Introduction

For this project we got the option of picking between four different parallel programming assignments. The one that we picked was about parallelizing a particle simulator. The original particle simulator had the time complexity of $O(n^2)$ where n is the amount of particles and our task was to first optimize the code to achieve a time complexity of O(n) and then parallelize it in order to achieve time complexity close to O(n)/P where P is the amount of threads.

Method

Serial Optimization

In order to first achieve the time complexity of O(n) the original code obviously needed to be optimized. This was because the original code worked in the way that every single particle compared itself to every other particle and if those particles were inside the cutoff range they would also calculate and apply relevant forces to those particles. The biggest problems with this is since obviously not every single particle will be relevant to every other particle this adds a lot of unnecessary work. Our solution to this problem was to divide the simulation into a grid of squares and mapping all the particles to different squares. Also by setting the sides of the cells to never be smaller than the cutoff range we could also ensure that all the relevant particles to a given particle were in it's moore neighbourhood. What this meant was that for each cell we only had to check eight other cells, which is constant work.

Shared Memory Implementation

Going from the serialized solution to a parallel solution when using a shared memory design was not terribly difficult. Basically what had to be done was to identify the critical sections of the code and make sure that it will run with mutual exclusion. We then give each thread a set of particles to each keep track of. The critical section is the parts of the code that write to the grid structure.

Message Passing Implementation

Compared to the shared memory implementation the message passing implementation was a lot tougher to implement. For this the design had to ensure that we did as little memory sharing as possible. What we did here was instead of assigning a portion of the particle array to every thread we instead assign a set of rows of the grid matrix. By doing it this way each thread was responsible of a set of particles that were geometrically close. For this to work we had to implement a way for passing particles between threads and also passing the border rows since we still need the moore-neighbourhood for each particle.

Result

To make sure that we reached the desired time complexity of O(n) the most convenient way was to plot the execution time based on how many particles. In order to do this we ran the program five times for each implementation and took the median value. These plots can be seen in the appendix, all of the computations were done on the KTH shell server for this course. Figure 1 shows the plot for the optimized serial version of the program inside a log-log plot. As you can see the graph is indeed linear. As for figure 2 it shows an execution of the Pthread implementation on 1 thread and it's also pretty much linear. Figure 3 shows an execution of the OpenMP implementation on a log-log plot which also shows close to linear execution time. Figure 4 shows the execution time for the MPI implementation of the program. These figures all show close to linear execution time (O(n)). The next problem was to achieve O(n)/P where P is the amount of threads and n is the amount of particles. In order to show how parallelized the programs were, we plotted the different speed-ups based on the number of threads. To calculate the speedup you take the serial execution time and divide it by the parallelized execution time. For the Pthread and OpenMP implementations the speedup looks to be very close to the requested but not quite. (see figure 5 and 6)
As for the MPI implementation doesn't seem to be quite as efficient as the shared memory implementations. This is probably because while threads in shared memory implementations always share the same information when it comes to message passing implementations the threads have to send the information that they share. This of course take extra time and we believe this is the reason for the slightly lower performance.

