HIGH PERFORMANCE COMPUTING

Heat Conduction Equation

January 24, 2018

António Pedro Araújo Fraga

Student ID: 279654

Cranfield University

M.Sc. in Software Engineering for Technical Computing

Contents

Abstract

Three numerical schemes were applied to compute a solution for a parabolic partial differential equation, the heat conduction equation. It was used an explicit scheme and two implicit schemes. The solutions were computed both sequentially and in parallel making use of **MPI**, **M**essage **P**assing **I**nterface technology. Two pairs of **time** and **space** steps were studied, along with their effect on the solving system. **Speed-ups** of solutions computed with several processes were analysed, and their values were discussed.

Table 1: Nomenclature

Diffusivity	D
First derivative in time	$\frac{\partial f}{\partial t}$
First derivative in space	$\frac{\partial f}{\partial x}$
Time grid position	n
Space grid position	i
Function at time and space grid position	f_i^n
Time step	Δt
Space step	Δx
Time value	t
Space value	x
Analytical function at specific space and time values	$f(x,t)$
Initial Temperature	T_{in}
Surface Temperature	T_{sur}

Introduction

Numerical methods are used to obtain an approximated solution for a given problem. The exact solution of those problems are usually computed in **Nondeterministic Polynomial Time**, therefore an approximated solution is often accepted. The approximation factor is often related with the number of space steps that a time step is split on. A high number of space steps leads to a more accurate solution, keeping in mind the existence of **round-off** errors [1].

Although, decreasing the value of space steps results in a more "computational hungry" process. Which means that the time used to compute a solution increases, unless several processes are used to compute the final solution. Parallel computing is simple if it's done with **independent** data. In this case, there's no need to exchange data between workers. Sometimes the methods used to compute the solution have dependencies between space steps calculations. Thus, unless a process communicates with different processes that are computing dependable data, it's not possible to compute an approximated solution.

The challenge is to minimize the communication overhead. To achieve that, one has to

keep in mind how to perform communications. Deciding **when** and **how** to **exchange** data between processes.

Memory management plays an important role on **efficiency** as well. A program is split into **segments**, and each segment is split into **pages**. The size of each segment depends of the compiler. Each of them contains important data like the **stack**, **heap**, and actual data. But each **page** is divided into equal parts, **4kb** on UNIX based systems. Therefore, whenever a program accesses an address, a page is loaded into **cache**, which is a very fast type of memory close to the **CPU**. This means that accessing memory within the same page won't add the overhead of moving the data from the main memory to cache again. Such knowledge might make one think of how to write specific parts of the code.

Problem definition

The problem is defined in the previously developed **report**[1] for **Computational Methods** & **C++** modules. It is intended to examine the application of distributed memory parallel programming techniques on the referred problem.

Analysis

The analysis of **parallel programming** can be done by measuring times of execution. One should measure the time of execution with a single process, comparing it with the execution time in parallel with **p** processors. Therefore, the **Speedup** concept can be defined[2],

$$Speedup(P) = \frac{Time_{parallel}(P)}{Time_{Sequential}}$$

It is important to define the concept of **Theoretical Speedup** as well. Whenever a program is computed in parallel, the optimal speedup is obtained by taking the number of processes used to compute a solution. Often, the **Optimal Speedup** is difficult to be achieved. This happens because the solution is computed in a distributed system. Processes, by default, are not able to access data from different processes easily. One must create a form of **IPC**, or **I**nter **P**rocess **C**ommunication. The **MPI** technology offers an **API** to achieve that.

Procedures

Three different schemes/methods were used to compute a solution for the given problem, one of them is an explicit schemes, **Richardson**, **DuFort-Frankel**, and two of them are implicit schemes, **Laasonen Simple Implicit** and **Crank-Nicholson**. The space step was maintained at **0.05** *ft*, and the time step took the value of **0.01** *h*, studying every solutions in intervals of **0.1** hours from **0.0** to **0.5**. The **Laasonen Simple Implicit** solution was also studied with different time steps, always maintaining the same space step, $\Delta x = 0.05$:

- $\Delta t = 0.01$
- $\Delta t = 0.025$
- $\Delta t = 0.05$
- $\Delta t = 0.1$

As referred, considering the initial equation, these methods can be written in its discretized form.

