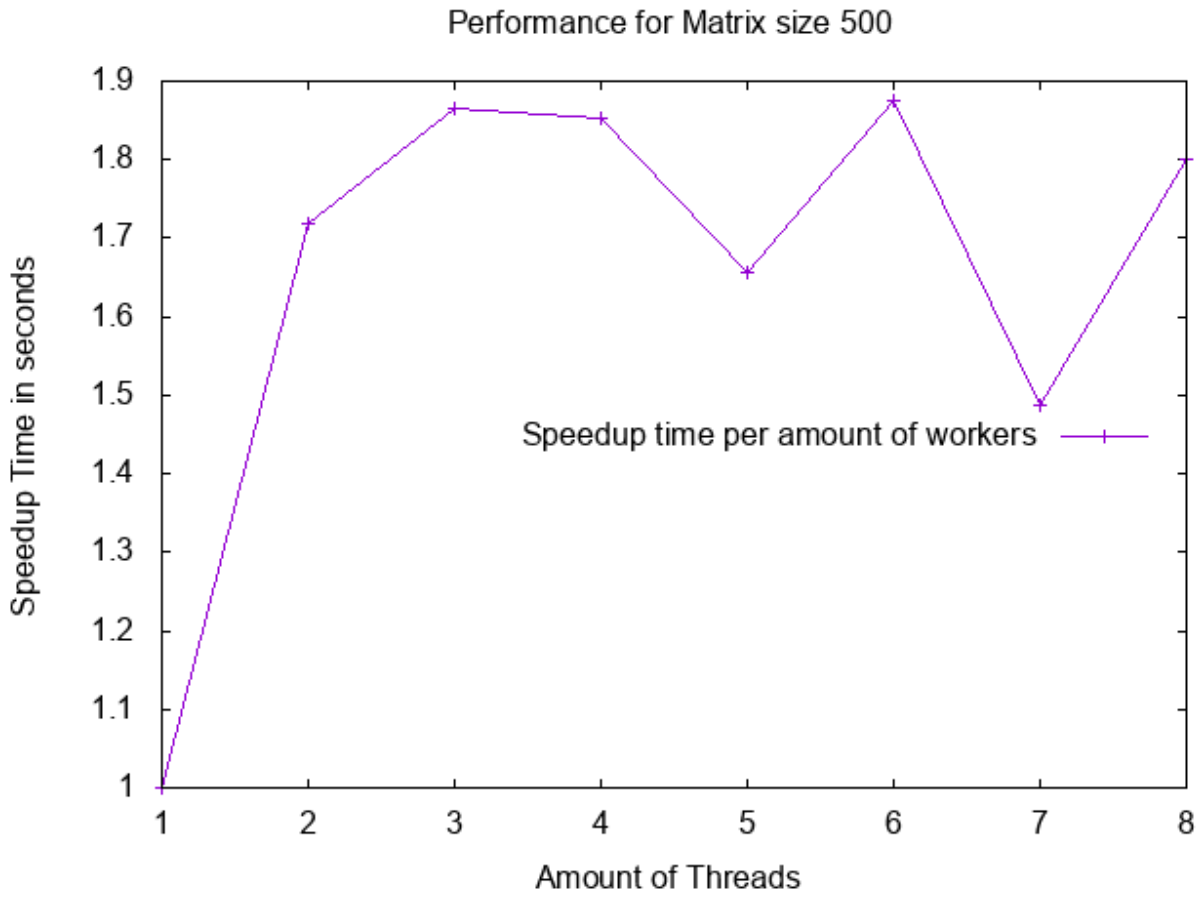

Figure 3: Speedup times for matrix using reduction with size 10000

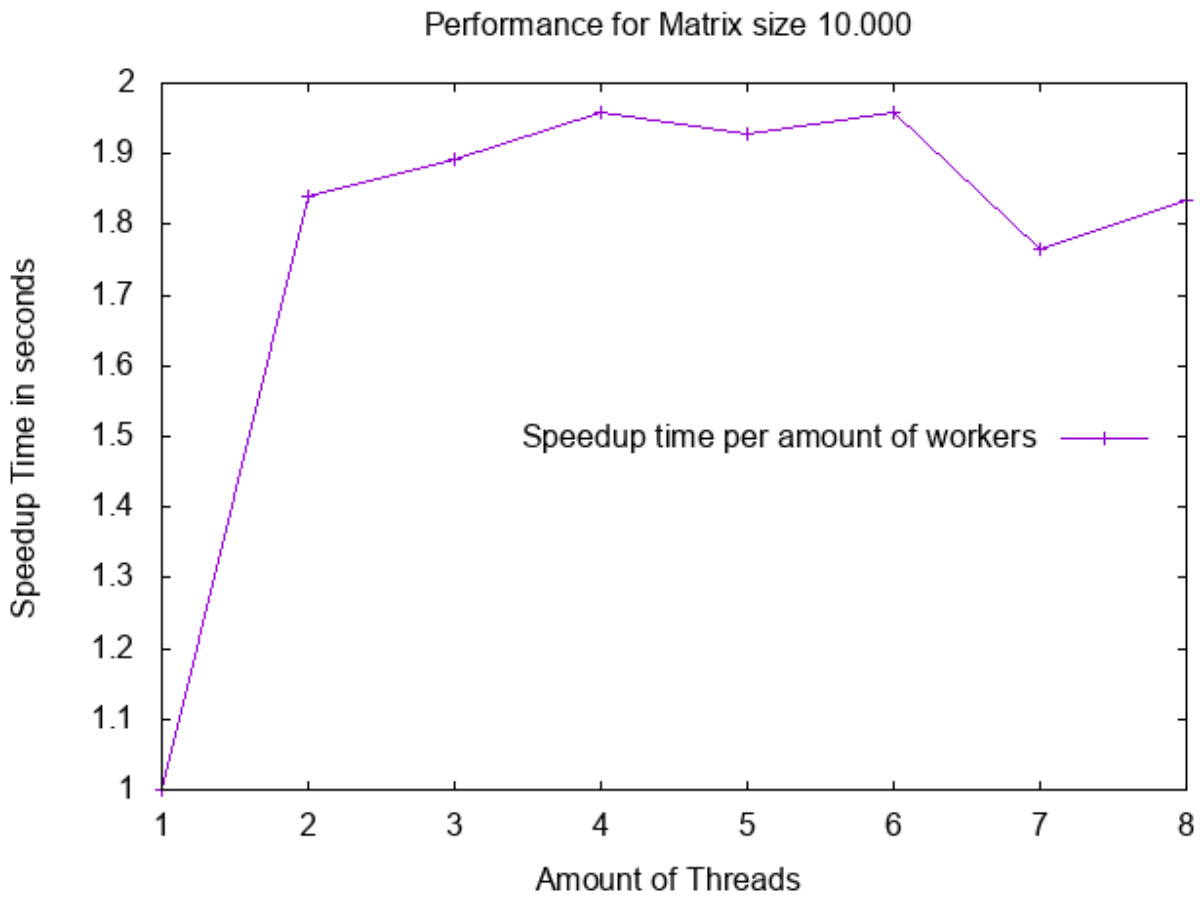

Figure 1,2 and 3 showcase the speedup when running the matrix using OpenMP reduction on different sizes.

The three figures above show the speedup for different amount of threads on three sizes. The first figure shows the speedup when running in parallel on a matrix of size 10. In this case, there was a slowdown in performance most likely due to the fact that the overhead of the context switches are greater than the actual speed of running the program sequentially.

In the other two figures (2 and 3), the size is much bigger and the performance shoots up for higher amount of threads. The peak for speedup is when running on 4 threads, my guess is because the computer that ran the program ran it on 4 cores.

