Assignment 3

Introduction to Parallel Programming HT 2024

Lukas Bygdell Malmstig
lukasbygmal@gmail.com

Oskar Perers
oskar.perers.6706@student.uu.se

I. Exercise 1

For the first exercise we took our (base) implementation of the Sieve of Eratosthenes algorithm for finding prime numbers from the last assignment and reformatted the code to use Open MP instead of pthreads. The new implementation, in short, just replaced the creation of pthreads with a ”pragma omp parallel”. We use ”pragma omp single” to get the actual number of threads we are using and then dived the work using the thread ID the same way we did before. For this algorithm this is a quite nontraditional way to divide up this work as a ”pragma omp for” could replace it which would be easier to implement. We decided to stick with the original solution instead to make a better comparison between pthreads and Open MP. If using Open MP to implement the algorithm from scratch we would use that method instead as that is much easier and lets us avoid doing this work-division manually. Still just using ”pragma omp parallel” from Open MP feels easier than using pthreads.

The speedup graphs calculated using Equation (1) can be seen in Figure 1 and Figure 2 for the new solution and the original solution respectively. The differences between these two solutions seem to be minimal at most. This is logical as Open MP often uses pthreads under the hood and is just an abstraction layer. The compiled versions of these solutions are most likely very similar and therefore the performance is about the same.

$$Speedup = \frac{T_{old}}{T_{new}} \tag{1}$$

II. Exercise 2

Exercise 2 is based on Conway’s game of life. We received a sequential solution to the problem and then remade it into a parallel version using Open MP. The solution is using ”pragma omp for” in combination with the collapse argument to parallelize the loops in the sequential implementation. The speedup for three different board sizes calculated using Equation (1) can be seen in 3 and 4 for 1000 and 2000 time steps respectively. What we can see from them is that the program can’t utilize much more than 4 threads efficiently for a board size of 64x64. For the larger board sizes the efficiency of adding more threads is much higher. We also see that there is no notable difference in speedup going from 1000 to 2000 time steps. This is most likely because there is an implicit barrier at the end of each time step resulting in the average time per time step being about the same.

Threads/Matrix size	64	256	1024
1	0,009602	0,6715	49,91
2	0,004976	0,3357	24,81
4	0,002592	0,1680	12,44
8	0,001606	0,08493	6,284
16	0,001322	0,04531	3,451
32	0,003342	0,02697	2,151

Table I
Table for the runtime[s] for matrix multiplications for different sizes and different numbers of threads where only the outermost loop is parallelized.

III. Exercise 3

For exercise 3 we were directed to look at a code example from lecture 7 for matrix multiplication. The assignment was to implement this code, parallelize it and run it with different levels of parallelization and with a different number of threads. We expanded this test to run it with different sizes of matrices. The result can be seen in Table I where only the outermost loop is parallelized, Table II where the outer two loops are parallelized and Table III where all three loops are parallelized.

What we can see is that the efficiency of going from 8 to 16 threads for size 64x64 drops substantially while only one loop is parallelized while being much better in the other two cases. Going from 16 to 32 gets worse performance when one loop and two loops are parallelized while having some improvements when all three loops are parallelized. For all other tests the efficiency of adding more threads is good and there is no substantial difference between the number of loops parallelized that can’t be described by margin of error.

The inconsistencies we see at size 64x64 for 16 and 32 threads is most likely due to the work being easier to split among the higher number of threads as the work is split into smaller pieces. While for all other cases all threads have enough to do without splitting up the work more.

IV. Exercise 4

For both the column and the row-oriented Gaussian elimination we can only parallelize the inner for-loops. This is because the inner loop depends on all previous iterations of the outer loop. But each inner loop has no such dependencies. For the row oriented method we will encounter issues tho as each loop tries to update the same x[row] and we will therefore encounter errors or have to have each thread wait for the previous one to do their update of x[row]. This can be avoided by declaring

Threads/Matrix size	64	256	1024
1	0,009493	0,6702	49,74
2	0,004761	0,3350	24,60
4	0,002396	0,1674	12,26
8	0,001193	0,08398	6,164
16	0,0004157	0,04488	3,215
32	0,0006495	0,02606	2,186

Table II
TABLE FOR THE RUNTIME[S] FOR MATRIX MULTIPLICATIONS FOR DIFFERENT SIZES AND DIFFERENT NUMBERS OF THREADS WHERE THE OUTER TWO LOOPS ARE PARALLELIZED.

