High Performance Technical Computing

Augustin Reille

2 November 2017

Abstract

A one-space dimensional problem is considered, to examine the application of distributed memory parallel programming techniques for the numerical solution of the heat conduction equation.

Contents

List of Figures

List of Tables

Listings

Nomenclature

T_i^n . . . Temperature at space i and time n
D . . . . Diffusivity of the wall
L . . . . Total length of the wall
T . . . . Total time of the study
Δx . . . Space step
Δt . . . Time step
δx . . . . partial derivative according to space
δt . . . . partial derivative according to time
n_{space}. . . number of iterations over space
n_{pes}. . . . number of processors

1 Introduction

A PDE (Partial Differential Equation) is an equation which includes derivatives of an unknown function, with at least 2 variables. PDEs can describe the behavior of lot of engineering phenomena [1], that's why it is important to known how to resolve them computationally. One way to resolve PDEs is to discretize the problem into a mesh. Applying finites differences on some points of the mesh, we can find a function which is more or less the same than the unknown function described by the PDE.
To find the unknown function, applying differences on a mesh work differently according to the difference chosen, but also according to the chosen mesh. For a one-dimensional problem, time steps and space steps can impact the accuracy of the result found. Also, the chosen differences affect the result. A discretisation scheme is the way we choose to apply finites differences.
In this study we are going to see how the accuracy of known schemes are evolving for a one-dimensional problem, and how the meshing impact the accuracy of those schemes.

2 Methods

Our study results are to be stored in matrixes. Several schemes are used :

- FTCS simple explicit (forward time, central space, explicit)
- Laasonen simple implicit (forward time, central space, implicit)
- Crank-Nicholson (Trapezoidal)

For all of the used schemes, the initialization of the result matrix is run the same way. First the study grid must be chosen. The initials conditions are:

- T_{int} of 100°F
- T_{sur} of 300°F
- $D = 0.1 ft^2/hr$

For each methods two sets of step size are to be used:

1. $\Delta x = 0.5$ and $\Delta t = 0.1$
2. $\Delta x = 0.005$ and $\Delta t = 0.001$

so that the result matrix could be initialized. We set:

$$n_{space} = \frac{L}{\Delta x} + 1 \tag{1}$$

$$n_{time} = \frac{T}{\Delta t} \tag{2}$$

n_{space} is the number of iterations over the grid, according to the coordinate x. n_{time} is the number of iterations over the grid, according to the time t.

$$\begin{array}{cccccc} i=0 & i=1 & \cdots & n_{space}-1 & n_{space} & \\ \begin{pmatrix} T_{ext} \\ T_{ext} \\ \vdots \\ T_{ext} \end{pmatrix} & \begin{pmatrix} T_{int} \\ 0 \\ \vdots \\ 0 \end{pmatrix} & \cdots & \begin{pmatrix} T_{int} \\ 0 \\ \vdots \\ 0 \end{pmatrix} & \begin{pmatrix} T_{ext} \\ T_{ext} \\ \vdots \\ T_{ext} \end{pmatrix} & \begin{array}{c} j=0 \\ j=1 \\ \vdots \\ n_{time} \end{array} \end{array}$$

This is how the matrix is initialized, according to initial conditions.

2.1 FTCS simple explicit

In order to perform the parallelization, we assume that the number of space steps can be divided by the number of processors. The number of space steps divided by the number of processors defines the new space step for each processor:

$$n'_{space} = \frac{n_{space}}{n_{pes}} \tag{3}$$

The stencil for FTCS explicit scheme is the following:

