Study of parallel algorithms applied to the Heat diffusion problem

Yoann Masson
Student in Software Engineering at Cranfield University

1st of February, 2018

Abstract

This report illustrates the work I have conducted to use parallel algorithms to solve the heat conduction equation.The equation is given by the following formula:

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{1}$$

Theses algorithms are running on multiple processors allowing multiples calculations to be performed at the same time by different processors. However, in the process of resolving this equation we need to use results computed by other processors, that is because the computation of a given time in space requires the temperature of its neighbor points in space. The problem is that communication between processors is the most costly operation in parallel programming.
Throughout the report, we will see that parallel algorithm can drastically improve the processing time if well used and if needed. As we will see with the explicit solver, being not complex enough, it is actually computed faster with only one processor because computing serially the result is faster than the communication time implied by multi-processors programming. Implicit solvers need to solve a PDE for each time steps, making parallel programming much more relevant and interesting.

Contents

Introduction

The processing power of computer has, without a doubt, incredibly increased over the last decades pushing scientists to conduct more and more complex and large programs on computers. However, each time the processing power increased the problem became bigger and more complex because we always want to do better and faster. A way to overcome the current capacity of computer is: multi processors programming.
It consists in splitting tasks among processors to reduce the time it takes to run the program. In theory we can expect a to be twice more efficient with twice the number of processors. But it does not work like this because every processor has its own memory space, so when a processor needs a result from another processor: the result has to be transferred, inferring communication time that is really costly.
With that in mind, we can say that there is two factors to keep in mind when choosing the number of processor:

- the size of the problem
- the number of communication between processors

In the example we will conduct, we will face the heat conduction equation problem. This is a problem where the solution is a matrix with one coordinate being time and the other being space. The general formula is:

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{2}$$

where t is time and x is space.

In this report we will study through explicit and implicit solvers the impact of communication costs, implied by the chosen number of processors. We will see how theses chooses impact an explicit solver (Forward in time, central in space) and two implicit solver (Laasonen and Crank-Nicholson).

Chapter 1

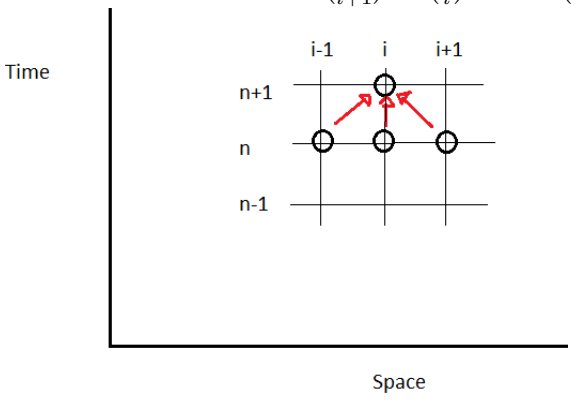
Explicit solver: FTCS

1.1 Theory

The forward in time central in space, FTCS, explicit solver requires just simple operations to compute points in time and spaces, the formula becomes (see more in the appendix):

$$f\binom{n+1}{i} = D\frac{\Delta t}{\Delta x^2}(f\binom{n}{i+1} - 2f\binom{n}{i} + f\binom{n}{i-1})) + f\binom{n}{i} \tag{1.1}$$

This equation means that to compute the next time step $f\binom{n+1}{i}$ we need the previous results from $f\binom{n}{i+1}$, $f\binom{n}{i}$ and $f\binom{n}{i-1}$.



The basic technique of multi-processors programming is splitting the tasks between multiples processors meaning that the result matrix is going to be split. But how ? it could be horizontally, vertically and in cube where every processor is in charge of one part of the matrix, see figures below.