Explicit Schemes

This type of schemes rely only on the previous time steps to calculate the current time step solution. In the case of both used methods, they were relying in known values of the **n - 1** and **n** time steps to calculate a value for the **n + 1** time step. Thereby, the second time step can not be calculated by these methods, because there's no possible value for a negative time step. A different method, for the same equation, with two levels of time steps was used in order to overcome this situation, the **Forward in Time and Central in Space** scheme. It's known that this method is **conditionally stable**, and its stability condition is given by[**?**],

$$\frac{D\Delta t}{(\Delta x)^2} \leq 0.5$$

Therefore, considering $\Delta t = 0.01$, $\Delta x = 0.05$, and $D = 0.1$, this method is declared stable. It's important to have a stable solution for the first iteration, since it is a major influence on the overall solution[**?**]. Therefore, this iteration could be calculated with the following expression,

$$f_i^{n+1} = f_i^n + \frac{D\Delta t}{(\Delta x)^2}(f_{i+1}^n - 2f_i^n + f_{i-1}^n)$$

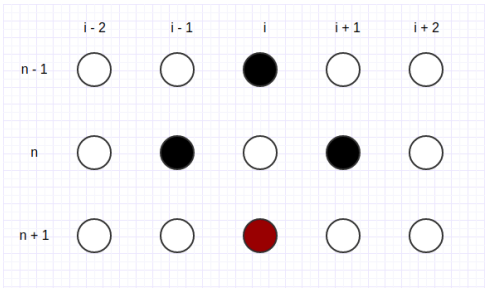

Figure 1: DuFort-Frankel's method stencil.

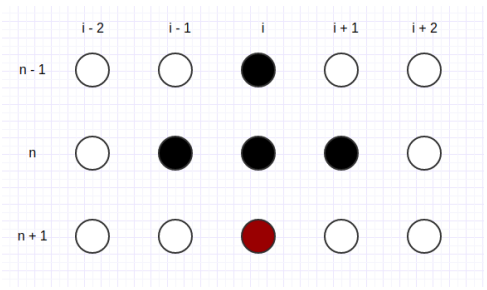

Figure 2: Richardson's method stencil.

Richardson

The Richardson method can be applied by having a central in time and central in space scheme. Regarding to stability issues, this method is unconditionally unstable. This method is of order $\mathbf{O}(\Delta x^2, \Delta t^2)$ [**?**]. Following the heat conduction equation, the expression could be represented at **Figure 2** and could be written as following:

$$\frac{f_i^{n+1} - f_i^{n-1}}{2\Delta t} = D\frac{f_{i+1}^n - 2f_i^n + f_{i-1}^n}{(\Delta x)^2}$$

Which corresponds to,

$$f_i^{n+1} = f_i^{n-1} - \frac{2\Delta t D}{(\Delta x)^2}(f_{i+1}^n - 2f_i^n + f_{i-1}^n)$$

DuFort-Frankel

The DuFort-Frankel scheme can be applied by having central differences in both derivatives, but to prevent stability issues, the space derivative term f_i^n can be written as the average value of f_i^{n+1} and f_i^{n-1}. The method stencil can be observed at **Figure 1**. Therefore this method is of order $\mathbf{O}(\Delta x^2, \Delta t^2, (\frac{\Delta t}{\Delta x})^2)$ [**?**], it is declared as unconditionally stable and it may be formulated as follows:

$$\frac{f_i^{n+1} - f_i^{n-1}}{2\Delta t} = D\frac{f_{i+1}^n - f_i^{n+1} - f_i^{n-1} + f_{i-1}^n}{(\Delta x)^2}$$

Which is equivalent to,

$$f_i^{n+1} = f_i^{n-1} - \frac{2\Delta t D}{(\Delta x)^2}(f_{i+1}^n - f_i^{n+1} - f_i^{n-1} + f_{i-1}^n)$$