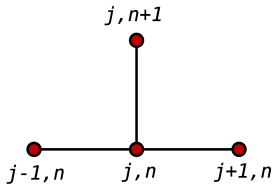

Figure 1: Stencil for FTCS explicit scheme

That means that to calculate a temperature for a time step and a space step, we need some previous values in time : the value in space, which we can access easily, and the previous and next values in space. That was the main problem for parallelizing this problem. The way we managed it is the following.
For each processor, a vector of size $[n'_{space}+2]$ is created. The size is incremented by 2 because each vector will contain 2 "ghost" values (will be explained later). The first vector will be our boundary values vector, so we fill this vector with T_{int}.
For the first and last processor respectively, the first and last value of the vector are replaced by T_{ext}.
For example, for $n_{space} = 12$ and $n_{pes} = 4$, at $t = 0h$ we have :

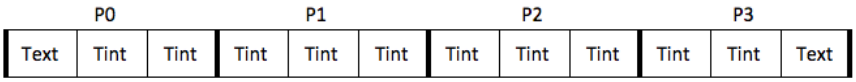

Figure 2: Separation of values per processors at t=0h

At each time step, each processor perform 4 MPI actions:

- it sends its first value to the previous processor (MPI_Send())
- it receives the first value of the next processor (MPI_Receive())
- it sends its last value to the next processor (MPI_Send())
- it receives the first value of the previous processor (MPI_Receive())

N-B : In the previous explanation, ghost values are not considered : they are only used to store the values received from neighbour processors.
In Figure 3, we can see which values are exchanged at the boundaries, in order to calculate the next boundary value. This pattern is due to the stencil : to continue with the example of the Figure 3, in order to calculate the value of $h[1]$ at time step $t + \Delta t$, the processor needs the $h[0]$ value, which comes from the previous processor.

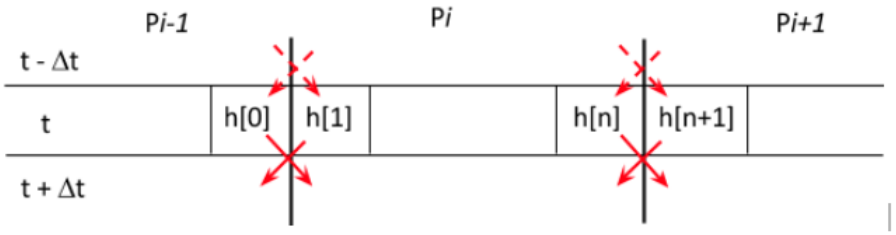

Figure 3: Exchanges between processors at each time step

Calculated values are gathered (MPI_Gather()) in the final result matrix, which is stored and printed in root processor.

2.2 Implicit schemes

For both implicit schemes, the vector solution is given by a matrix equation. Thomas algorithm is used to resolve the system for each space step.

2.2.1 Laasonen

We have :

$$\frac{T_j^{n+1} - T_j^n}{\Delta t} = D\frac{T_{j+1}^{n+1} - 2T_j^{n+1} + T_{j-1}^{n+1}}{\Delta x^2} \tag{4}$$

Written in a matrical way, the solution at space step $n+1$ is given by :

$$\begin{bmatrix} 1+2C & -C & 0 & \cdots & 0 \\ -C & 1+2C & -C & \ddots & \vdots \\ 0 & -C & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1+2C & -C \\ 0 & \cdots & 0 & -C & 1+2C \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_i \\ \vdots \\ T_{ntime} \end{bmatrix}_{n+1} = \begin{bmatrix} T_1 + CT_0 \\ T_2 \\ \vdots \\ T_i \\ \vdots \\ T_{ntime} + CT_f \end{bmatrix}_n \tag{5}$$

with

$$C = \frac{D\Delta t}{\Delta x^2}$$

The first vector at space n is our boundaries conditions vector, so it is known. In this scheme we must be aware that the right-term vector is actually bigger than the other, of 2 values, which corresponds to the exterior temperatures, which stays the same all the time.
In serial, we apply Thomas algorithm for each space step, at the reduced vector, and add the $n+1$ vector to our result matrix. The new right term vector is the one found with the Thomas algorithm.
There was several ways to parallelize this study. The one I chose is to parallelize Thomas algorithm. The algorithm I used is described in [5] and in [6]. The main idea is to make a recursive procedure with a LU decomposition. We must note that this algorithm will work with a matrix with differents tridiagonal values.