Appendix

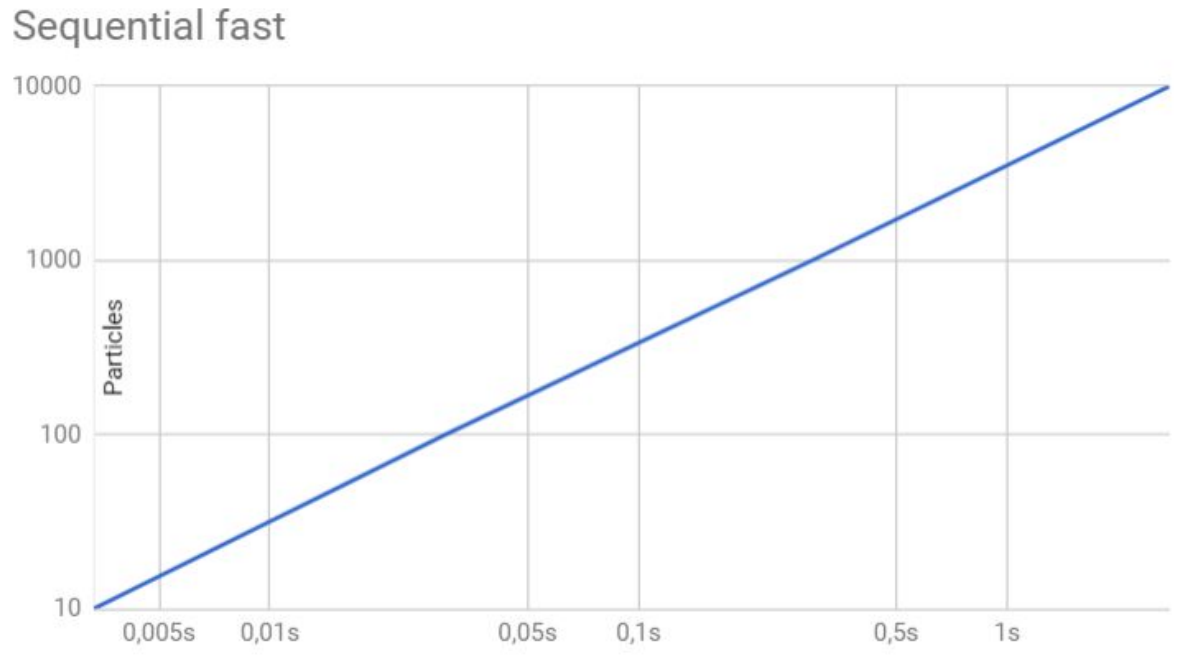

Figure 1, the serial program's execution time with respect to the amount of particles.

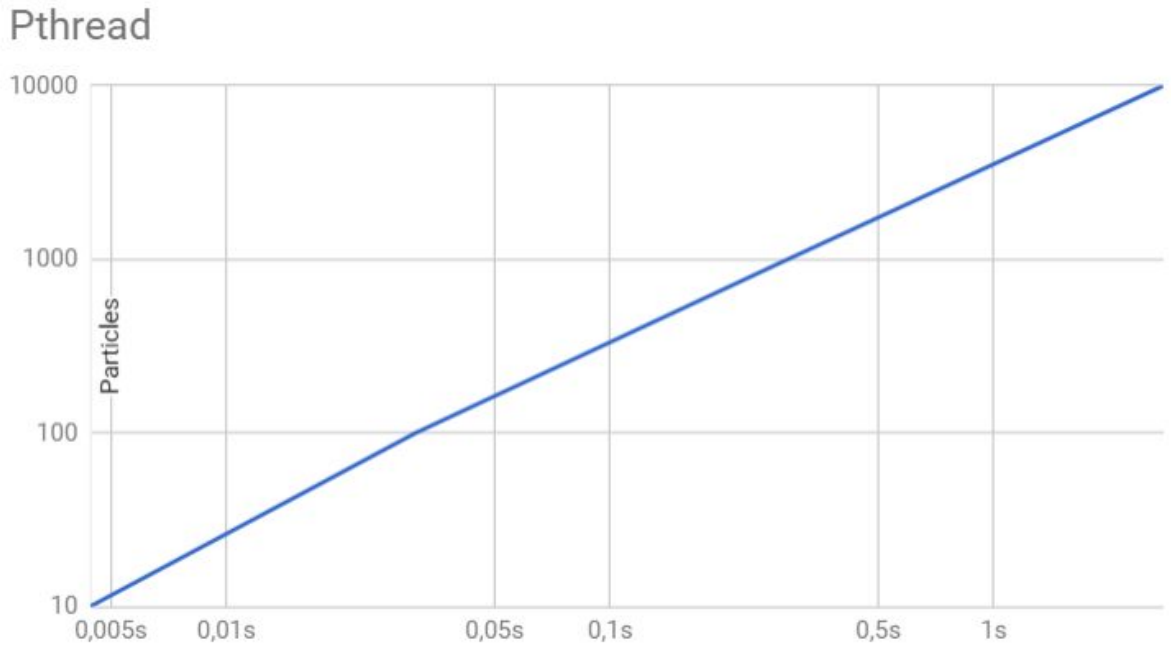

Figure 2, the Pthread program's execution time with respect to the amount of particles.