Implicit Schemes

In other hand, implicit schemes rely not only on lower time steps to calculate a solution, but also on the current time step known values. Each time step solution can often be solved by applying the Thomas Algorithm, which is an algorithm that can solve tridiagonal matrix systems, $Ax = r$[**?**]. This algorithm is a special case of the LU decomposition, with a better performance. The matrix A can be decomposed in a lower triangular matrix L and an upper triangular matrix U, therefore $A = LU$[**?**]. This algorithm consists of two steps, the downwards phase where the equation $Lp = r$ is solved and the upwards phase, solving $Ux = p$[**?**], obtaining a solution for x.

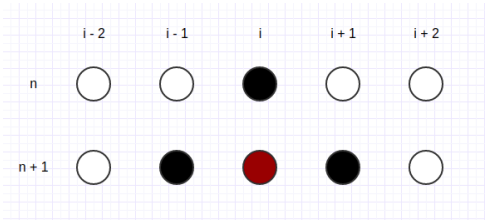

Figure 3: Laasonen's method stencil.

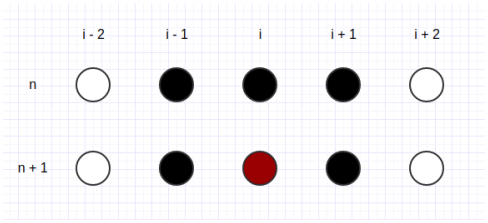

Figure 4: Crank-Nicholson's method stencil.

Laasonen Simple Implicit

The time derivative is considered forward in time. Central difference is used in space derivative, and the scheme is of order $\mathbf{O}(\Delta x, \Delta t^2)$[**?**], and unconditionally stable. Concluding, the below equation could be established, and it could be represented at **Figure 3**:

$$\frac{f_i^{n+1} - f_i^n}{\Delta t} = D\frac{f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1}}{(\Delta x)^2}$$

Assuming that $c = \frac{\Delta t D}{(\Delta x)^2}$, the equation could be represented as:

$$(1-2c)f_i^{n+1} = f_i^n + c\left[f_{i+1}^{n+1} + f_{i-1}^{n+1}\right]$$

The values of the first and last space position of each time step are known, they are represent by the T_{sur} value. Therefore, in every second and penultimate space step, two terms of the previous equation could be successfully inquired. For the second space step, the equation could be divided by having the unknown terms in the left side and the known terms in the right side:

$$(1-2c)f_i^{n+1} - cf_{i+1}^{n+1} = f_i^n + cf_{i-1}^{n+1}$$

And the same could be done for the penultimate space step:

$$(1-2c)f_i^{n+1} - cf_{i-1}^{n+1} = f_i^n + cf_{i+1}^{n+1}$$

For every other space steps with unknown values, the expression could be generalized as:

$$(1-2c)f_i^{n+1} - c\left[f_{i+1}^{n+1} + f_{i-1}^{n+1}\right] = f_i^n$$

Considering that the maximum number of space steps is **m**, the previous expressions could form a system of linear equations, $A.x = r$:

$$\begin{bmatrix} (1-2c) & -c & 0 & 0 & \dots & 0 & 0 \\ -c & (1-2c) & -c & 0 & \dots & 0 & 0 \\ 0 & -c & (1-2c) & -c & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & -c & (1-2c) \end{bmatrix} \begin{bmatrix} f_1^{n+1} \\ f_2^{n+1} \\ f_3^{n+1} \\ \dots \\ f_{\mathbf{m}-1}^{n+1} \end{bmatrix} = \begin{bmatrix} f_1^n + cf_0^{n+1} \\ f_2^n \\ f_3^n \\ \dots \\ f_{\mathbf{m}-1}^n + cf_{\mathbf{m}}^{n+1} \end{bmatrix}$$