Figure 4: Speedup times for matrix using locks

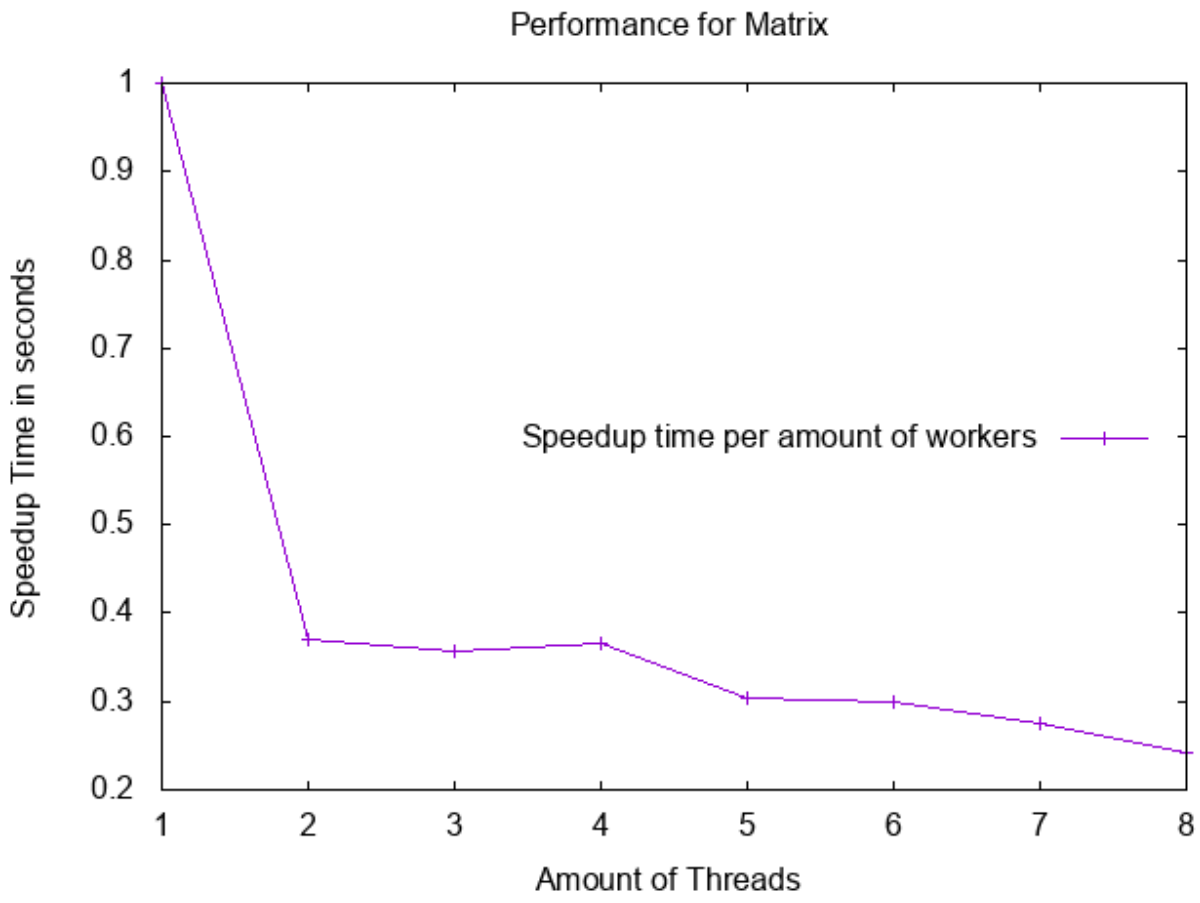

Figure 4 shows the performance of running the matrix with critical sections for size 1000. As you can see, the overhead of the locks that are used internally in the critical section functionality slow down the program for parallel execution.

2 Palindromes

Similarly to Matrix, I executed the program for three different sizes of dictionaries for 10 different number of threads.

