Parallel Implementation of Numerical Algorithms for PDE's*

FELIPE BOTTEGA DINIZ† CARLOS A. DE MOURA‡

Abstract

Our goal is to illustrate, through a quite simple example, the use of parallel computing for evolution linear partial differential equations. We consider the constant coefficient one-dimensional heat conduction model for a homogeneous bar. Approximations to the solution for the mixed initial and boundary-value problem are computed with sequential and parallel implementations of the same algorithm. The obtained results are then compared.

Resumo

Nosso objetivo é ilustrar, por meio de um exemplo, a utilização da computação em paralelo para equações diferenciais parciais lineares de evolução. Consideramos um modelo linear para a condução de calor sobre uma barra unidimensional de material homogêneo. Aproximações para a solução do problema misto associado – condições iniciais e de contorno – são obtidas com implementações sequencial e em paralelo do mesmo algoritmo. Os resultados são comparados.

1 Introduction

Nowadays in nearly all science and technology areas, to treat data and analyze phenomena described by a mathematical model, the use of computers is a strong need. The most detailed the models are, more computational workload is required.

Numerical calculations performance is strongly linked to the arithmetics processors. Despite being able to perform calculations quickly, they are bounded by physical constraints. And these constraints lead to bounds for the calculation

* *Palavras chave:* Computação Científica, EDP, Análise Numérica, Algoritmo de Diferenças Finitas, Processamento em Paralelo; Projeto de Iniciação Científica de 2011, apresentado resumidamente como poster nos eventos *UERJ sem Muros* e *X Semana do IME*

† Aluno de Iniciação Científica, Bolsista do CNPq, Departamento de Matemática Aplicada, IME/UERJ,
felipebottega@gmail.com

‡ Departamento de Matemática Aplicada, IME/UERJ, demoura@ime.uerj.br

speed. This also represents a limitation for the models, as it makes no sense to create very high precision models, as long as they are bound to a slow computer treatment. In the late 80's, the work reported in [4] showed the viability of a different approach to handle this difficulty – *Parallel Computing*. This can be described in a quite simple way: just make use of several processors working together, sharing the whole task.

2 The physical problem

Consider heat conduction in a one-dimensional homogeneous bar. Suppose that the bar two extreme points are kept at 0^o C temperature (boundary conditions) and that the bar lays on the x axis, in such a way that one of its end points remains at the origin. Denote by x the coordinate of a (material) point of the bar and further assume that the bar length is L. The temperature of each bar point at a given instant is also known, these values being called the initial condition. To model the temperature evolution we take the equation

$$\alpha u_{xx} = u_t \, , \tag{1}$$

where t denotes time and $u(x,t)$ is the temperature at instant t of the point on position x. The figure below shows the details.

Let the initial condition be

$$u(x,0) = f(x), \qquad 0 < x < L \, ,$$

while the boundary conditions were assumed to be

$$u(0,t) = 0, \qquad t > 0 \, ,$$

$$u(L,t) = 0, \qquad t > 0 \, .$$

Thus the problem to be solved is the following:

Which is the state of the bar temperature[1] *at any fixed instant* $T > 0$*?*

[1] By the state of the bar temperature at time T we mean the temperature value for each bar point at the instant T.

3 The approximation algorithm

First, we split the bar in r equal parts, each with size equal to Δx. We also split the time interval from 0 to T in s equal parts of size Δt.
By making use of the Taylor series development for the function $u(x,t)$ we reach the two following identities:

$$u_t = \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} + O(\Delta t) \ ,$$

$$u_{xx} = \frac{u(x-\Delta x,t) - 2u(x,t) + u(x+\Delta x,t)}{\Delta x^2} + O(\Delta x^2) \ .$$

By replacing these identities in (1) we obtain

$$\alpha\frac{u(x-\Delta x,t) - 2u(x,t) + u(x+\Delta x,t)}{\Delta x^2} + O(\Delta x^2) = \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} + O(\Delta t) \ .$$

Denote $j \cdot \Delta x = x_j$ and $n \cdot \Delta t = t_n$, so that $[x_{j-1}, x_j]$, for $(j = 1 \ldots r)$, is the jth piece of the bar and $[t_{n-1}, t_n]$, for $(n = 0 \ldots s)$, is the nth temporal step.