Crank-Nicholson

The time derivative is considered forward in time, and the space derivative can be replaced by the average of central differences in time steps **n** and **n + 1**. The method is of order **O**(Δx^2**,** Δt^2**)** [**?**], is declared unconditionally stable and it could be represented at **Figure 4**. Thus:

$$\frac{f_i^{n+1} - f_i^n}{\Delta t} = \frac{1}{2}D\left[\frac{f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1}}{(\Delta x)^2} + \frac{f_{i+1}^n - 2f_i^n + f_{i-1}^n}{(\Delta x)^2}\right]$$

In this method, the coefficient had a new value, $c = \frac{1}{2}\frac{\Delta tD}{(\Delta x)^2}$, and assuming that $p = f_{i+1}^n + f_{i-1}^n$, the equation could be written as follows,

$$(1-2c)f_i^{n+1} = (1-2c)f_i^n + c\left[f_{i+1}^{n+1} + f_{i-1}^{n+1} + p\right]$$

Following the same logical principles of the previous scheme, some expressions could be generalized for the second,

$$(1-2c)f_i^{n+1} - cf_{i+1}^{n+1} = (1-2c)f_i^n + c\left[f_{i-1}^{n+1} + p\right]$$

, penultimate,

$$(1-2c)f_i^{n+1} - cf_{i+1}^{n+1} = (1-2c)f_i^n + c\left[f_{i+1}^{n+1} + p\right]$$

, and every other space steps with unknown values.

$$(1-2c)f_i^{n+1} - c\left[f_{i+1}^{n+1} + f_{i-1}^{n+1}\right] = (1-2c)f_i^n + cp$$

Thus, a tridiagonal matrix system is obtained,

$$\begin{bmatrix} (1-2c) & -c & 0 & 0 & \dots & 0 & 0 \\ -c & (1-2c) & -c & 0 & \dots & 0 & 0 \\ 0 & -c & (1-2c) & -c & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & -c & (1-2c) \end{bmatrix} \begin{bmatrix} f_1^{n+1} \\ f_2^{n+1} \\ f_3^{n+1} \\ \dots \\ f_{\mathbf{m}-1}^{n+1} \end{bmatrix} = \begin{bmatrix} (1-2c)f_1^n + c\left[f_0^{n+1} + p\right] \\ (1-2c)f_2^n + cp \\ (1-2c)f_3^n + cp \\ \dots \\ (1-2c)f_{\mathbf{m}-1}^n + c\left[f_{\mathbf{m}}^{n+1} + p\right] \end{bmatrix}$$

Solution Design

The code was first planned with an initial structure and suffered incremental upgrades. A **method** class was created, being a prototype with multiple inheritance, containing three sub classes: **Analytical**, **Implicit** and **Explicit**. Therefore, the **Implicit** class is an Abstract class as well. This class has three sub classes, representing the three explicit methods used in this problem. Similarly, the **Implicit** class is also an abstract class, having two implicit methods classes as sub classes. The previously described inheritance structure can be more easily visualized on **Figure 5**.

A Method class contains a **Problem** object. The **Problem** class represents the Heat Conduction problem, containing informations about the time and space steps, the solution and initial conditions.

An **Input and Output Manager** class was developed so that the code related with plots and tables exportations could be separated from the logical source code. This class was developed with several methods regarding data interpretation and structuration in order to easily export plot charts. A **gnuplot c++ library** was used, therefore the gnuplot syntax could be directly used from the c++ code, cutting down the need of developing external bash scripts for this specific purpose.

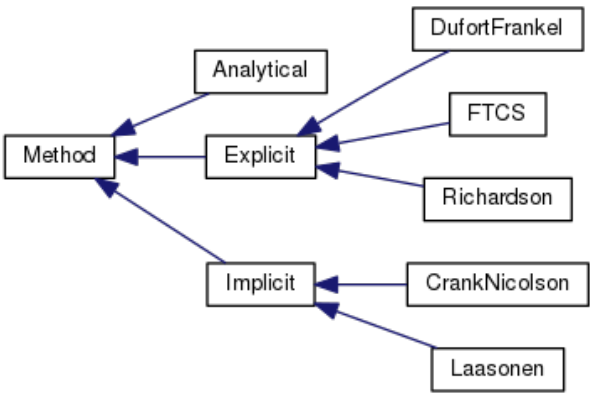

Figure 5: Method Class inheritance diagram.