$$\begin{bmatrix} a_1 & c_1 & 0 & \cdots & 0 \\ b_2 & a_2 & c_2 & \ddots & \vdots \\ 0 & b_3 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & a_{N-1} & c_{N-1} \\ 0 & \cdots & 0 & b_N & a_N \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ l_2 & 1 & & 0 & \\ & l_3 & \ddots & & \\ & 0 & \ddots & 1 & \\ & & & l_N & 1 \end{bmatrix} \begin{bmatrix} d_1 & u_1 & & & \\ & d_2 & u_2 & 0 & \\ & & d_3 & u_3 & \\ & 0 & & \ddots & \ddots \\ & & & & d_N \end{bmatrix} \tag{6}$$

Then we have the relations between coefficients :

$$a_1 = d_1 \tag{7}$$

$$c_j = u_k \tag{8}$$

$$a_k = d_k + l_k u_{k-1} \tag{9}$$

$$b_k = l_k d_{k-1} \tag{10}$$

where $j = 1, ..., N$ and $k = 2, ..., N$. By putting 8 and 10 into 9, we have the recursive relationship to find d_j:

$$d_j = \frac{a_j d_{j-1} - b_j c_{j-1}}{d_{j-1}} \tag{11}$$

Then we create 2x2 matrices in order to process the parallelisation recursively (for $j = 1, ..., N$):

$$R_0 = \begin{bmatrix} a_0 & 0 \\ 1 & 0 \end{bmatrix} \tag{12}$$

and

$$R_j = \begin{bmatrix} a_j & -b_j c_{j-1} \\ 1 & 0 \end{bmatrix} \tag{13}$$

and by setting $S_j = R_j R_{j-1} ... R_0$, we have

$$d_j = \frac{\begin{pmatrix}1\\0\end{pmatrix}^t S_j \begin{pmatrix}1\\1\end{pmatrix}}{\begin{pmatrix}0\\1\end{pmatrix}^t S_j \begin{pmatrix}1\\1\end{pmatrix}} \tag{14}$$

The algorithm is the following:

1. On each processor, matrix R_k is created. k is the number of rows which have to be treated by the process.
2. On each processor, matrix S_j is created.
3. Calculate local d_k.
4. Send local d_k to next processor.
5. Each process calculate the local l_k
6. Distribute d_j and l_j values on all processors.
7. On each processor, a local forward and backward substitution are performed to obtain the solution

This algorithm is performed on each time step, and the found value is inserted in the result matrix. Previous time vector is used for actual time space.

2.2.2 Crank-Nicholson

We have :

$$\frac{T_j^{n+1} - T_j^n}{\Delta t} = D\frac{1}{2}\left(\frac{T_{j+1}^{n+1} - 2T_j^{n+1} + T_{j-1}^{n+1}}{\Delta x^2} + \frac{T_{j+1}^n - 2T_j^n + T_{j-1}^n}{\Delta x^2}\right) \tag{15}$$

Written in a matrical way, the solution at space step $n+1$ is given by :

$$\begin{aligned}
&\begin{bmatrix} 1+C & -C/2 & 0 & \cdots & 0 \\ -C/2 & 1+C & -C/2 & \ddots & \vdots \\ 0 & -C/2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1+C & -C/2 \\ 0 & \cdots & 0 & -C/2 & 1+C \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_i \\ \vdots \\ T_{ntime} \end{bmatrix}_{n+1} \\
&= \begin{bmatrix} 1-C & C/2 & 0 & \cdots & 0 \\ C/2 & 1-C & C/2 & \ddots & \vdots \\ 0 & C/2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1-C & C/2 \\ 0 & \cdots & 0 & C/2 & 1-C \end{bmatrix} \begin{bmatrix} T_1 + CT_0 \\ T_2 \\ \vdots \\ T_i \\ \vdots \\ T_{ntime} + CT_f \end{bmatrix}_n
\end{aligned} \tag{16}$$

with

$$C = \frac{D\Delta t}{\Delta x^2}$$

The method and the algorithm used are the same as Laasonen scheme, except that the vector which is found with Thomas algorithm is multiplied by the right-term matrix before the next time step.