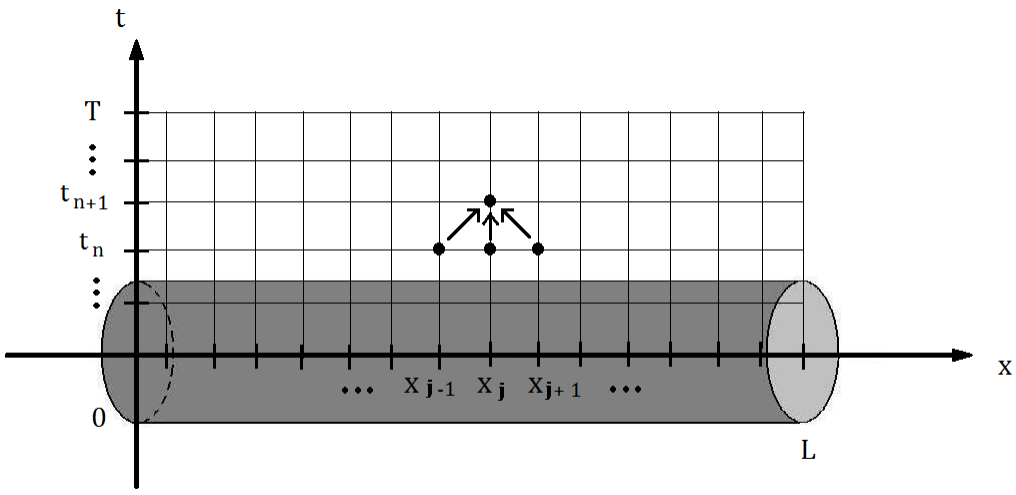
Now let $\lambda = \alpha \cdot \Delta t/\Delta x^2$, so that the above identity may be rewritten as:

$$u(x_j, t_{n+1}) = \lambda\big(u(x_{j-1}, t_n) - 2u(x_j, t_n) + u(x_{j+1}, t_n)\big) + \epsilon \ ,$$

where $\epsilon = O(\Delta x^2) + O(\Delta t)$ is the error associated to the replacement of the above derivatives by the corresponding quotient differences. If the discretizing parameters Δx and Δt are taken small enough, hopefully this error is small. The above mentioned replacement corresponds to neglecting the error, and with this procedure we no longer will calculate the exact values for the solution $u(x_j, t_n)$ but an approximation to it, which will be denoted $U(x_j, t_n)$. To simplify the notation, make $U(x_j, t_n) = U_j^n$. This way we are led to

$$U_j^{n+1} = \lambda U_{j-1}^n + (1 - 2\lambda)U_j^n + \lambda U_{j+1}^n \ .$$

This relation allows us to calculate the bar temperature in x_j at the $(n+1)$th temporal step through three other bar points temperature values at the nth temporal step. Thus a recursive algorithm was defined. The figure below illustrates the process.



To find out the bar state at the $(n+1)$th temporal step we need to calculate the values of

$$U_0^{n+1},\ U_1^{n+1},\ldots,U_r^{n+1}.$$

As the values of U_0^{n+1} and U_r^{n+1} are known (boundary conditions), it suffices to calculate the values of $U_1^{n+1},\ U_2^{n+1},\ldots,U_{r-1}^{n+1}$. Thus, we have a system of $r-1$ equations to compute. This system can be written in the form of a matrix equation, as shown below.