Despite the referred classes, a header file with useful **macros** was declared. This file contains information about which conditions to test, like the initial temperature and the surface temperature. Therefore, if for some reason, one of this values changes, it can be easily corrected.

The **Matrix** and **Vector** classes, which were provided in the c++ module were reused to represent a solution matrix or a solution vector of a certain iteration.

The several objects in this structure could be instantiated in the main file, calling methods to compute the several solutions and to export their plot charts. The previously described classes can be represented in the **Figure 6** diagram.

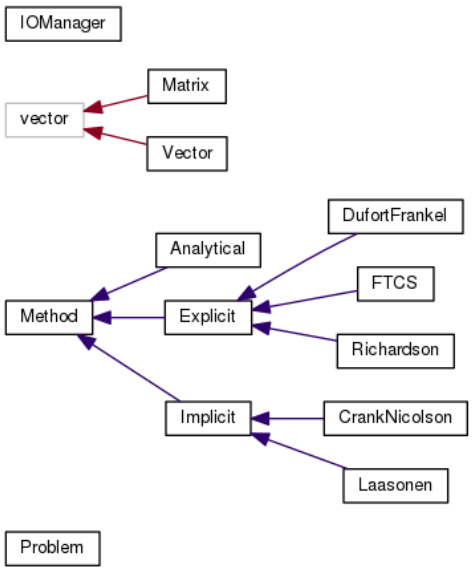

Figure 6: Class Diagram.

Results & Discussion

The results of the four methods, **Richardson**, **DuFort-Frankel**, **Laasonen Simple Implicit** and **Crank-Nicholson** can be seen in the following figures/tables. These results were used to analyze each solution quantitatively and qualitatively. In most of the plot charts, the obtained solution was compared to the analytical solution so that it would be possible to realize whether the solution was a good approximation or not. Notice that the next results are regarding to the "default" values of time and space steps, $\Delta t = 0.01$ and $\Delta x = 0.05$.

Table 2: Richardson method error table.

t \ x	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
0.1	1.05136e+06	631856	123707	8417.73	300.81	8417.73	123707	631856	1.05136e+06
0.2	1.33245e+11	1.39e+11	7.02854e+10	2.06123e+10	6.93136e+09	2.06123e+10	7.02854e+10	1.39e+11	1.33245e+11
0.3	2.14659e+16	2.74969e+16	1.97012e+16	9.88337e+15	5.98161e+15	9.88337e+15	1.97012e+16	2.74969e+16	2.14659e+16
0.4	3.91917e+21	5.60267e+21	4.87086e+21	3.31281e+21	2.58429e+21	3.31281e+21	4.87086e+21	5.60267e+21	3.91917e+21
0.5	7.74272e+26	1.19047e+27	1.18021e+27	9.72231e+26	8.60626e+26	9.72231e+26	1.18021e+27	1.19047e+27	7.74272e+26

By examining **Table 2**, it could be concluded that the solution given by the Richardson method was considerably different from the analytical solution. This was due to the fact that this method is declared as **unconditionally unstable**. As referred before, when a method is declared unstable, the error grows as the time advances. The error growth was responsible for obtaining a different solution, or a solution to a different problem. The mathematical calculations regarding the stability and accuracy properties of this method can be found under the appendix section.

When looking at **Figure 7**, it can be observed that the DuFort-Frankel solution is quite approximated to the real solution. This scheme, as it could be observed at **Figure 11**, is more time efficient comparing to the implicit unconditionally stable methods, the only disadvantage is the fact that it requires a different method for the first iteration.

Similarly of what could be concluded on DuFort-Frankel results, by observing **Figure 9** and **Figure 8**, it can also be deducted that these are good solutions. These schemes, Crank-Nicholson and Laasonen, are unconditionally stable as well. Therefore good results were expected.