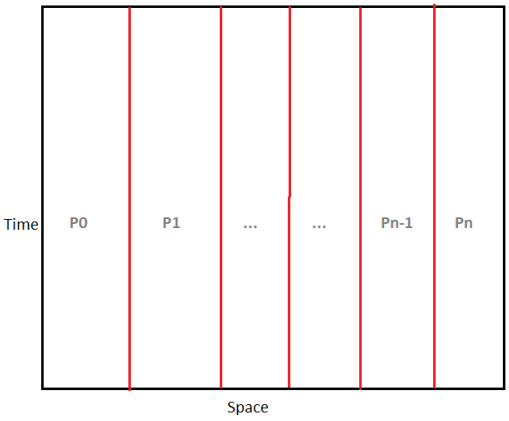


Figure 1.1: Vertical split

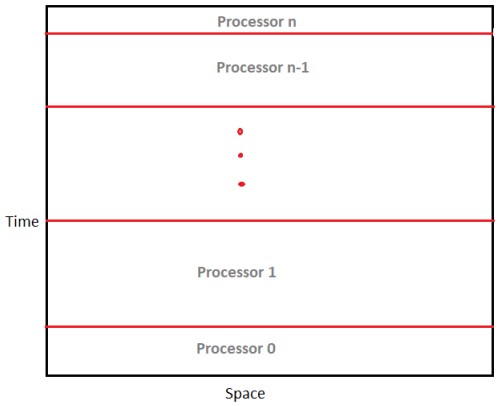


Figure 1.2: Horizontal split

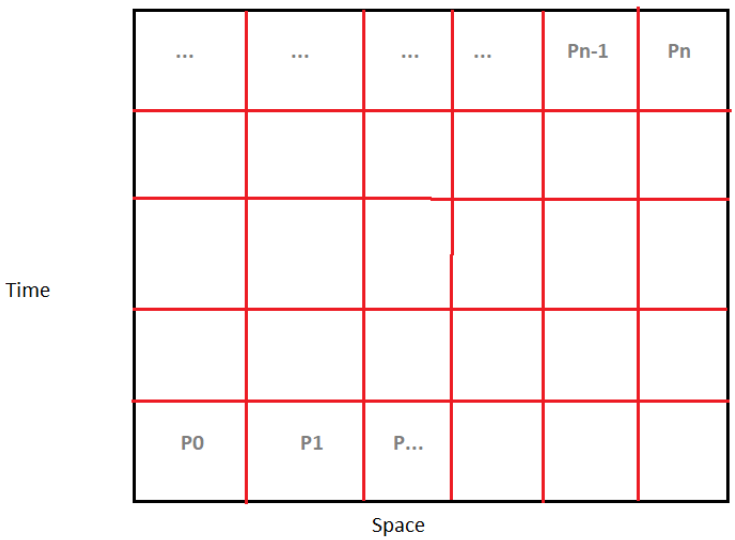


Figure 1.3: Cube split

To chose a way of dividing the matrix, we need to take a look of the dependencies between the different point in time and space, the stencil. Looking at the equation stencil, it is clear that if we split the matrix horizontally (meaning each matrix would take care of a time step and every space step) the performance would not increase because for one processor to compute it, it would have to wait for the previous processor to finish its job. Making the program serial because only one processor would be working at a time.
So we will be splitting the matrix vertically. It implies that when reaching a "boundary", the processor has to receive and send a result to its neighbor processor.

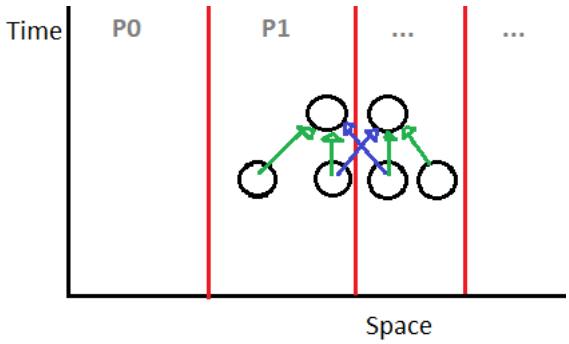


Figure 1.4: FTCS stencil

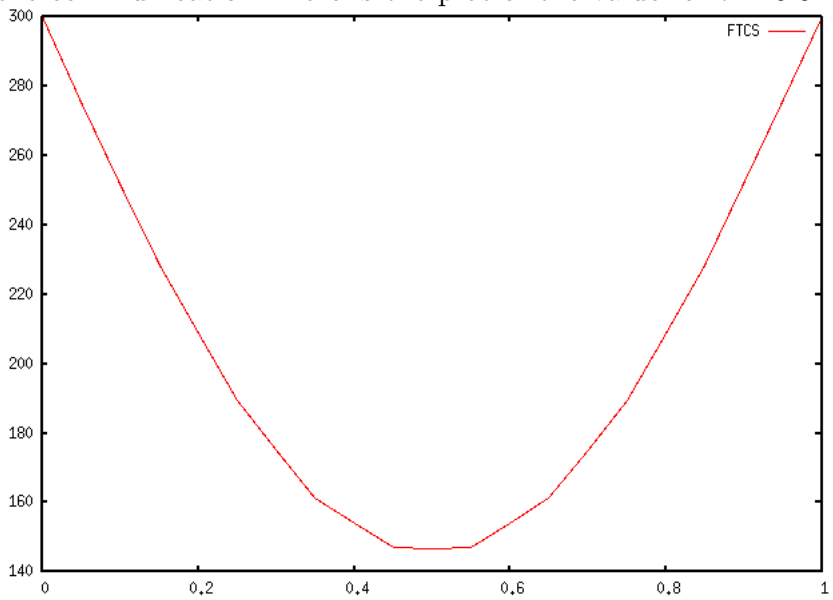
The above picture represents how the work is divided in this particular case, the green arrow are value that does not need to be transferred because the processor already has the correct value meanwhile the blue arrows represents value that are transmitted from one processor to another.

