INFO0939 High Performance Scientific Computing

Project 1

Authors :
Joachim Paquay S154432

Teachers :
C. Geuzaine
D. Colignon
Assistant :
A. Royer

Academic year 2019-2020

1 Introduction

In this project we will have a little introduction to the parallel world by using openMP on our program. The goal of this project is to generate a lot of grids and determine the probability of conduction depending on several parameters like the fiber densities, the size of the grid (N) and the number of samples of this grid (M).

2 Description of the implementation

In order to do this project, I took the given rand function in order to have a correct generation of random numbers.

In this project I chose to work like if the array containing the grid was a matrix so in the rest of the report I will explain like we worked with a matrix even tough it is not the case. To do that, I used the function getPosition (see just under).

2.1 int getPosition(int x, int y, int N)

This function will transform the x and y input (position of a tile in a matrix) using N into the position of this tile in a single row array.

2.2 bool isConductingCell(int nb)

This function will tell us if the number associate to a cell corresponds to a conductible cell (1) or not (0).

2.3 bool isConductingGrid(int* array, int N)

This function will tell us if the grid is conducting or not.
To do that, I simply checked for each rows of the last column of the matrix if the value of the cell is 2 (where 2 stands for connected conducting material).
If this is the case, the grid is conducting, if not, the gris is not conducting.

2.4 void recurs_func(int* array, int x, int y, int N)

This function is a recursive function and will modify the array in order to connect all the conductible cells (the value pass from 1 to 2) that can be connected (i.e. if they are neighbors with a connected cell).
To do that we set the cell at position (x,y) to 2 and then if a neighbor as a value of 1, we set him to 2 and call the function for this neighbor.

2.5 void getPPM(int N, int* array)

This function will generate a ppm image corresponding to the representation of the array containing the cells.

2.6 int core(int deadline, Seed seed, int N, int nb_fibers)

This function will first generate the array (of size N*N) with only zeros.
Then for each fibers it will randomly generates a x (<N) , y (<N)and dir (<2) and it will set the value of the cell at pos (x,y) to 1.
Then if dir is equal to 0, it sets the left and right neighbor to 1 and if dir is equal to 1, it sets the two vertical to 1.
Then we loop over the leftmost cells and for each of these cells we call the recursive function.
Then we check is the grid is conductible and we generate the image if the deadline parameter is of value 0.
Finally, we return 1 if the grid is conductible or 0 if not.

2.7 int main(int argc, char** argv)

In the main, we first read the parameters from the command line, then compute the number of fibers and after that, if the flag is 1, we do the program in parallel and if the flag is 0, we do the program is sequential.

For the parallel part, we create a Seed for each tread and then we loop over M and for each iteration we call the function "core" that will return a 1 or 0 depending if the generated grid is conductible or not. Then we sum all the results from "core" and we compute the probability of conduction.

For the sequential part, we also create one Seed and we call once the "core" function and we display if the generated grid is conductible or not.

3 Study of the effect of loop scheduling on the parallel implementation

In the theoretical slides there are three types of loop scheduling :

- Static scheduling : the work will be divided in the number of threads and each thread will do a part of this work.
- Dynamic scheduling : the work will be divided in equal parts (normally there will be more parts than threads) and each thread will do a part. When a thread finished its part, it will start an other part.
- Guided scheduling : the principle is the same as the dynamic scheduling except for the fact that the size of the different parts will decrease exponentially.

To study the loop scheduling, I made a graph with a line (corresponding to the real execution time associated with the number of cores) for each type of loop scheduling (i.e. static, dynamic and guided). The program was compiled with *gcc -O2 -lm -std=c99 -fopenmp -o conducting main.c* (check the next section for the difference between O0 and O2) and launched with *./conducting 1 500 0.23 10000* for each number of threads thanks to the shell script in the *script.sh* file.