$$\begin{pmatrix} U_1^{n+1} \\ \\ U_2^{n+1} \\ \vdots \\ \vdots \\ U_{r-2}^{n+1} \\ \\ U_{r-1}^{n+1} \end{pmatrix} = \begin{pmatrix} 1-2\lambda & \lambda & 0 & 0 & 0 & \ldots & 0 \\ \lambda & 1-2\lambda & \lambda & 0 & 0 & \ldots & 0 \\ 0 & \lambda & 1-2\lambda & \lambda & 0 & \ldots & 0 \\ \vdots & & & \ddots & & \ddots & \vdots \\ 0 & \ldots & \lambda & 1-2\lambda & \lambda & \ldots & 0 \\ \vdots & & & & \ddots & & \vdots \\ 0 & 0 & \ldots & 0 & \lambda & 1-2\lambda & \lambda \\ 0 & 0 & 0 & \ldots & 0 & \lambda & 1-2\lambda \end{pmatrix} \begin{pmatrix} U_1^n \\ \\ U_2^n \\ \vdots \\ \vdots \\ U_{r-2}^n \\ \\ U_{r-1}^n \end{pmatrix}$$

We denote the vector at left by U^{n+1} (state of the bar at $t = t_{n+1}$), the coefficient matrix by S^n and the vector at right by U^n (state of the bar in $t = t_n$).

Thus the equation above can be written in the compact form as

$$U^{n+1} = S^n U^n \ .$$

Note that U^n can be written in terms of U^{n-1}, U^{n-1} can be written in terms of U^{n-2}, so that we can continue until we reach U^0. This means that for any value of $n+1$, we can compute U^{n+1} if U^0 is known:

$$U^{n+1} = S^n U^n = S^n\big(S^{n-1}U^{n-1}\big) = \ldots = \big(S^n S^{n-1} \ldots S^1 S^0\big)U^0 \ . \qquad (3.1)$$

This is the alternate way to use the semi-group property of evolution differential equations, as suggested in [3]. Despite of requiring a greater amount of calculations it is amenable to a parallel implementation.
To guarantee the scheme convergence, which means that the approximations hereby defined approach indeed the true solution, the *CFL Condition* (Courant-Friedrichs-Lewy)[2] must hold. For this scheme, this condition states that $\lambda = \alpha\Delta x/(\Delta t)^2 \leq \frac{1}{2}$. To have it fulfilled we must choose Δt very small, as compared to Δx , and this leads to a large value for N , which by its turn forces too many calculations.