1.2 Results

As discuss earlier, there is a dilemma between the gain of processing time and the inferred communication costs. The more processor you have, the smallest the sub matrices are but the more communication there is.
Every problem has its own balance, let's see how this apply in this case with a 21*100 matrix (Δx=0.05 and Δt=0.01).

1.2.1 Numerical value

The result from the parallel program are the same as the one with the serial program which is a good thing, because some precision can be lost during the communication. Here is the plot of the value for $t = 0.5$.



1.2.2 Time performance

We can assume that the time for the serial program and the time for the parallel program with one processor is the same. After verification, it is. The time is 0.1 ms on average of 10 tries.
The important fact to look at, when considering time performance, is strong scalability and weak scalability. Strong scalability is the aptitude of a program to gain time by adding more CPU.

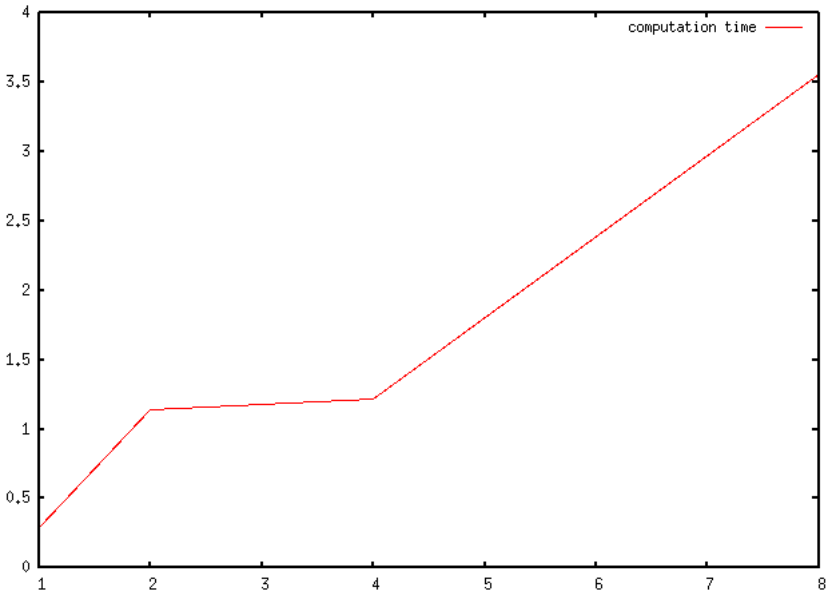


Figure 1.5: Strong scalability FTCS

The above figure shows the computation time per processor used. The tendency is clear, the more processor there is the less performance we get. It can be explained by two reasons:

- the problem is not complex enough to require multi processor programming
- the problem is too small

To see what it is, we need to study weak scalability. This is the aptitude for the program to follow the size of the problem with the number of processor. In other words, if we double the problem we should double the number of processor and expect no change in time.

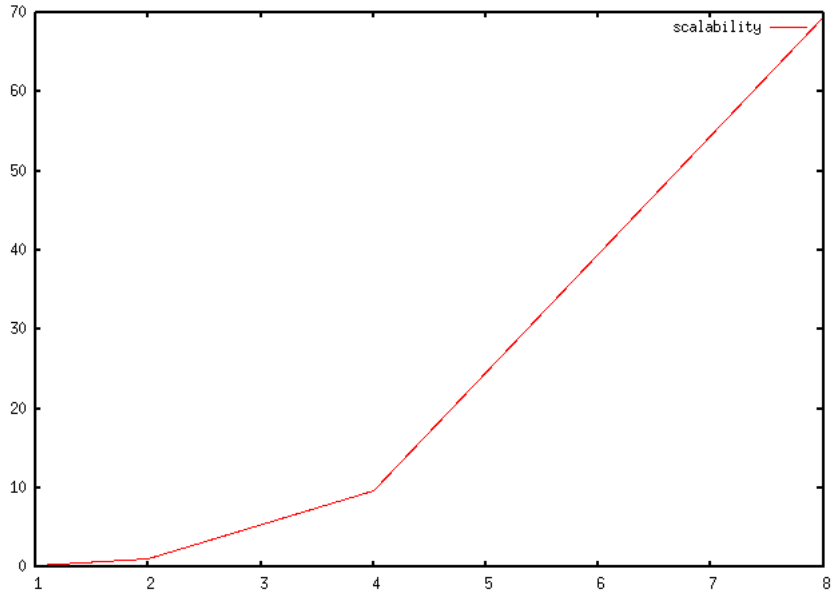


Figure 1.6: Weak scalability FTCS

We expected the computation time not to change too much, but the trend in processor number does not match the size of the problem.
In conclusion, we can say that the explicit solver is not fit for multi-processor programming because the problem is not complex enough and communicating results takes longer than actually computing it.