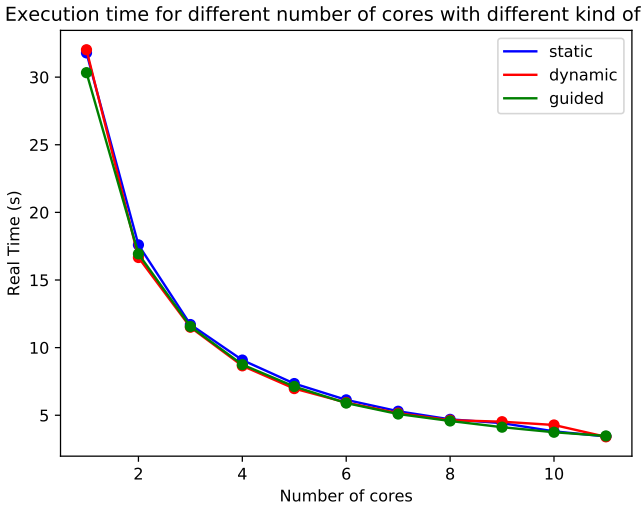

Figure 1: Execution time for different number of cores with different kind of loop

As we can see on the figure 1, there is no real difference between the different type of loop scheduling (even tho we can see that the guided scheduling seems to be a little more better), this can be explain by the fact that the workload of the CPU is more or less the same thanks to the load balancing.

4 Analyze of the efficiency of the parallel implementation

To analyse the efficiency of the parallel implementation, I collected all the real time for each execution of the code for each cores and then I plotted them.
I compiled with *gcc -O0 -lm -std=c99 -fopenmp -o conducting main.c* and *gcc -O2 -lm -std=c99 -fopenmp -o conducting main.c* and lauched with *./conducting 1 500 0.23 10000* for each number of threads thanks to the shell script in the *script.sh* file.

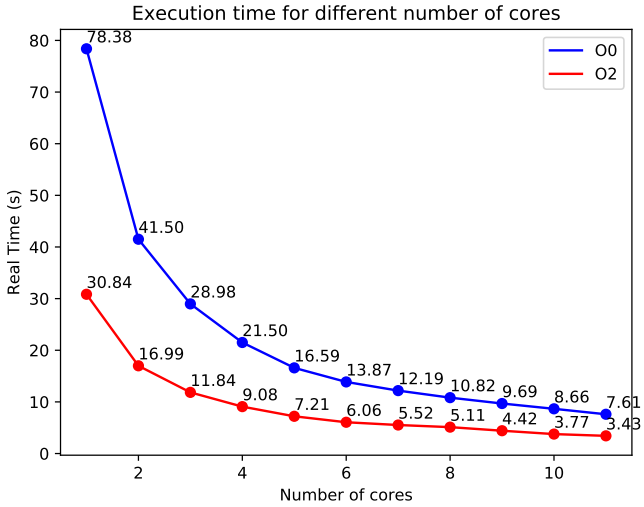

Figure 2: Execution time for different number of cores

We can clearly see that running the program in parallel is a lot better than doing it sequentially and the more cores, the better.

First we can observe that the O2 compilation is better than O0 but the general observations don't change between these two.

We can also observe that the graph as a 1/x form (where x is the number of cores) and if we multiply the number of cores by 2, we divide the time by 2 as well (we can check the pairs 2-4, 3-6, 4-8,... on the graph 2).
This result can be explain by the fact that in this project, there are no calculations except some operation in the order of $\Theta(1)$ (so this is negligible) before and after the parallel part. So, all the main operations are done in the parallel part and so it leads to the fact that if we double the number of cores, the time will be divided by 2.

Of course, the division is not perfect but this is due by the fact that if we run the program several time, there will be different results which is totally normal.

5 Evolution of the probability of conduction for fibers densities

To study the evolution of the probability of conduction for fiber densities and the effect of the parameters M and N on this evolution of the probability, we need to study some graphs. To do that, we generated some plots with N = 25, 50, 100, 200, 300 and 400 and with M = 1000 and 3000.