Figure 5: Speedup times for dictionary of size 100

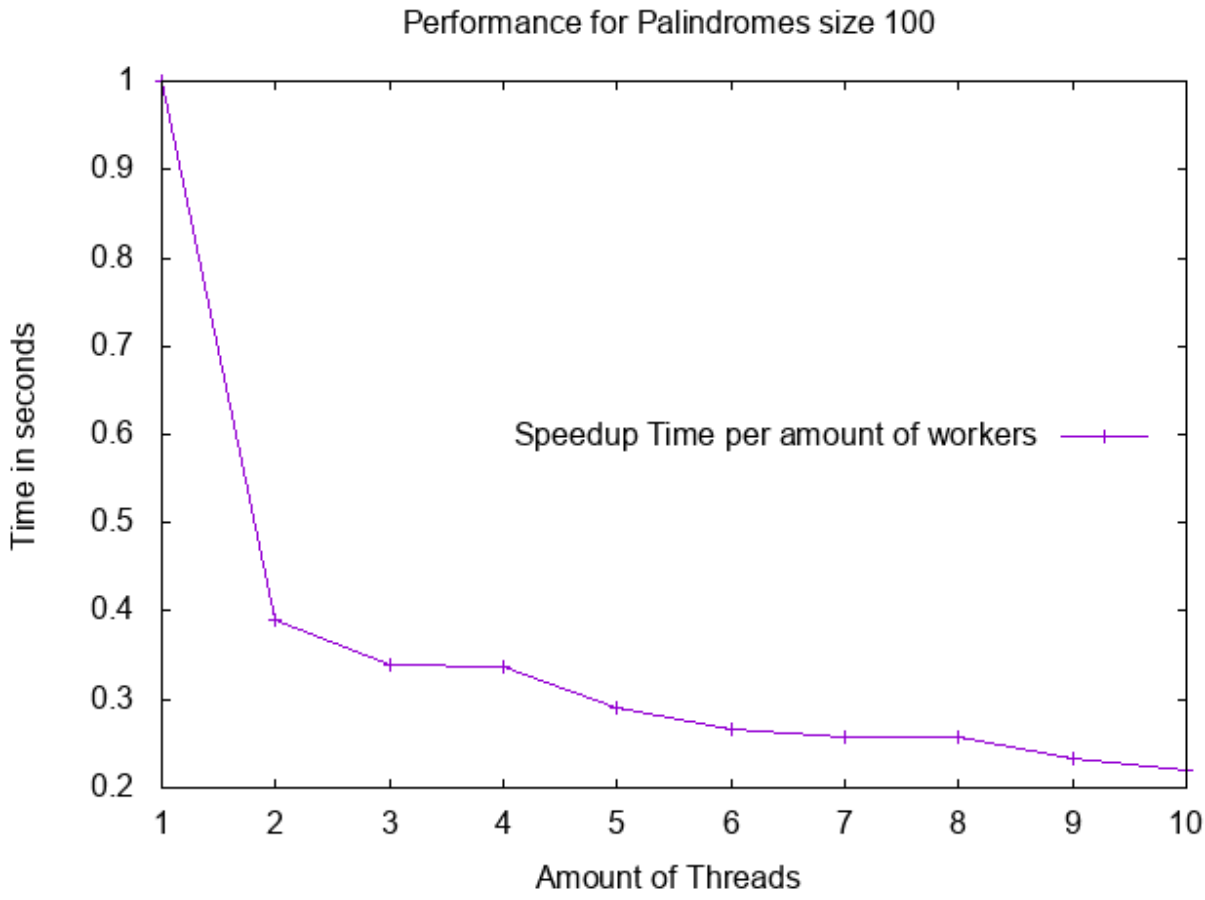

Figure 6: Speedup times for dictionary of size 5000

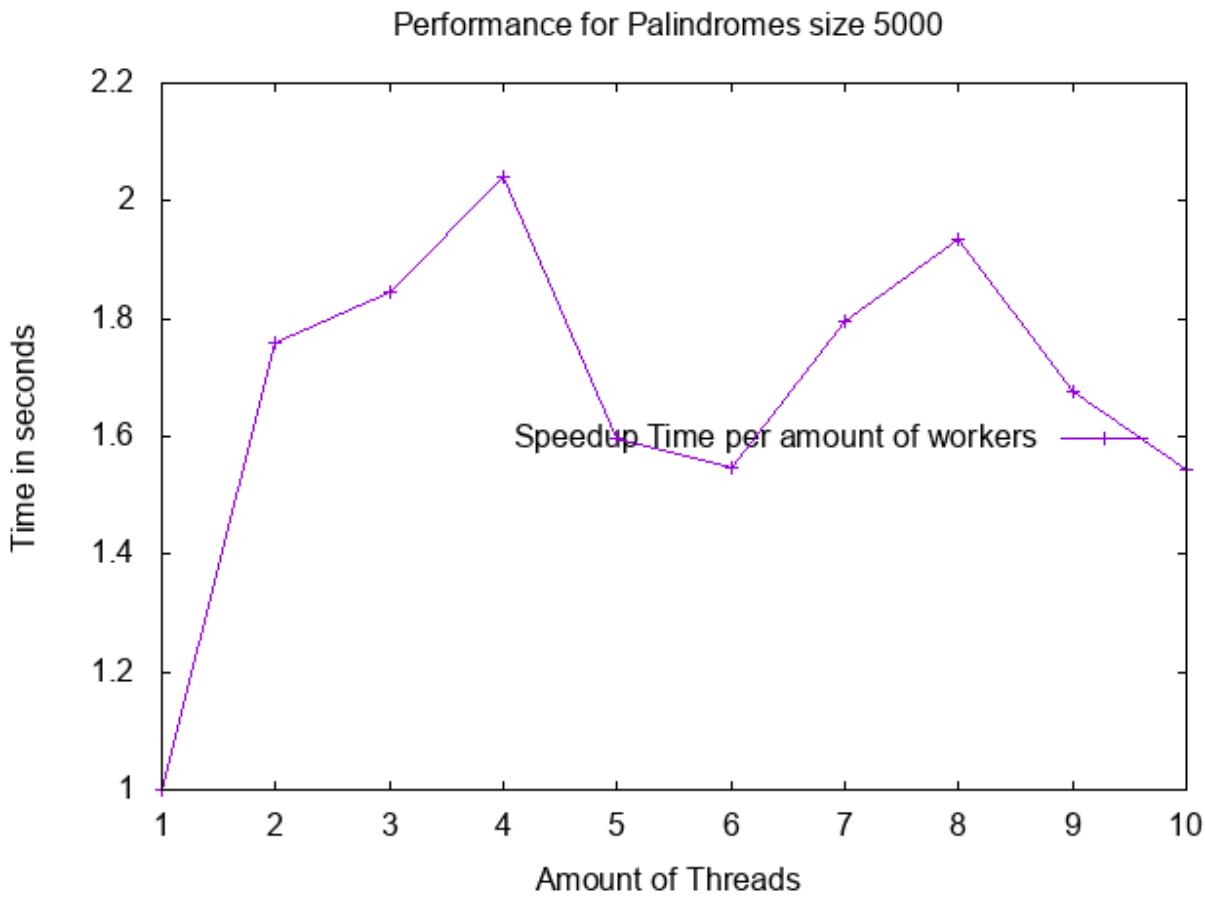

Figure 7: Speedup times for dictionary of size 21453

In this case, the speedup gets better and better for greater sizes. For a dictionary of size 100, running the program in parallel is inefficient meaning there is a certain threshold where running the program sequentially is better than in parallel (on my machine). Also worth noting is that palindromes do not use locks or reduction, there are no shared variables or race conditions.