Threads/Matrix size	64	256	1024
1	0,009705	0,6889	51,54
2	0,004856	0,3448	25,73
4	0,002439	0,1723	12,85
8	0,001211	0,08621	6,465
16	0,0006263	0,04564	3,369
32	0,0004638	0,02655	2,170

Table III
TABLE FOR THE RUNTIME[S] FOR MATRIX MULTIPLICATIONS FOR DIFFERENT SIZES AND DIFFERENT NUMBERS OF THREADS WHERE ALL THREE LOOPS ARE PARALLELIZED.

the for-loop as a reduction to sum x[row] in each thread and then join the values together.

After getting it working locally there were errors running it on Tusilago and therefore the runtime experimentation wasn't possible. There is most likely some problems with false sharing on the older architecture and there wasn't enough time to get it working correctly.

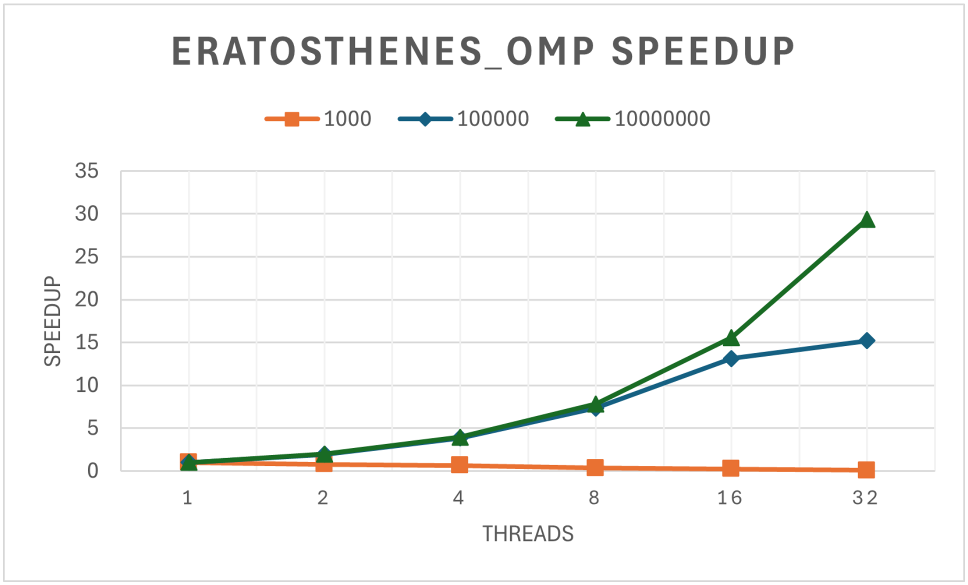

Figure 1. Speedup graph for the Open MP implementation of Eratosthenes with different max values.

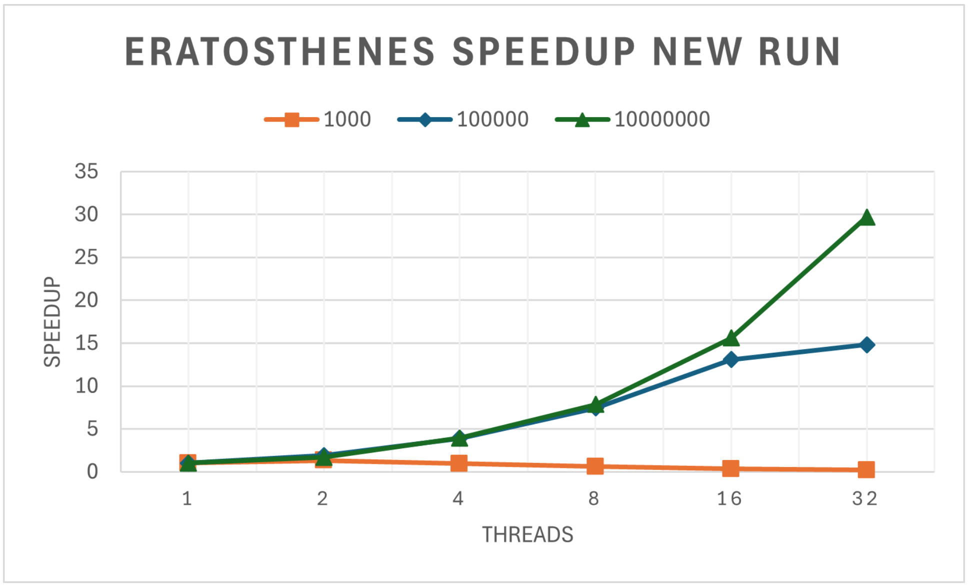

Figure 2. Speedup graph for the original implementation of Eratosthenes with different max values.