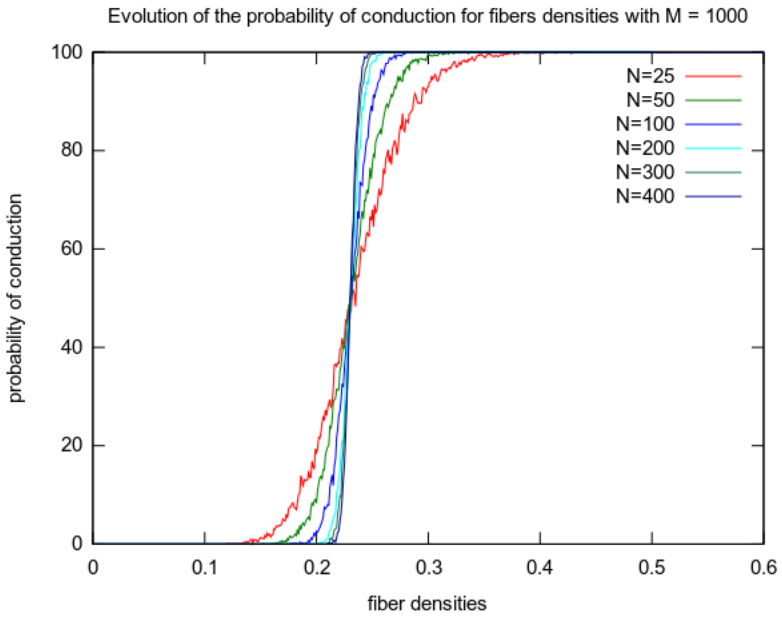

Figure 3: Evolution of the probability of conduction for fibers densities for several N (25, 50, 100, 200, 300 and 400) and several M = 1000

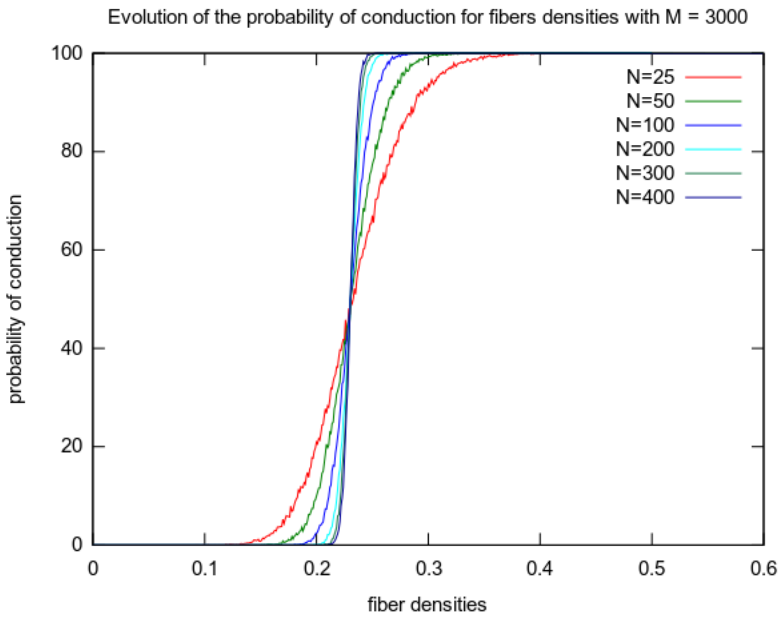

Figure 4: Evolution of the probability of conduction for fibers densities for several N (25, 50, 100, 200, 300 and 400) and M = 3000

The figure only shows the probability of conduction for fiber densities between 0 and 0.6 because we can clearly see that for all the graphs, the probability is 100% at a density of 0.4, so there is no point to go up to 1 because there will only be a probability of 100%.

From these graphs we can clearly see the influence of the parameter N and M.
For N, we can see that when it grows up, the inclination becomes more important which leads to two conclusion. In the case of a small N (25), a small density (+- 0.15) will give us some conduction and from a 0.4 density we can see that we reach the probability of 100%. (see graphs 3 and 4)
But now for a big N (400) (see graphs 3 and 4), we can clearly see that if we want to have some conduction with a small density we have to take a bigger density than N=25, the conduction begins to appear around a density of 0.22 and we will have a probability of 100% around a density of 0.25.
This can be explain by the fact that for a big N there will be a lot of fibers so there will also be a lot of non connecting fibers and so sometimes it only missed one more conducted cell to make a conductive grid and this translate the fact that the inclination is growing up with N. Indeed for a small N there will be less conducting fibers and so it reduces the chance that one more conducted cell will make a conductive grid.
For M, we can see from the graphs that its only impact is to make the graph smoother and more accurate, indeed if we have more samples, we have a better accuracy on the probability.

6 Conclusion

In conclusion, we could study the effect of parallelism in a code and that showed us the importance of this kind of programming techniques.