# INFO 0939: Project 1 – Due on 15 October 2019

(Intermediate deadline: 1 October 2019)

## Goals of the Project

- Refresh your C language knowledge.
- Experiment with the SLURM job scheduler.
- Write your first parallel program using OpenMP.
- Implement a Monte-Carlo method to model the electric conductivity of a simple composite material.

## Statement

A composite material is a material made by combining two or more constituent materials having different physical properties in order to obtain a new material with characteristics different from its constituents (see Figure 1). These composite materials can find applications in several sectors like aerospace, transportation, construction, sports, electronics, etc.

In this project, you are asked to study the influence of the density of conducting fibers in the non-conducting matrix on the electrical conductivity of the resulting composite material using a simple percolation model describe below. This algorithm must be parallelized with the OpenMP library.

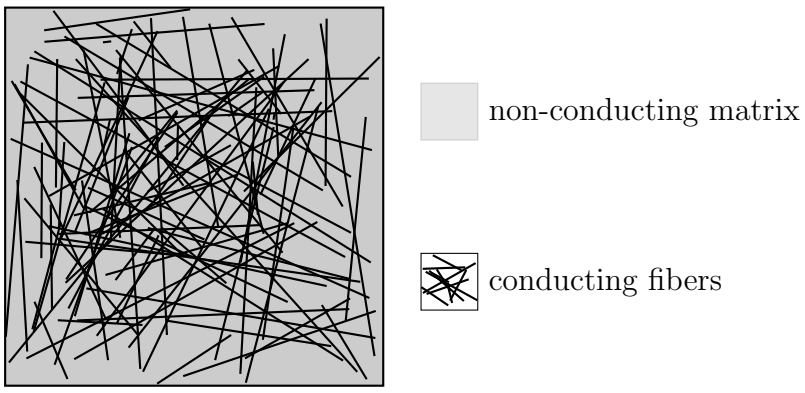


Figure 1: Example of a composite material made by non-conducting matrix with conducting fiber. This is this structure that will be study in this project.

### The percolation algorithm

Let a square grid of size $N$ made by $N^2$ square cells. Each cell could be either made of a non-conducting (matrix) or a conducting (fiber) material. A conducting cell is said to *connected* if there is a path of conducting cells connecting it to the left boundary of the domain. Otherwise, it is said to be *non-connected*. A path is defined by a set of neighboring conducting cells (by the left, the right, the top or the bottom). The square grid is said to be conducting if it exists at least one path that links the left boundary to the right boundary of the grid (see Figure 2).

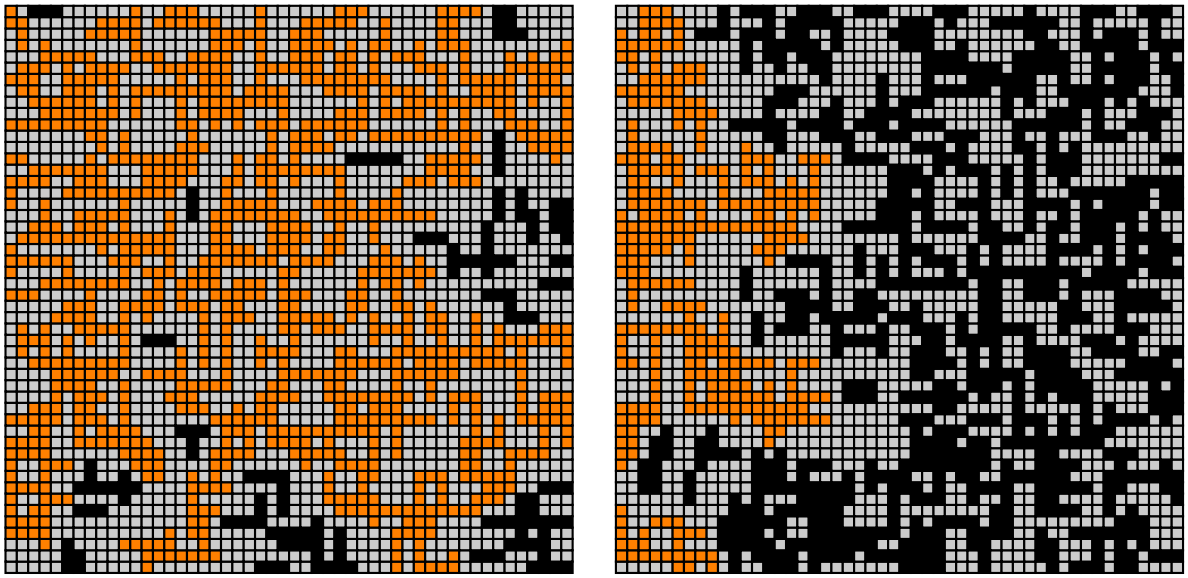

Figure 2: Examples of conducting grid (left) and non-conducting one (right). Grey cells are made of non-conducting (matrix) material, black cells are made of non-connected conducting material and orange cells are mode of connected conducting material.

Note that each fiber must have fixed length of 3 cells and that the density of fibers can be computed by dividing the number of fibers by the number of cells $N^2$.