In other hand, when a quantitative analysis was done, it could be seen that the Crank

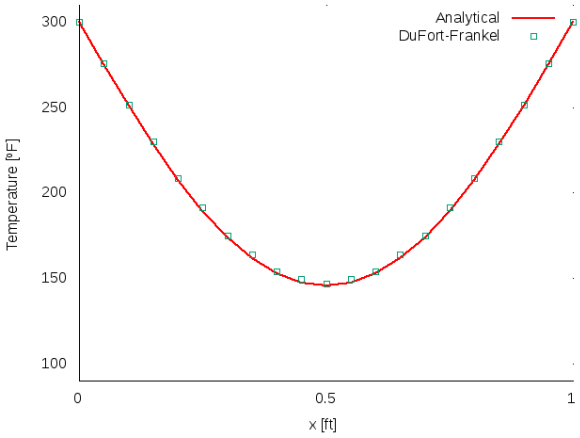

Figure 7: DuFort-Frankel's solution at $t = 0.5$.

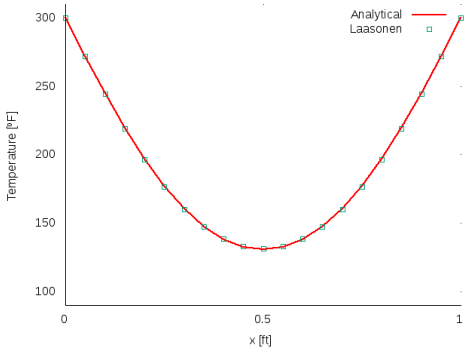

Figure 8: Laasonen's solution at $t = 0.4$.

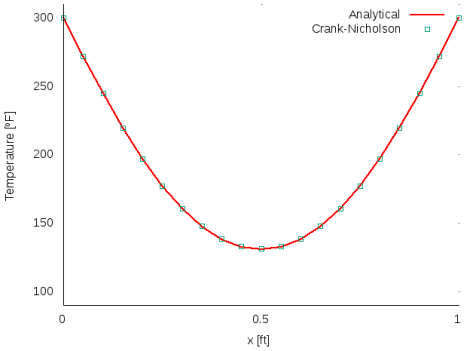

Figure 9: Crank-Nicholson's solution at $t = 0.4$.

Nicholson scheme is more accurate than the Laasonen and DuFort-Frankel methods. By looking at **Figure 10**, it can be observed that the second norm value of the **Error matrix** of this scheme is smaller than the values obtained by the other methods **Error Matrices**.

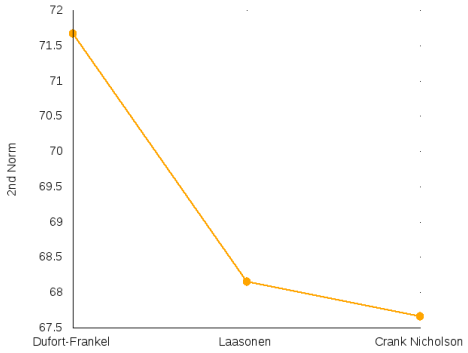

Figure 10: 2nd norm values of Error Matrices.

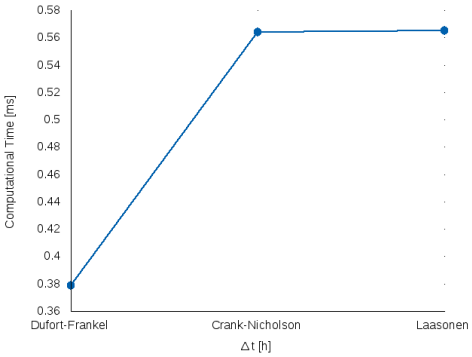

Figure 11: Computational times of stable methods.

Laasonen Implicit Scheme: study of time step variation

Laasonen Implicit Scheme is an unconditionally stable scheme to solve Parabolic Partial Differential Equations. Therefore, with the right time and space step, there's almost no error related to the development of its results throughout the time advancement.