2.3 Analytical solution

The analytical solution of the problem considered will be used to compare our results and to calculate the errors. The result of the analytical solution for a time t and a position x is given by :

$$T = T_{ext} + 2(T_{int} - T_{ext}) \sum_{m=1}^{m=\infty} e^{-D(m\pi/L)^2 t} \frac{1-(-1)^m}{m\pi} sin(\frac{m\pi x}{L}) \qquad (17)$$

2.4 Speed-up calculation

To investigate the performance of our programs, we used MPI functions :

- MPI_Wtime : returns an elapsed time on the calling processor.[7]
- MPI_Wtick : returns the resolution of MPI_Wtime.[7]

The actual speed-up is calculated as following:

$$speedup = finaltime - starttime \qquad (18)$$

where $finaltime$ is a double containing the value of MPI_Wtime at the end of the program, and $starttime$ is a double containing the value of MPI_Wtime at the beginning of the program.

3 Results

3.1 Step size effect

We must note that due to the step sizes, first calculations could not be done in parallel.

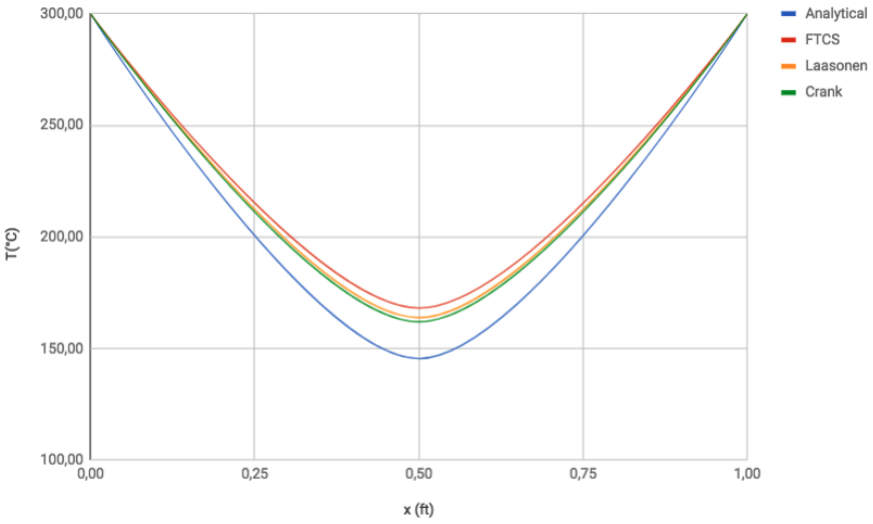

Figure 4: Results with differents serial schemes at $t = 0,5h$; $\Delta x = 0,5$; $\Delta t = 0,1$

We can see quite a big difference between the analytical solution and the other schemes. The errors had been calculated to evaluate the accuracy of each method.

Table 1: Errors for different serial schemes at $t = 0,5h$; $\Delta x = 0,5$; $\Delta t = 0,1$

FTCS	Laasonen	Crank-Nicholson
3,04%	2,46%	2,21%

We observe that for these step sizes, at t = 0,5h, Crank-Nicholson method is the most accurate, and Forward Time Central Space is the less accurate. Let see the differences when we decrease the step sizes.