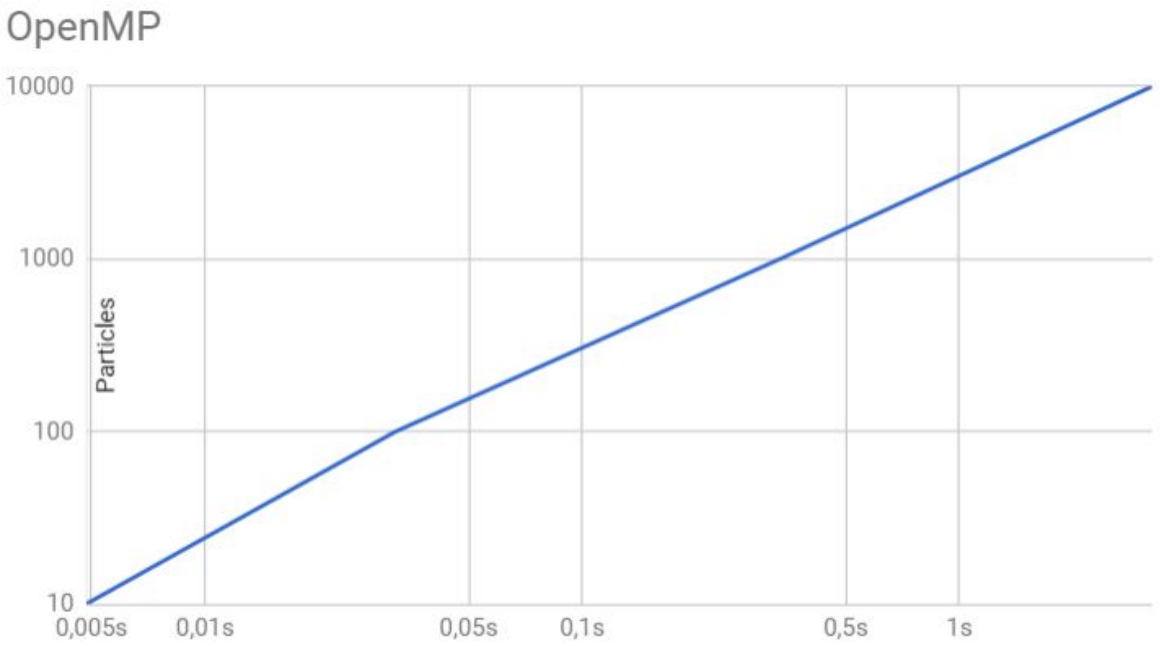

Figure 3, the OpenMP program's execution time with respect to the amount of particles.

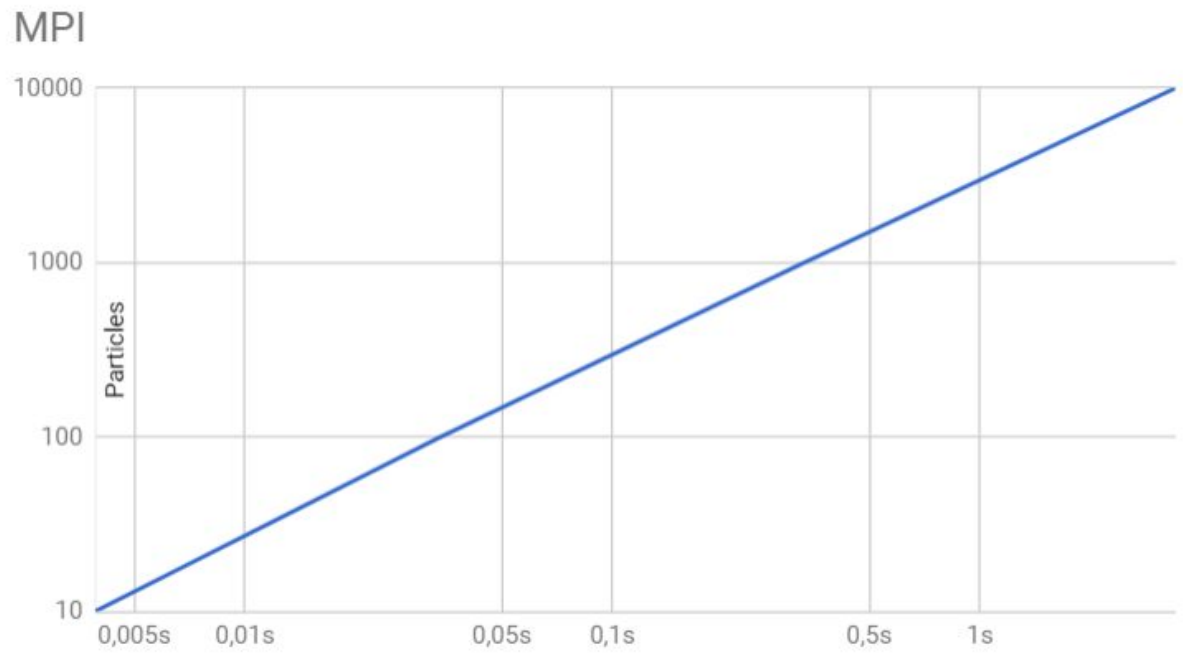

Figure 4, the MPI program's execution time with respect to the amount of particles.

