Core Programming, Data Visualisation and Analysis for Scientists Mini Project

Author: Dovudkhon Abdubokiev

Student id: 2160648

Introduction

In this project, we simulated one of the most popular cellular automata models, forest fire model. Basic rules of this model are defined as follows:

A square grid of size N is randomly filled with trees, with probability p that any given cell contains a tree. To start the fire, any living trees in the top row are then set on fire. Then, for each point on the grid at time step t:

1. any living tree next to a tree which was burning at step t 1 starts burning
2. any tree which was burning at step t 1 is now dead
3. all other sites stay in the same state (i.e. empty sites remain empty, living trees with no burning neighbours remain alive, dead trees remain dead).

A neighbour is defined as a cell to the left, right, above or below the cell we are interested in, with diagonal cells not considered to be neighbours (i.e. we are using a Von Neumann neighbourhood). The simulation proceeds until no more trees are burning. (taken from the coursework task)

I implemented this model using MPI parallelisation and BlueCrystal4.

Implementation

I have implemented the required features in the coursework.

Firstly, the code can both read in an initial file from a text file and generate random grid by itself. To choose one of this, arguments must be provided when running the code. To read in from a text file, the user has to input "./forest_fire fire grid_filename" if we consider they have a compiled file named forest_fire. To generate the grid randomly, the user has to input "./forest_fire rand N p M". Here, N is the grid row and column size, p is the probability that a given cell has a tree, M is the number of independent runs of the model.

It is also possible to average the values over M independent runs as well as running the model only once. The code outputs: Average Number of steps until the fire stops, fraction of runs where the fire reaches the bottom of the grid, and the average time spent for each run of the model. If the user chooses to read in a file or sets the argument M as 1, the algorithm gives these values for one run of the model.

The grid is distributed row-wise to ensure easy process in sharing data between tasks. Each task has its own i0 and i1 variables which determines the starting and the final row that task will update.

The most important function is the update_fire function. At each time step, every task updates their assigned rows and the necessary information is shared between the tasks using point-to-point mpi communication rather than collective mpi communications. This approach offers a big advantage as we don't have to share the full grid to every task in every step. This saves us memory and time as we don't have to store and share the redundant data that a task is not going to use anyway. However, the implementation of point-to-point communication is elegant and we have to be very careful to avoid sender and receiver mismatches and end up in a deadlock. I have based my implementation on the image blurring example that we covered during workshops. The first part of the function update the state of every "tree" by checking the state of their four neighbors. Task local variables local_burning and local_reached_bottom are updated by sharing them with all other tasks and storing them as global variables using MPI_ALLreduce().

Convergence Analysis

Using the code I have written, I have experimented several times with different quantities to achieve some meaningful results. In the first experiment, I tried to learn how the average number of steps until the fire stops changes as we have more and more independent runs of the model. For this, I fixed the grid size as N = 100 and for different values of p I have examined how the average number of steps as we increase M. As a result, we have achieved the following graph.