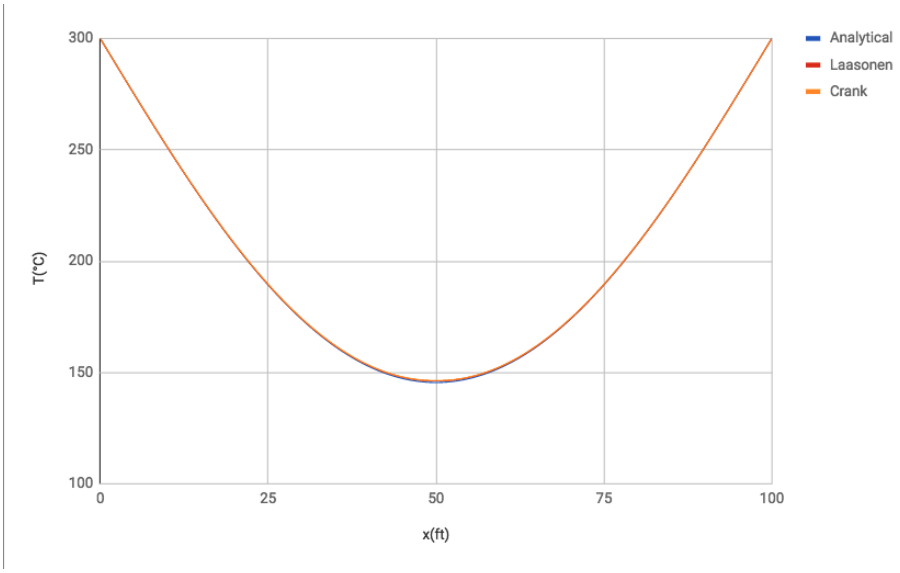

Figure 5: Results with differents parallelized schemes at $t = 0,5h$; $\Delta x = 0,005$; $\Delta t = 0,001$

With these time and space steps, the stability criterion for FCTS method was not verified. The values are not displayed because of their nlack of interest. We can see a very small difference between the analytical solution and the other schemes. The errors had been calculated to evaluate the accuracy of each method.

Table 2: Errors for different parallelized schemes at $t = 0,5h$; $\Delta x = 0,005$; $\Delta t = 0,001$

	Laasonen	Crank-Nicholson
Serial	0,01%	0,04%
Parallel	0,10%	0,15%

We observe that by reducing a lot the step size, accuracy of the results is very good. We also observe that serial results are quite the same as the analytical results. We will discuss this later.

3.2 Speed-up investigation

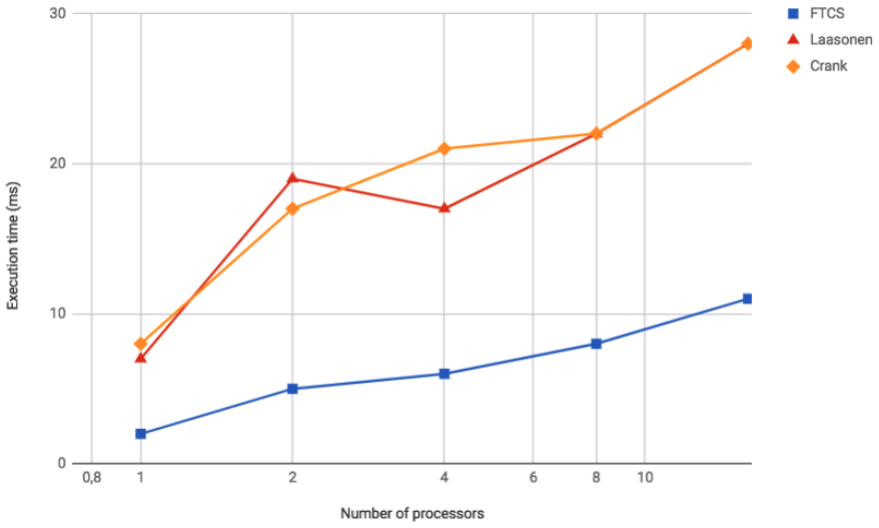

Figure 6: Speed-up results for each method

As we see in Fig. 6, the speed-up time increase as we increase the number of processors.