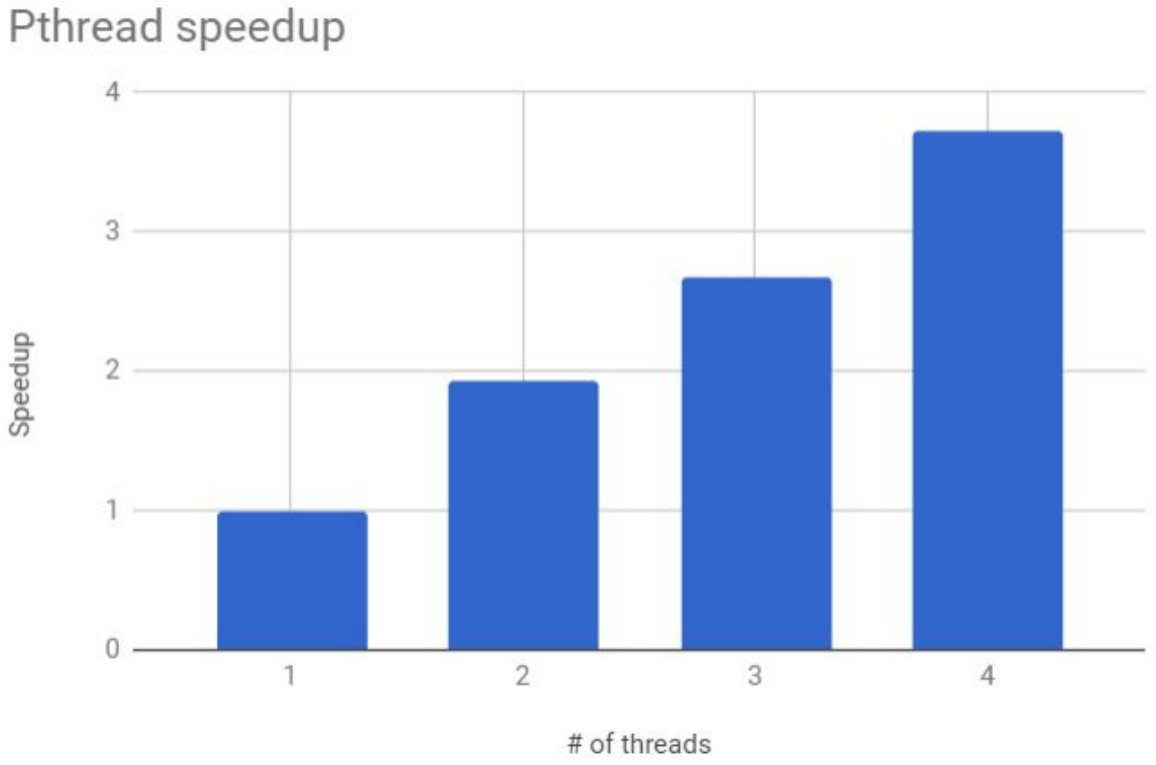

Figure 5, the Pthread program's speedup with different amount of threads.