Chapter 2

Implicit Solver

2.1 Theory

Once the implicit solver applied the formula becomes a PDE to solve (see more mathematic details in the appendix), this is laasonen example:

$$\begin{pmatrix} 1+2C & -C & 0 & ... & 0 \\ -C & 1+2C & -C & ... & 0 \\ ... & ... & ... & ... & ... \\ 0 & ... & -C & 1+2C & -C \\ 0 & ... & 0 & -C & 1+2C \end{pmatrix} \begin{pmatrix} f\binom{n+1}{0} \\ f\binom{n+1}{1} \\ ... \\ ... \\ f\binom{n+1}{k} \end{pmatrix} = \begin{pmatrix} f\binom{n}{0} \\ f\binom{n}{1} \\ ... \\ ... \\ f\binom{n}{k} \end{pmatrix} \quad (2.1)$$

Contrary to explicit solvers, implicit solvers are more complex to solve. That is because, implicit solver relies on previous time step but also on current time step. The figures below show the stencil of Crank-Nicolson and Laasonen implicit solver. INSERT STENCIL. We can not split the matrix as we did with the explicit to split the task in this case, what I am using is a parallel version of thomas algorithm
(***reference:*** *http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.706.6668&rep=rep1&type=pdf p. 367*)

Chapter 3

Implementation

Now that's we saw the theory and the results, let's take a look at the C++ implementation and the design of the application.

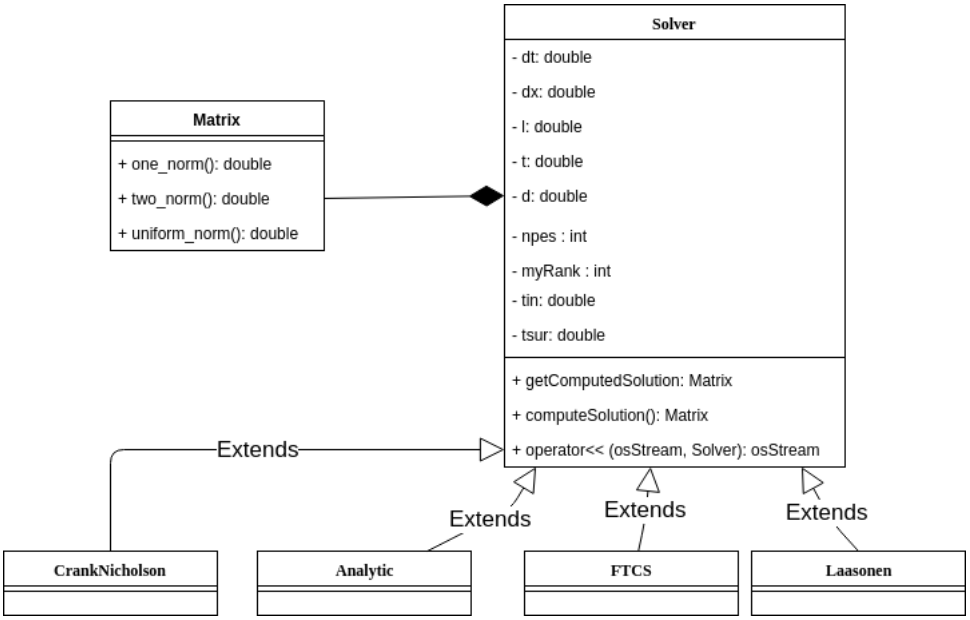


Figure 3.1: Class Diagram

Class Diagram This is the class diagram of the application implementing the three schemes and the analytic solution. The*Solver* class uses *Matrix* to store the solution of the problem where rows are time steps and columns are space steps. *Solver* declares the virtual method *computeSolution()* so that classes that inherit from it need to give an implementation of *computeSolution()* if they wish to be instanciated in a *main* program.