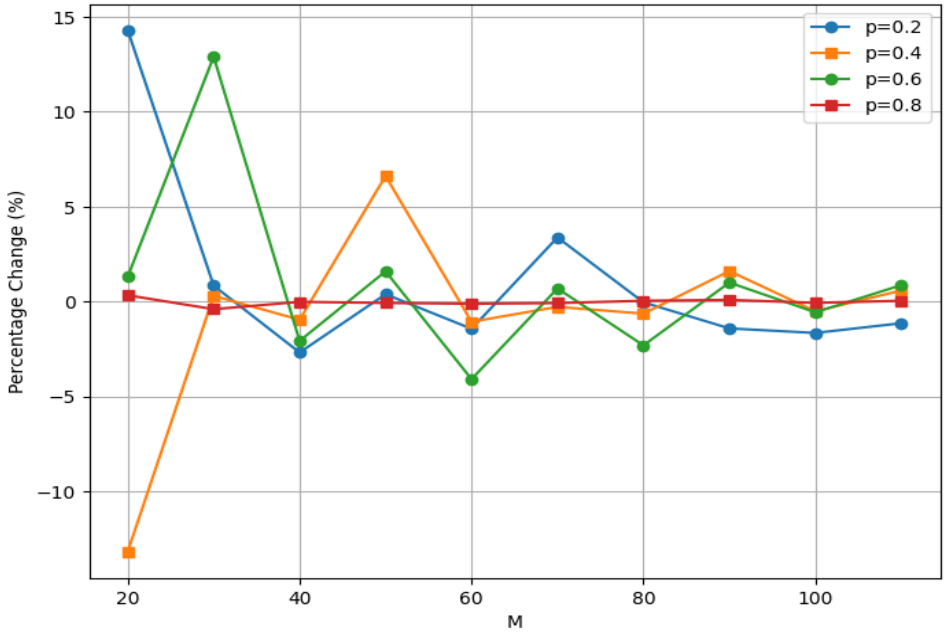

Figure 1: Percentage change in the average number of steps until the fire stops

Since the values of average steps for different values of p varied by a lot, it wasn't useful to plot the actual number of steps against M. The lines for p=0.2 and p=0.4 looked like a straight line if we plotted in that style. For reference, the values fluctuated around 4, 12, 125 and 180 for p = 0.2, 0.4, 0.6, 0.8 respectively. From figure 1, we can clearly see that the lines fluctuate a lot at first, but then slowly come close to zero as M increases. This shows that the average number of steps is not changing by a lot as M increases and possibly reaching convergence. However, at M=50, it's hard to say that any desirable convergence is achieved except for p=0.8. The lines p=0.2 and p=0.6 looks small in magnitude at M=50, but they both keep fluctuating after that. The line for p=0.8 is very stable even when M is low and achieves converge very early on. Hence, it can be concluded from the graph that p indeed matters for the speed of convergence of values. From the data we have collected, it can be concluded that as p increases the average number of steps before fire burns out becomes stable and converges early. This can be attributed to the fact that the grid is very populated when p is high and in almost all cases the fire reaches the bottom of the grid, which can be seen in the following graph.

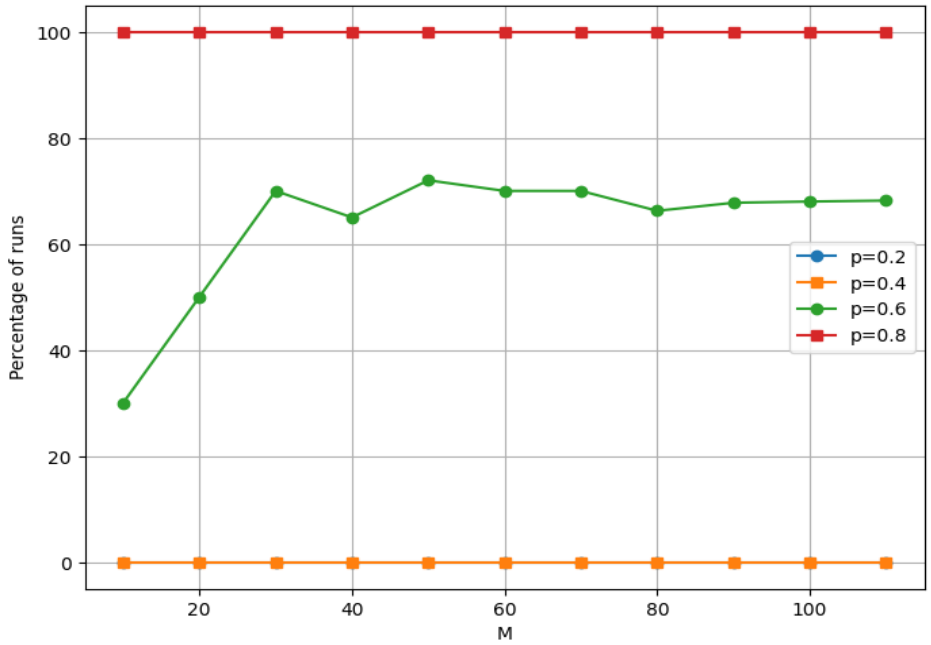

Figure 2: The percentage of independent runs in which the fire reaches the bottom of the grid

From figure 2, we can analyze how the proportion of runs in which the fire reaches the bottom of the grid converges as M increases. It is obvious that the only line fluctuating is the line p=0.6, which converge to around 70 after M=80. While the fire never reaches the bottom when p=0.2 and p=0.4, it always reaches the bottom

when p=0.8. This figure align with the idea proposed in the forest fire video, that there is a sudden jump in around 0.6 in probability of the fire reaching the bottom.