The algorithm to solve this percolation problem is the following (input: grid size $N$ and fiber density $d$):

1. Generate the grid;
    - Generate an array of $N^2$ integers and initialize it to 0 (0 stands for non-conducting material). This array contains the cells;
2. Fill it by conducting fibers;
    - Knowing $d$ and $N$, compute the number of fibers;
    - For each fiber, generate 2 random integers (called $x$ and $y$) between 0 to $N$ and 1 random integer (called $dir$) equal to 0 or 1;

- Set the cell at $(x, y)$ to 1 (1 stands for conducting material);
- If *dir* is equal to 0, set the two horizontal neighbors to 1. Otherwise, do the same for the two vertical neighbors. Note that if the fiber is close to a boundary of the grid, just ignore the boundary and cut the fiber;
- Restart to generate another fiber if necessary;

3. Find a path by using the following recursive algorithm:

   - Create a function that takes as input the array of cells and three integers corresponding to $x$, $y$ and $N$ (the grid size);
   - In this function, if the cell is made of conducting material, assign the cell at $(x, y)$ to 2 (2 stands for connected conducting material). Then, call the same function (recursive algorithm) with new values of $x$ and $y$ corresponding to neighbors of the current cell;
   - Loop over the leftmost cells and for each cell call the previous function;
   - After looping over all leftmost cells, loop over the rightmost cells to detect if one of them is connected;
   - If a rightmost cell is connected, then the grid is conducting. Otherwise, it is non-conducting.

### The Monte-Carlo method

Knowing a grid size $N$ and an fiber density $d$, generate $M$ different grids and determine if they are conducting or not. With the number of conducting grids and the total number of grids, a measurement of the probability of conduction can be computed.

### The PPM format

The image format used in this project is the PPM format encoded as follows (see `http://netpbm.sourceforge.net/doc/ppm.html`):

1. The characters "P6" (a "magic number" for identifying the PPM file type)
2. Whitespace (blanks, TABs, CRs, LFs)
3. A width, formatted as ASCII characters in decimal
4. Whitespace
5. A height, again in ASCII decimal
6. Whitespace

7. The maximum color value (*Maxval*), again in ASCII decimal. Must be less than 65536 and more than zero. In this project this value is set to 255 (corresponding to a `unsigned char` type in C).

8. A single whitespace character (usually a newline).

9. A raster of *Height* rows, in order from top to bottom. Each row consists of *Width* pixels, in order from left to right. Each pixel is a triplet of red, green, and blue samples, in that order. Each sample is represented in pure binary by either 1 or 2 bytes. If the *Maxval* is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first. A row of an image is horizontal. A column is vertical. The pixels in the image are square and contiguous.

Lines starting with "#" are comments.

## Instructions

### Intermediate deadline

Implement a C program that

1. takes from the command line the grid size $N$ and a fiber density $d$. A call to your program must thus look like `./conducting 0 50 0.4`;

2. determines if the grid generated by your program is conducting or not, and prints a message on the terimal.

3. saves the resulting image in PPM format.

For this deadline, the program should not be multithreaded. Only a working sequential version is needed.

### Project deadline

With codes written for the intermediate deadline, implement a C program that

1. takes from the command line a flag $f$, the grid size $N$, the fiber density $d$, and the number of samples $M$. A call to your program must thus look like `./conducting 1 50 0.4 1000`. If the flag $f$ is equal to 0 the program should run the code of the intermediate deadline, otherwise it should run the code of the final deadline.

2. determines the probability of conduction, and print it to the screen.

This C program should be run on multithreaded architectures. To this end, you should write a shell script that reserves a given number of cores on the supercomputer (NIC4), with appropriate memory and job duration. Before launching your code in parallel on multiple cores, make sure that it works fine with only one core.

Write a report of maximum 5 pages where you:

1. describe your implementation;
2. study the effect of the loop scheduling on your parallel implementation;
3. analyze the efficiency (strong scaling) of your parallel implementation;
4. build a graph showing the evolution of the probability of conduction for fiber densities between 0 and 1.
5. discuss the obtained results and the effect of parameters $N$ and $M$ on this evolution of the probability.

Submit your C code on the Montefiore submission platform `https://submit.montefiore.ulg.ac.be/`. Note that no errors have to be reported during the automatic tests performed on this platform. **If your last submission generates error(s), a default grade of 0/20 will be attributed.**

When your code passes on the submission platform, send your report in PDF format together with your C code and SLURM submission script to `anthony.royer@uliege.be` and `cgeuzaine@uliege.be`. The files (report, C code, SLURM script) should be named

```
project1_Lastname.pdf
project1_Lastname.c
project1_Lastname.sh
```

## Remarks

The number of arguments and their values are passed as arguments (`argc` and `argv`) to the `main` routine: `int main(int argc, char **argv)`. For `./conducting 50 0.5 1000`; `argc` will be 4, `argv[0]` will be `"./conducting"`, `argv[1]` will be `50`, `argv[2]` will be `0.5` and `argv[3]` will be `1000`.

If your operating system cannot natively display PPM images, you can use the following commands in Matlab: `img=imread('file.ppm'); image(img);`.

Make your code easily readable by other people and by "future you". This implies to:

- correctly indent your code;
- make things as simple as possible... while still being efficient;

- use pertinent, meaningful variable and function names, even if it makes them longer;
- make functions when necessary.

As usual, don't write the code all at once without testing. Write it part by part and test every small functionality separately.