The base class *Solver* class holds all of the information regarding the initial problem. Since it is attribute and not just hard-coded value in the code, all the child class are bounded to those values. Let's say that we want to see the solution with a surface temperature of 250°F instead of 300°F, we just have to instantiate the object with a different *Tsur* value.

functions To print out the solution both in screen or in a file, I added the redefinition of the " << " operator. On screen, the matrix will print the solution value for time 0.5 and will write on a file in a way that would be exploitable by GNUPlot. I added the minus operator in the matrix class to be able to subtract matrices in order to get the error matrix.
When instantiated an object of a Solver child class, you have to respect some rules in order not to get exceptions. For example, it is impossible to have a dx larger than a L it would not make any sense, so the program would raise an *invalid_argument* exception

Appendix A

Mathematics derivation

Dufort-Frankel

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{A.1}$$

$$\frac{f\binom{n+1}{i} - f\binom{n}{i}}{\Delta t} = D\frac{f\binom{n}{i+1} - 2 * f\binom{n}{i} + f\binom{n}{i-1}}{\Delta x^2} \tag{A.2}$$

$$f\begin{pmatrix} n+1 \\ i \end{pmatrix} = D\frac{\Delta t}{\Delta x^2}(f\begin{pmatrix} n \\ i+1 \end{pmatrix} - 2f\begin{pmatrix} n \\ i \end{pmatrix} + f\begin{pmatrix} n \\ i-1 \end{pmatrix})) + f\begin{pmatrix} n \\ i \end{pmatrix} \tag{A.3}$$

Laasonen

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{A.4}$$

$$\frac{f\binom{n+1}{i} - f\binom{n}{i}}{\Delta t} = D\frac{f\binom{n+1}{i+1} - 2f\binom{n+1}{i} + f\binom{n+1}{i-1}}{\Delta x^2} \tag{A.5}$$

$$f\begin{pmatrix} n \\ i \end{pmatrix} = -Cf\begin{pmatrix} n+1 \\ i+1 \end{pmatrix} + (1+2C)f\begin{pmatrix} n+1 \\ i \end{pmatrix} - Cf\begin{pmatrix} n+1 \\ i-1 \end{pmatrix} \text{ with } C = D\frac{\Delta t}{\Delta x^2} \tag{A.6}$$

$$\begin{pmatrix} 1+2C & -C & 0 & ... & 0 \\ -C & 1+2C & -C & ... & 0 \\ ... & ... & ... & ... & ... \\ 0 & ... & -C & 1+2C & -C \\ 0 & ... & 0 & -C & 1+2C \end{pmatrix} \begin{pmatrix} f\binom{n+1}{0} \\ f\binom{n+1}{1} \\ ... \\ ... \\ f\binom{n+1}{k} \end{pmatrix} = \begin{pmatrix} f\binom{n}{0} \\ f\binom{n}{1} \\ ... \\ ... \\ f\binom{n}{k} \end{pmatrix} \tag{A.7}$$

Crank Nicolson

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{A.8}$$

$$\frac{f\binom{n+1}{i} - f\binom{n}{i}}{\Delta t} = D\frac{f\binom{n}{i+1} - 2f\binom{n}{i} + f\binom{n}{i-1}}{2\Delta x^2} + D\frac{f\binom{n+1}{i+1} - 2f\binom{n+1}{i} + f\binom{n+1}{i-1}}{2\Delta x^2} \tag{A.9}$$

$$-Cf\binom{n+1}{i+1} + (1+2C)f\binom{n+1}{i} - Cf\binom{n+1}{i-1} = Cf\binom{n}{i+1} + (1-2C)f\binom{n}{i} + Cf\binom{n}{i-1} \tag{A.10}$$

$$\begin{pmatrix} 1+2C & -C & 0 & ... & 0 \\ -C & 1+2C & -C & ... & 0 \\ ... & ... & ... & ... & ... \\ 0 & ... & -C & 1+2C & -C \\ 0 & ... & 0 & -C & 1+2C \end{pmatrix} \begin{pmatrix} f\binom{n+1}{0} \\ f\binom{n+1}{1} \\ ... \\ ... \\ f\binom{n+1}{k} \end{pmatrix} = \tag{A.11}$$

$$\begin{pmatrix} Cf\binom{n}{1} + (1-2C)f\binom{n}{0} + C * Tsurface \\ Cf\binom{n}{2} + (1-2C)f\binom{n}{1} + Cf\binom{n}{0} \\ ... \\ ... \\ C * Tsurface + (1-2C)f\binom{n}{k} + Cf\binom{n}{k-1} \end{pmatrix} \text{ with } C = D\frac{\Delta t}{\Delta x^2} \tag{A.12}$$