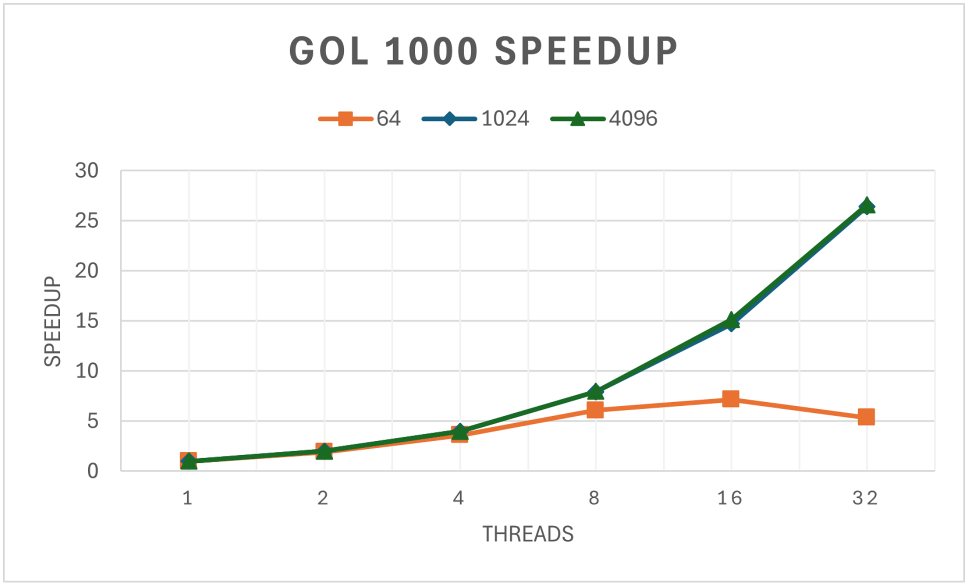

Figure 3. Speedup graph for Game of Life with different board sizes and 1000 time steps

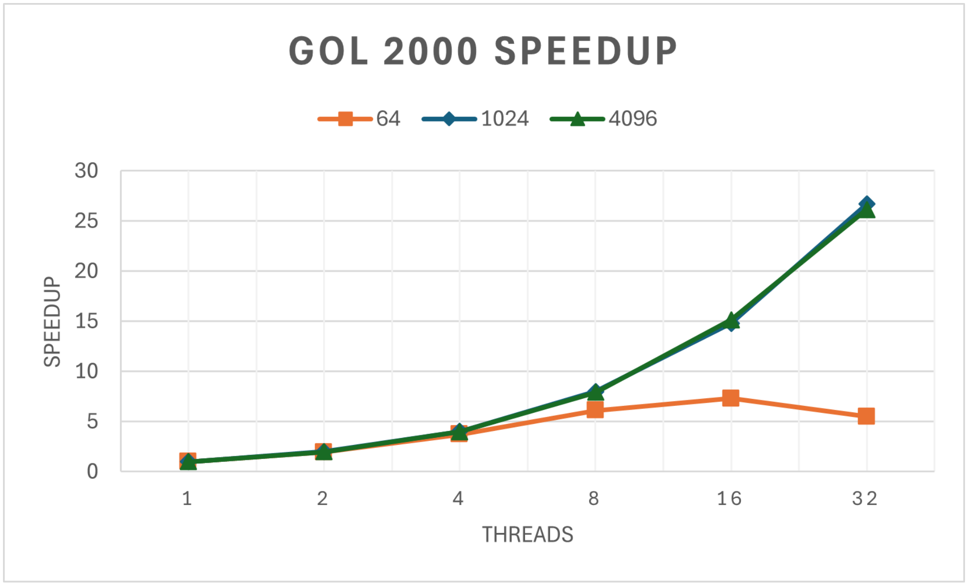

Figure 4. Speedup graph for Game of Life with different board sizes and 2000 time steps