ACSE-6 Group Assignment Report

Date: Feb 17th, 2021 Team Name: Old Team
Team Member: Ranran Tao, Ke Xu, Ruitong Zhang, Ziwei Zhu
Github link: https://github.com/acse-2020/group-project-old-team
Compiler used: Microsoft Visual C++ 14.28, GCC 9 OpenMP version: 2.0
Operating System: Windows 10, Mac OS X Number of logical processors: 8

1 Design of Code

1.1 Acceleration Strategy

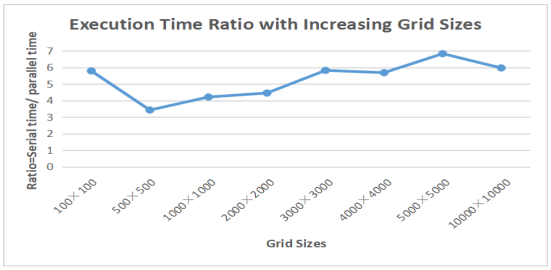
We extend and optimise the serial code which teacher provided to improve its efficiency. Our acceleration strategy consists of two main parts:
1. Using "parallel for" at suitable location to enable parallel iteration. In code comments we discuss where should/should not use parallelization in detail, please read code for more information.
2. Using "parallel sections" to let program compute the next generation while printing the last generation. Please see 2.4 of this report.

1.2 Execution and Input/Output

We write all codes in "ConwaysGame_Parallel_ppm.cpp" because it is not a large project. After compiling and execution, the program will do 100 iterations on a random 100*100 grid and output each step's image in format ppm(pbm).

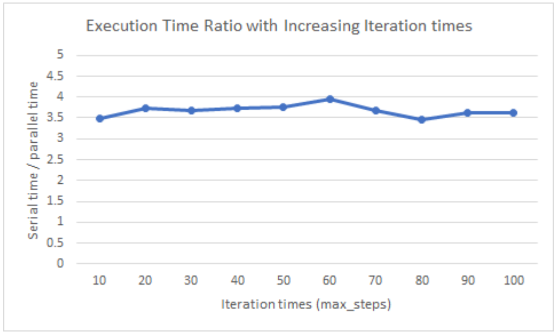
2 Analysis of Code

2.1 serial execution time/parallel execution time as a function of grid size



The above graph shows the ratio of serial execution time and parallel execution time as a function of grid size. It can be seen from the graph that the ratio slightly changes between 3.5 to 7, which means we increased the speed of computing by 3.5~7 times by parallelization. Also, the ratio generally increases when the grid size increases, which means that the optimisation becomes more evident if the grid size is larger.

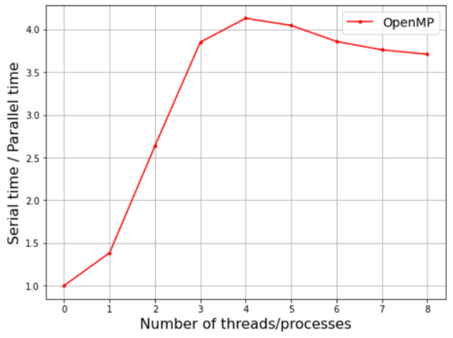
2.2 serial execution time/parallel execution time as a function of generation



This graph shows the ratio of serial time and parallel time with the variation of “max_steps” from 10 to 100 when the grid size is 1000x1000. It can be seen that the ratio tends to stabilize between 3.5 and 4.

2.3 serial execution time/parallel execution time as a function of cores

The changes of execution time with increasing number of cores are as follows.



1. We choose the 1000*1000 case matrix, under 100 generations. The reason we choose this case is that multithreading is not always a good choice, especially for a simple task. If we have two cores and split the work on two threads, this way we will have two threads working at full speed. However, threads are really expensive to create, so you need a pretty big workload to overcome the initial cost of creating the threads. In simple task, the cost of creating more threads usually overcome the benefit of distribute small tasks on different threads, so we choose a 1000*1000 case matrix, which is complex enough to show the benefit of parallel computing. We also do not want our matrix too complex (like 10000*10000), since it is time-consuming. Note the number of threads is for threads which are doing iteration, excluding the thread for printing.
2. When thread is equal to 0, the program is purely run in serial time. We can see the tendency of this curve generally follows Amdahl's Law as required.

2.4 What printing does to the execution time

Time used by different programs and grid sizes

Grid Size	Serial Program Time	Serial Program with No Printing	Parallel Program Time*
100*100	0.334	0.243	0.238
500*500	7.366	5.485	5.359
1000*1000	29.575	21.879	21.216
2000*2000	117.367	87.329	84.821

*Note: For comparison, here parallel program just printing grid while generate new_grid, no parallelization in other iteration

By doing some experiments(100 iterations) we find that printing takes about 26% total time. Although we cannot write one file in parallel, we can let do_iteration() compute next generation while grid_to_file(n - 1) prints last generation because both functions just read data from grid but don't change data in grid. Then we use implicit barrier after "parallel sections" to update the grid. From the chart we can see that this parallelization almost eliminates the extra time taken up by printing images.

3 Strengths and Weaknesses

We didn't make too many changes to the teacher's code in the end. However, we have tried many methods to improve efficiency, like change "if … else" to logical operations, use matrix operations to get the number of neighbours and so on. Those methods failed to decrease execution time and the code we commit is as fast as we can optimise.

4 Task Breakdown

Ke Xu(kx20): write, optimise image printing, explore what printing does to execution time
Ranran Tao(rt120): analysis on execution time ratio with increasing grid size
Ruitong Zhang(rz1120): analysis on execution time ratio with increasing iteration times
Ziwei Zhu(zz420): analysis on execution time with increasing number of cores