A reduction on these steps led to a higher computational time, since there's more calculations to be made. Whereas steps with higher values led to more inaccurate results[**?**]. This phenomenon could be explained with a concept that was introduced earlier, the **truncation error**[**?**]. This error can only be avoided with exact calculations, but can be reduced by applying a larger number of smaller intervals or steps. As referred before, different results of this method were studied by changing the time step size. The space step was maintained, $\Delta x = 0.05$.

Table 3: Laasonen method error table for the several Δt at $t = 0.5$

x / Δt	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
0.01	0.288694	0.385764	0.255427	0.0405061	-0.0611721	0.0405061	0.255427	0.385764	0.288694
0.025	0.738044	1.0344	0.805442	0.368491	0.157551	0.368491	0.805442	1.0344	0.738044
0.05	1.53627	2.15669	1.71375	0.864487	0.457364	0.864487	1.71375	2.15669	1.53627
0.1	3.29955	4.49523	3.46045	1.7082	0.898726	1.7082	3.46045	4.49523	3.29955

Table 3 and **figure 12** could support the previous affirmations. While observing **table 3**, it could be seen that the error is larger for bigger time steps, as it was expected. Whereas when observing **figure 12**, it can be identified a reduction in computational time as the **time step** becomes larger.

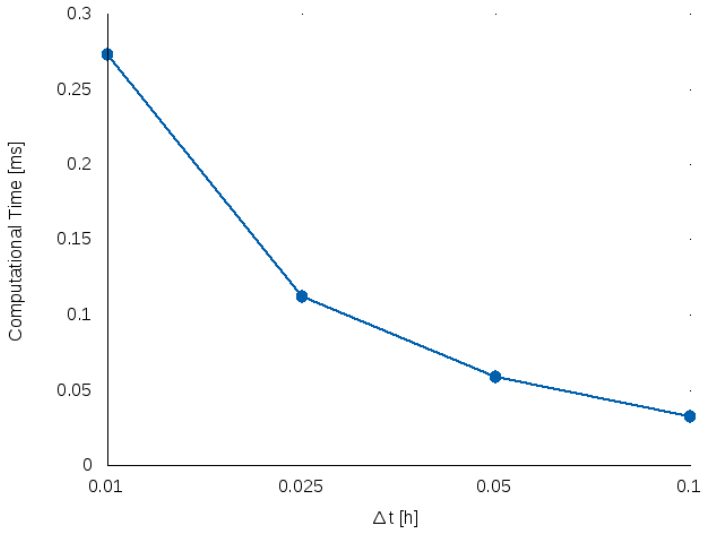

Figure 12: Laasonen method computational times for the several Δt.

Conclusions

The obtained results could support the theoretical concepts. Unstable methods demonstrated an error growth through the time progress. The **Forward in Time, Central in Space** explicit scheme was stable with the given initial conditions, therefore it could support a good solution for the explicit stable scheme, **DuFort-Frankel**. As referred, the solution of the DuFort-Frankel method strongly depends on the first iteration solution.

It could be observed that smaller steps can lead to a time expensive solution, whereas larger steps lead to an error increase. Stable methods could give a good solution with the right time and space steps, but by analysing the second norm value, it was concluded that the Crank-Nicholson method is more accurate. This is due to the fact that this method has a better approximation order.

It is important to have a balance between the two problems (time and approximation), a method should be computed in an acceptable time, and still obtain a good result. In realistic scenarios the problem solution is not known, therefore error estimates are impractical. The used step size should be small as possible, as long as the solution is not dominated with round-off errors. The solution must be obtained with a number of steps that one has time to compute.

References

[1] António Pedro Fraga, December 2017, *Heat Conduction Equation, C++ & Computational Methods* Available at: <http://pedrofraga.me/heat-conduction-equation.pdf> [Accessed 24 January 2018]

[2] Multiple authors, May 2017, *A Comprehensive Linear Speedup Analysis for Asynchronous Stochastic Parallel Optimization from Zeroth-Order to First-Order* Available at: <http://xrlian.com/res/Asyn_Comprehensive/paper.pdf> [Accessed 24 January 2018]

Appendices