4 Discussion

4.1 Analytical solution

First thing that should be considered is that the analytical solution should not be considered as the perfect exact solution, because of several factors.
First, the analytical solution is calculated by the computer, so there may be some numerical error.
Secondly, we have to consider the calculation of the sum inside equation (17). Indeed, as m should be infinite, in theory, the biggest it is is the most accurate. However, increasing computational calculation may alterate the results. I choosed a compromise where values of the analytical solution at $t = 0$ were as close to initial conditions as possible, with an m not too big, to reduce run time and numerical error.

4.2 Step size study

With a big step size, we can see that serial results are quite accurate (less than 5%). However, by reducing the time step and increasing the size of the grid, we observe better accuracy (less than 1%) for serial results.
So we can say that by increasing the size of the grid, we improve the results of the serial programs.

However, we observe bigger errors for parallel programs than for serial programs, for a small step size. Except for the Forward Space Central Time method [2], we have a good stability.
So we can say that in this case, parallelizing complex programs leads to a good stability, but still with a bigger error than in the same program in serial.
That can be due to the fact that our problem is not complex enough, and putting it into parallel causes some more errors than in the serial program, which manages the problem well enough.

4.3 Performance study

We have observed that the parallelization of our serial programs was leading to a loss of runtime speed. But the purpose of parallel programs is to reduce the speed-up time.[3] One possible cause of this loss of speed could be, as seen before, the fact that our problem is not complex enough to be put in parallel, and the communication time induced by the parallelization decreases the efficiency of our serial algorithms.

5 Future work

Our future work would be based on three major axes : memory, runtime and interface optimization. It would be interesting to take a smaller time step, and to try to test our program with a parallel programming model, and run it on an HPC, as we have the opportunity to work on one here at Cranfield University.

5.1 Memory optimization

As said previously, memory use could be improved by at least two manners :

- Calculate only the values which will be used in the study
- For implicit schemes, replace the tridiagonal matrix by one vector

5.2 Runtime optimization

As said before, since we have a small study scale, it is not crucial to have a good runtime optimization. However, we still could optimize the runtime, particularly by using Thomas algorithm instead of LU algorithm.

5.3 Interface optimization

We also could improve the way the results are printed : here we write them brutally in our main function, or in a CSV file, which is used in a spreadsheet application in order to print graphics.
Maybe a little interface more user-friendly could be developped, using QT or something else, with the possibility for the user to input data, initial conditions, the chosen scheme, and with the printing of graphics and errors value.

6 Conclusion

For a one dimensional problem, PDE can be resolved using discretisation schemes. In this study we saw that several schemes can fit for this kind of problem. However, some schemes appears to be more accurate.
The accuracy of all the schemes seems to be impacted by common causes:

- Error grows as time step grows : the smallest time step must be used, while complying with the computational ressources that are at our disposal (indeed, numerical computational error could appear)
- Error tends to decrease as the time study grows

However, there is some factors which gives each scheme particular accuracy.

Resolving PDE with computational methods also introduce numerical errors, due to the computers way of calculating. Depending on the power of the computer we use, we must try to be as precise as possible by setting a study mesh with a grid as small as possible. Some other study with different computational ressource could teach us about the effect of a very small grid, i.e. the effect of a lot more calculations.

7 References

References

[1] Wazwaz, Abdul-Majid. *Partial differential equations*. CRC Press, 2002.

[2] Luciano Rezzolla *Numerical Methods for the Solution of Partial Differential Equations*, 2011

[3] Gottlieb, Allan; Almasi, George S. (1989). *Highly parallel computing*. Redwood City, Calif.: Benjamin/Cummings.

[4] D. Britz & Jorg Strutwolf, *Digital Simulation in Electrochemistry*. Springer, 2006

[5] George Karniadakis *Parallel Scientific Computing in C++ and MPI*. 1994.

[6] E.F. Van de Velde. *Concurrent Scientific Computing*. 1994.

[7] Open MPI v2.0.4 documentation.
`https://www.open-mpi.org/doc/v2.0/`