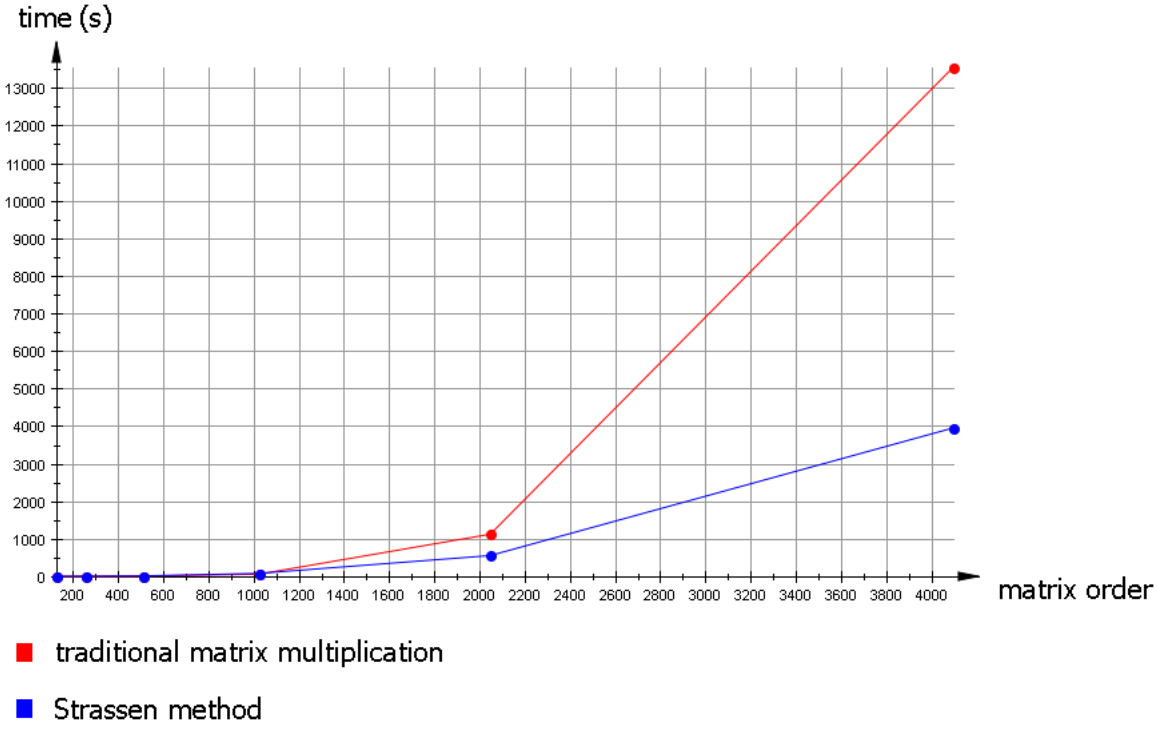
4 Matrix Multiplications

To get the product $S^n S^{n-1} \ldots S^1 S^0$ in (3.1) we will use *Strassen algorithm*, which requires matrices whose order must be a power of 2. As the coefficients matrices order is $(r-1)$x$(r-1)$, the bar must be split in $r = 2^p + 1$ (p is a natural) parts. Strassen algorithm requires fewer multiplications than the traditional matrix multiply algorithm. Indeed, given a square matrix of order 2^p, the number of multiplications in the traditional algorithm is $O((2^p)^3)$ while Strassen algorithm asymptotically requires $O((2^p)^{2.807})$. Nevertheless, Strassen algorithm requires a larger amount of sums[3] than the traditional algorithm. On acount of this, Strassen algorithm becomes more efficient only for matrices of high order. For further details about Strassen algorithm, consult [2].
The chart below shows the performance of traditional and Strassen algorithms for matrix multiplications[4].

[2]More information about this condition can be seen in [1].

[3]Processors calculate sums way faster than multiplications.

[4]All tests in this article were performed with LNCC's (Laboratório Nacional de Computação Científica) Sunhpc cluster. Each machine has the following specification: Sun Blade



For matrix with order not exceeding 1024, the traditional algorithm is more effective. As the order gets to 2048, Strassen algorithm fairs better and this improvement becomes stronger as matrices orders increase. In addition, the Strassen algorithm allows a high level of parallelism, which plainly justifies its employment for this problem.

5 Parallelism

Strassen algorithm runs as follows to get C as the product of two even order square matrices A and B . First split the matrices in 4 square blocks with the same order:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

x6250 Model, 2 Processors Intel Xeon E5440 Quad Core, giving 8 cores, and 16 GB PC2-5300 DDR2 memory

Instead of straightly performing the block multiplications on the right hand side, the calculation effort is lowered, as we carry only the 7 block multiplications that follows:

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \qquad P_2 = (A_{21} + A_{22})B_{11}$$
$$P_3 = A_{11}(B_{12} - B_{22}) \qquad P_4 = A_{22}(B_{21} - B_{11})$$
$$P_5 = (A_{11} + A_{12})B_{22} \qquad P_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

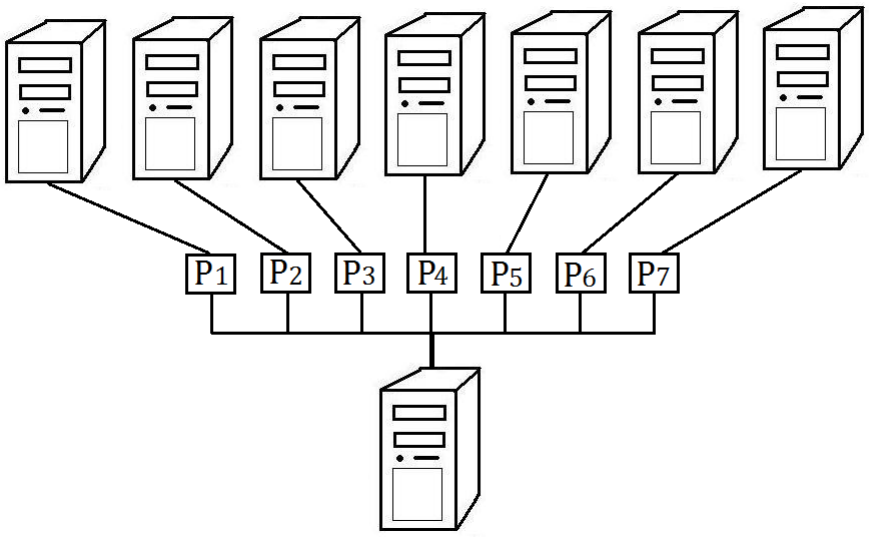
The blocks in C are then obtained from the matrices P_i through additions, as shown below:

$$C_{11} = P_1 + P_4 - P_5 + P_7$$
$$C_{12} = P_3 + P_5$$
$$C_{21} = P_2 + P_4$$
$$C_{22} = P_1 + P_3 - P_2 + P_6$$

Note that each P_i is defined as a product of sub-blocks from A and B – more precisely, linear combinations of these blocks –, and this implies that we can use Strassen algorithm again to calculate each one of these products, until we reach 4x4 matrices.

For the parallelization of this algorithm, we used 8 processors, 7 of them for computing each P_i and one to administrate and calculate the combinations between the P_i. The diagram below illustrates the process[5].

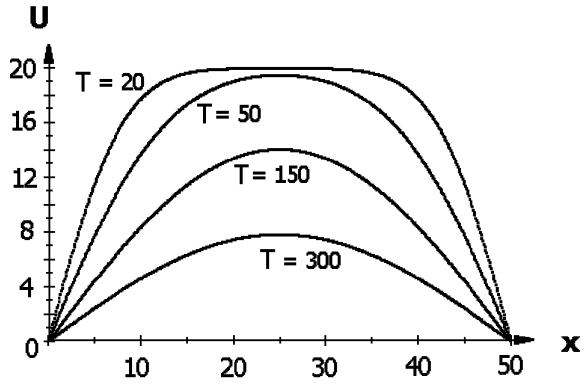
[5]We used C/C^{++} Programming Language and MPI (Message Passing Interface) library for parallelization.