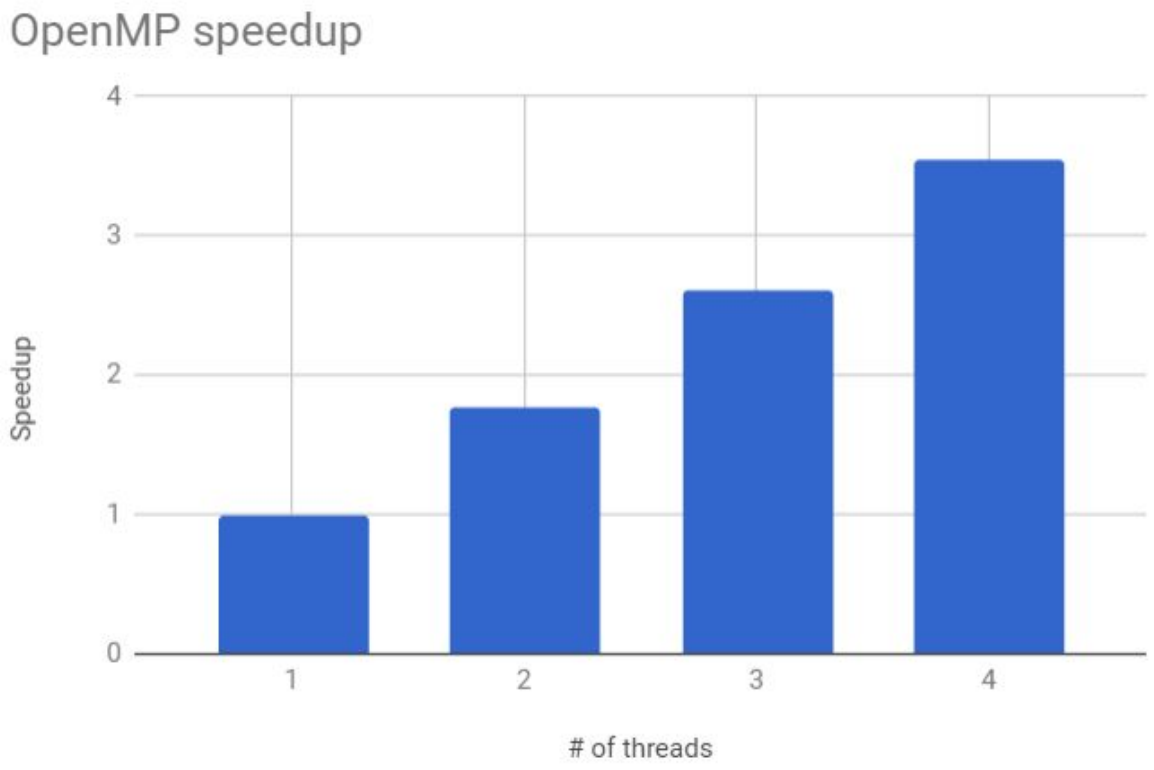

Figure 6, the OpenMP program's speedup with different amount of threads.

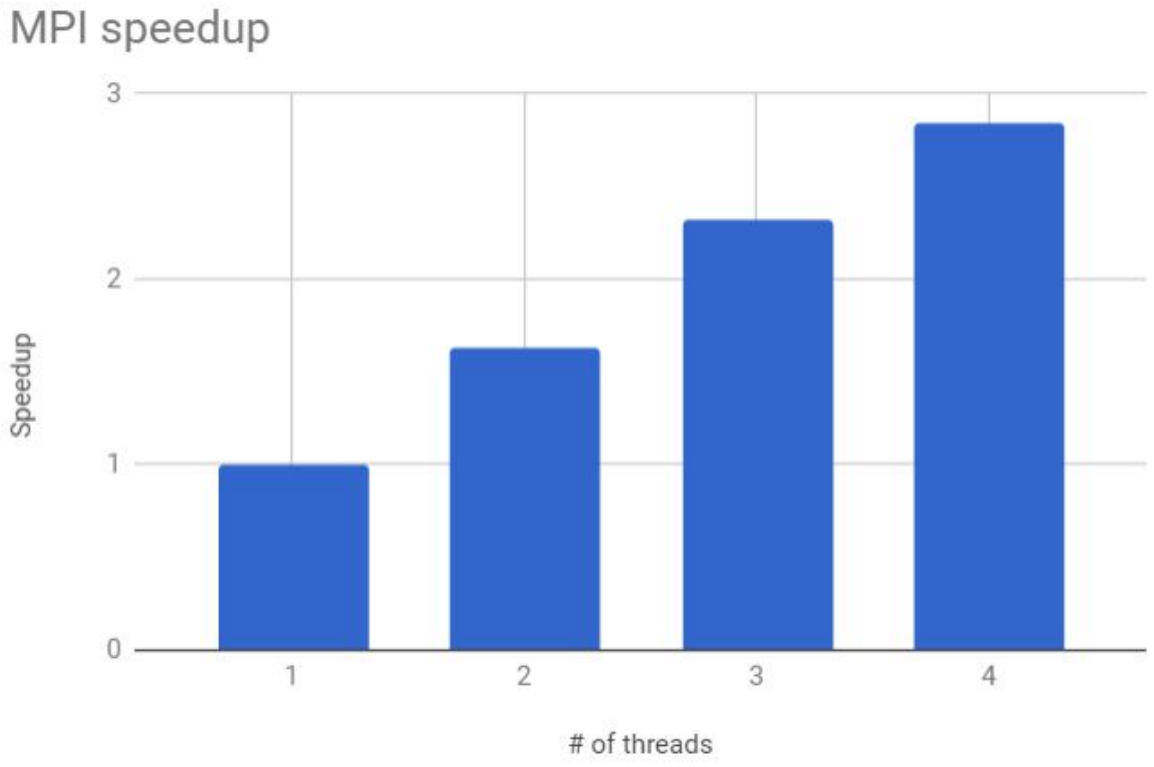

Figure 7, the MPI program's speedup with different amount of threads.