Performance Analysis

To analyze the parallel performance of the code, I have experimented with several different MPI tasks and grid sizes. To do this, I fixed p as 0.6 and M as 50. I let N be 50, 100 and 500. I ran the model several times and obtained the average times spent to run with various number of MPI tasks. To obtain the times I used MPI_Wtime() function to measure the time at different parts of the code and working out the difference.

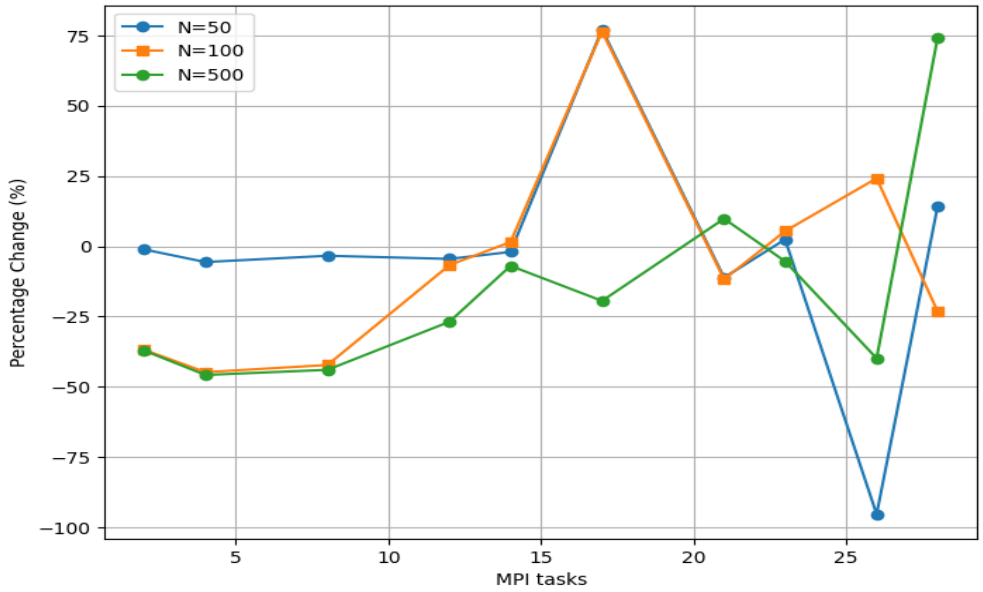

Figure 3: Percentage change in the average time spent to run a single independent run of the model.

Again because of the high difference in the time spent for different values of N, it was useless to plot the actual times spent for an individual run. Instead, I found it more useful to plot the percentage change as we increase the number of MPI tasks. It is easy to see that all figures show a negative value up to around 14 tasks, which means up to this point increasing the number of tasks decreased the time spent for each independent run. After 14 tasks, increasing the number starts to have a negative effect and increase the time spent by a large amount, around 75% at around 17 tasks. However, for N=500, the effect stays advantageous until 22 tasks. The main reason for this is that, when N is small and number of tasks is large, there isn't much load assigned to the tasks at once, but to keep the process working they still must communicate a lot. As a result, they spend much time communicating and little time doing actual work and they become ineffective. In fact, the code also has a feature to measure the time spent for calculations and communications

during a single run. Perhaps, if I plotted these values, the effect of too much time spent for communication would be even more obvious. Unfortunately, I no longer have enough time until the deadline. Possibly in the future, the project can be extended with that information and the trend would be understood better. According to the data I have I would choose 8 MPI tasks for N=50, 12 tasks for N=100 and 26 tasks for N=500.

Conclusion

In summary, we implemented a cellular automata model, forest fire model. From our model, we drew several useful conclusions. We concluded that the value of p does affect the convergence of the average number of steps until fire burns out. At p=0.6, there is a sudden jump in probability of the fire reaching the bottom of the grid. Finally, we concluded that it is not always good to have more tasks or cores in parallel programming. After some point, the time spent for communication outweighs the advantage gained by parallelization.