COMP 529 **Shukhrat Khuseynov (0070495)** **01.03.2020**

Assignment 1 - Part 1 (Report)

Both the task and data parallelism were implemented for the game of life. Although I could not run the gnuplot on my computer, I was printing the world matrix as a double array of ones and zeros for checking correctness. For each run the random seed was specified as 529 to have consistent and comparable output.

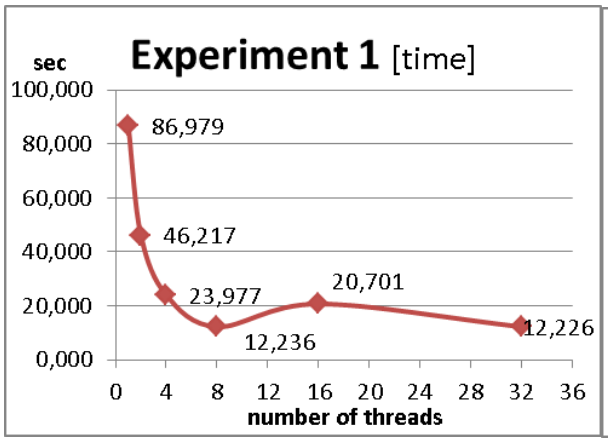
The task parallelism consists of dividing the region of main time loop to two tasks: one task for dealing with the next world calculations and one task for plotting. Although these two tasks can hypothetically be concurrent, the calculations have to be made before plotting. Therefore, taskwait was used as a barrier between two tasks, making the task parallelism an implementation bottleneck of parallel code. In other words, this part has not increased the speedup much.

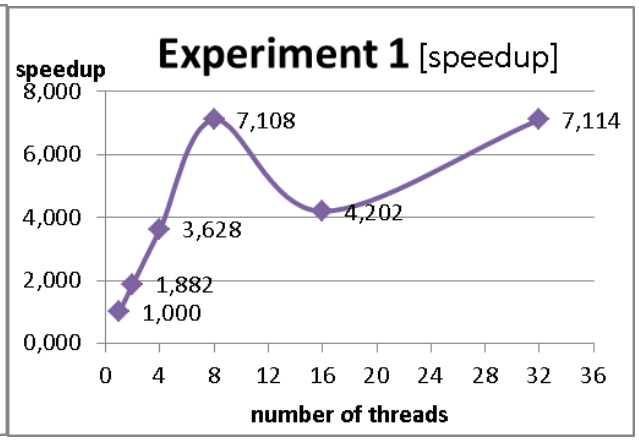
The data parallelism, on the other hand, has reasonably improved the speedup. Since the shared data of world matrix is not overwritten but only read throughout the double loop, there was no any dependency to parallelize each iteration in both dimensions easily. The only problem was population variable, being dealt with summing reduction, which made the parallel implementation slower. The static schedule was chosen to distribute the work equally among the threads.

The correctness of parallel version was ensured by running several different inputs with same random seeds and comparing the output with the serial version, assuming that the serial implementation is correct. The diff command was used, showing identical results.

Running the jobs without disabling the plotting feature indeed took comparatively much more time with respect to the jobs without activated plotting.

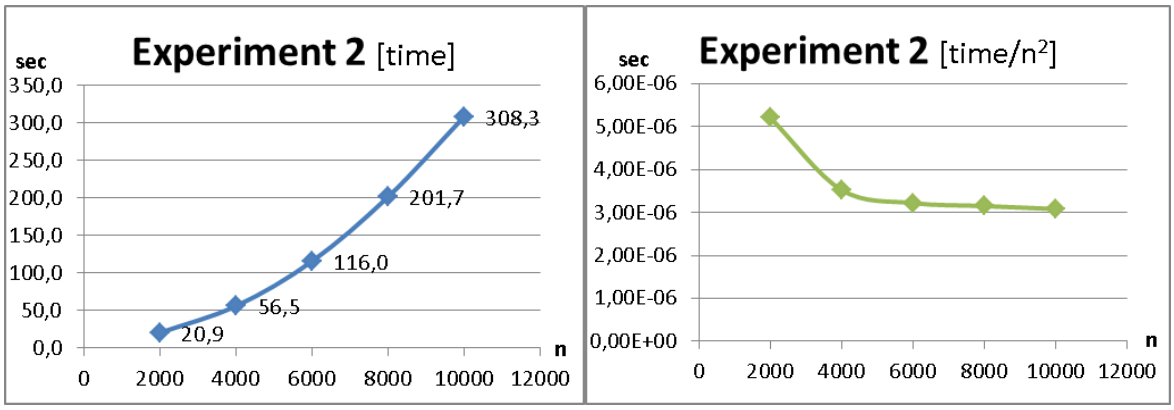
The experiments were conducted as asked in the assignment description (see game-of-life.out for the job's output). Here are the plotted graphs:





Experiment 1, being a strong scaling experiment, truly demonstrated a tendency of the running time to decrease or equivalently the speedup to increase until some point, as seen

above. For some reason the speedup decreased with 16 threads, maybe due to some parallelization overhead or some machine/architecture specificity.



Experiment 2, being more similar to weak scaling studies, depicted how time spent per each cell in the calculations of world matrix (runningtime/n^2) decreases with more threads/processors used.

To conclude, having also done some runs with the original serial code and comparing with the parallel implementation, the noticeable speedup can be observed, which is very fascinating.