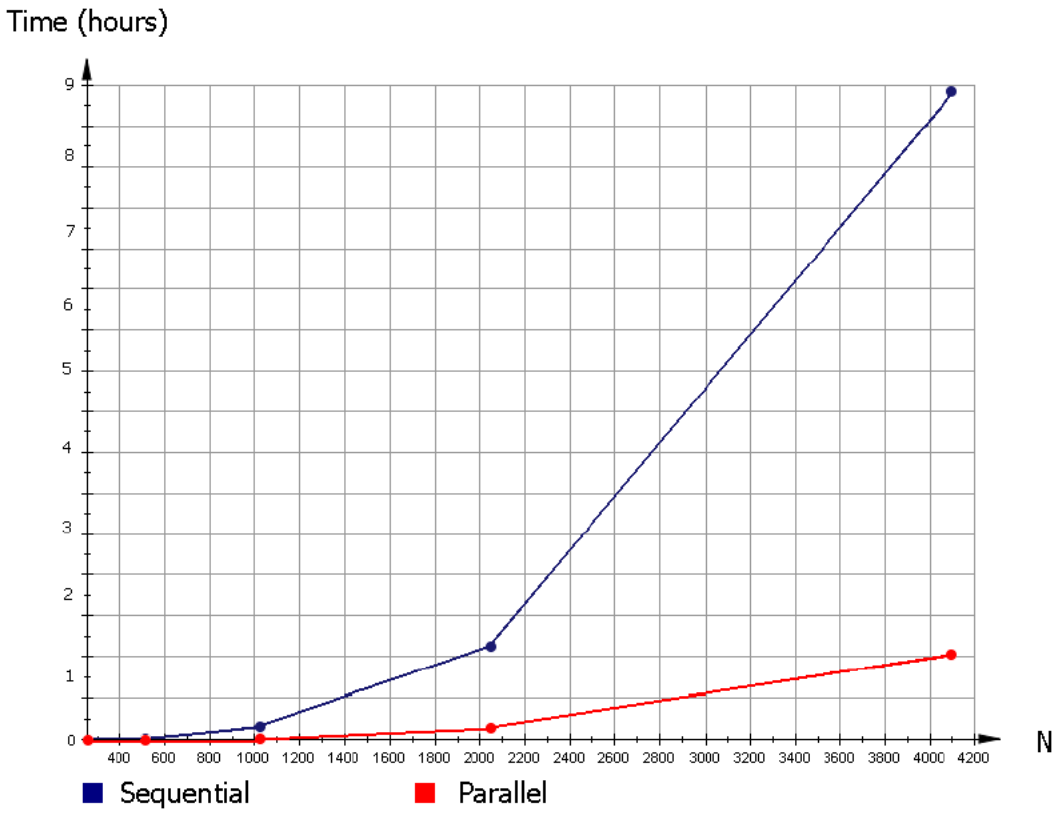
6 Computer results

Several tests for the linear homogeneous material heat conduction problem were computed sequentially and in parallel. In all cases the efficiency of the parallel algorithm was higher as compared to the sequential one.

One of these tests will be presented. Consider a 50 cm long bar, take the coefficient $\alpha = 1$, assume that the initial bar temperature is constant equal to 20^oC. We want to know the bar temperature state for $20s$, $50s$, $150s$ and $300s$. The above described algorithm generated the chart below which exhibits the sought data.



We also present a comparison chart between the algorithms, which clearly shows that the parallel algorithm outperforms the sequential one. As already observed, this improvement is stronger for matrices of high order. In fact, the higher the matrix order, the greater the difference in efficiency between the algorithms.



Although the performance difference between the algorithms remains clear from the graphic above, a more precise comparison is given by the ratio between the execution time of the parallel algorithm and the sequential. This ratio is called *speed-up*. The average speed-up for this test was roughly 7.25, which means that the parallel algorithm is 7.25 times faster than the sequential one.

We recall that *Amdhal Law* relates the speed-up and the effective use of each processor in the parallel algorithm:

$$\text{speed-up} = \frac{1}{(1-P) + P/M} \ ,$$

P: Percentage of Parallelization,
M: Quantity of Processors.

As we are using 8 processors and the speed-up is 7.25, Amdhal Law allows us to obtain the percentage of parallelization. Since the sequential part of the algorithm is relatively small, we conclude that the percentage of parallelization is 98%, which means that each processor was used close to its maximum performance.

7 Conclusion

Computational treatment for many mathematical models may become unfeasible with sequential algorithms, while parallel computing may lead to a breakthrough as regards to processing time.
The simple example hereby presented shows some points sought in parallel computing research.
An area with many open problems, parallel computing is currently being used for heavy computations in economic models, in meteorology forecasts, in engineering simulations, in biology and medical modeling, among other applications.

Acknowledgements

The authors gratefully acknowledge the financial support[6] of CAPES and FAPERJ (Grant Proc. 23038.000611/2010-14).

References

[1] COURANT, R.; FRIEDRICHS, K.; LEWY, H.: On the Partial Difference Equations of Mathematical Physics, IBM Journal of Research and Development, 11, 215–234, Mar. 1967; Original in German: Über die partiellen Differenzengleichungen der mathematischen Physik, Mathematische Annalen, 100(1), 32–74, 1928.

[6] *Cooperação entre as Pós-Graduações em Modelagem Computacional e Ciências Computacionais LNCC/UERJ* – FAPERJ 12/2009

[2] CORMEN, T.H.; LEISERSON, C.E.; RIVEST, R.L.; STEIN, C.: Introduction to Algorithms, MIT Press, Boston, Third Edition, 2001.

[3] MOURA, CARLOS A. de; CASTRO, M. CLICIA S. de; CASTRO, VINICIUS B. de: A strategy for parallel implementation of finite difference numerical schemes associated to evolution differential equations. Proc. Appl. Math. & Mechanics, 7, 2020013-2020014, 2007, DOI: 10.1002/pamm.200700084. Presented at: ICIAM-07 International Conference on Industrial and Applied Mathematics, Zurich.

[4] GUSTAFSON, J.L.; MONTRY, G.R.: Development of Parallel Methods for a 1024-processor Hypercube, SIAM Jour. Sc. Stat. Computing, 9(4), 609–638, Jul. 1988.