# Um Método Numérico com Paralelismo no Tempo para Aproximar Soluções de EDP's

**Washington S. da Silva**[1], **Maria C. S. de Castro**[2], **Carlos A. de Moura**[2]

[1] CEFET/RJ – UnED Itaguaí – Itaguaí, RJ – Brasil

[2]Instituto de Matemática e Estatística (IME)
Universidade do Estado do Rio de Janeiro (UERJ) – Rio de Janeiro, RJ – Brasil

wsdasilva14@gmail.com, mariaclicia@gmail.com, demoura@ime.uerj.br

***Abstract.*** *This article presents a numerical method with parallelism in time and investigate its computational feasibility. This numerical method is associated with problems in the initial condition and boundary condition for partial differential equations (evolutionary). Most numerical methods associated with evolutionary partial differential equations and traditionally found in the literature include only the parallelism in space. The implementation was written in C language using the MPI library for a cluster with multiple cores. Performance analysis of obtained results from the experiments reveal a scalable numerical method and requires little communication among processors.*

***Resumo.*** *Este artigo apresenta um método numérico com paralelismo no tempo e investiga a sua viabilidade computacional. Esse método numérico está associado a problemas de condição inicial e de contorno para equações diferenciais parciais (evolutivas). A maioria dos métodos numéricos associados a equações diferencias parciais evolutivas e tradicionalmente encontrados na literatura contemplam apenas o paralelismo no espaço. A implementação foi desenvolvida em linguagem C utilizando a biblioteca MPI, para um cluster com mútiplos cores. A análise de desempenho dos resultados obtidos com os experimentos revelam um método numérico escalável e que exige pouca comunicação entre processadores.*

## 1. Introdução

Muitos fenômenos físicos são modelados matematicamente por equações diferenciais parciais (EDP's). Quando esses fenômenos são modelados com EDPs em função do espaço e do tempo são denominadas EDPs evolutivas. Nesse contexto, é extremamente importante o paradigma da programação paralela, pois através dele busca-se otimizar o desempenho da simulação computacional de um dado modelo matemático.

A maioria das pesquisas realizadas no intuito de paralelizar métodos numéricos associados a EDP's evolutivas concentram-se na paralelização no espaço. Podemos citar, por exemplo, a técnica de decomposição do domínio (*domain decomposition – DD*), baseada no paradigma divisão e conquista (*divide-to-conquer*) [de Moura 2002].

Podemos encontrar na literatura várias pesquisas realizadas que abordam métodos numéricos com paralelismo no tempo. Podemos citar, por exemplo, os trabalhos de Lions et al. (2001) e Maday e Turinici (2002). Nesse artigo propomos um método numérico com

paralelismo no tempo. A grande vantagem é que para determinar as soluções aproximadas de uma EDP evolutiva num passo de tempo $t = n$, com $n > 1$, não é necessário conhecer as soluções intermediárias, entre os passos de tempo $t = 1$ e $t = n - 1$, como ocorre com os métodos numéricos que contemplam apenas a paralelização espacial.

Usando a EDP da difusão do calor, que é o protótipo das EDP's parabólicas, no caso unidimensional, e a partir de um método numérico sequencial conhecido, é apresentado nesse trabalho um esquema numérico para EDP's evolutivas que pode facilmente ser paralelizado. Nesse esquema não há necessidade de calcular a solução aproximada em cada ponto interior da malha gerada na discretização do domínio onde a EDP da difusão está definida, como ocorre nos métodos numéricos tradicionalmente encontrados.

## 2. EDP da Difusão – Solução Numérica

Considere o problema de condução do calor numa barra composta de material homogêneo, de secção reta uniforme e comprimento *L*, com constante de difusividade térmica $\alpha$, que depende apenas do material. Suponha que a distribuição de temperatura ao longo da barra seja unidimensional. Nessas condições, é possível modelar a evolução da temperatura considerando a EDP

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

onde a função $u = u(x, t)$ nos dá a temperatura da barra no instante $t$ no ponto $x \in [0, L]$, $u_t$ é a derivada de $u$ com relação ao tempo $t$ e $u_{xx}$ é a derivada de segunda ordem de $u$ com relação à variável $x$, ao longo do comprimento $L$ da barra.

A condição inicial deve contemplar a distribuição da função $u$ no início do processo (quando $t = 0$). Assim, a condição inicial deve ser expressa por uma função supostamente conhecida

$$u(x, 0) = f(x), \qquad x \in [0, L]$$

Quanto às condições de fronteira, consideramos a Condição de Dirichlet, a saber,

$$\begin{cases} u(0, t) = g(t) \\ u(L, t) = h(t) \end{cases}, \qquad t \geq 0$$

em que $g(t)$ e $h(t)$, sendo funções definidas nos extremos da barra e que dependem apenas do tempo $t$, são supostamente conhecidas.

Resolver computacionalmente a equação (1) significa, na nossa abordagem, discretizar o domínio contínuo da EDP, gerando então uma *malha*, e obter uma solução aproximada $U(x, t)$ por meio de diferenças finitas [Cunha 2003].

Para isso, considere $\Omega = [0, L] \times [0, T]$ o domínio de $u(x, t)$ e observe que dividindo o intervalo $[0, L]$ em $r$ partes iguais de comprimento $\Delta x$ e, de modo análogo, o intervalo $[0, T]$ em $s$ partes iguais de comprimento $\Delta t$, é gerada uma *malha* $(r+1) \times (s+1)$. Daí, fazendo $i \cdot \Delta x = x_i$ e $n \cdot \Delta t = t_n$, onde $[x_i, x_{i+1}]$, com $i = 0, ..., r - 1$, é a ***i-ésima*** partição ao longo do comprimento da barra e $[t_n, t_{n+1}]$, com $n = 0, ..., s - 1$, é o ***n-ésimo*** espaço de tempo no qual a temperatura evolui, é possível usar as fórmulas de diferenças finitas que aproximam $u_{xx}$ e $u_t$, dadas por

$$u_{xx}(x_i, t_n) \approx \frac{u(x_{i-1}, t_n) - 2u(x_i, t_n) + u(x_{i+1}, t_n)}{(\Delta x)^2} \tag{2}$$

$$u_t(x_i, t_n) \approx \frac{u(x_i, t_{n+1}) - u(x_i, t_n)}{\Delta t} \tag{3}$$

Para determinar o esquema explícito de soluções aproximadas para a EDP da difusão, basta substituir (2) e (3) em (1) obtendo

$$\frac{U(x_i, t_{n+1}) - U(x_i, t_n)}{\Delta t} = \alpha \frac{U(x_{i-1}, t_n) - 2U(x_i, t_n) + U(x_{i+1}, t_n)}{(\Delta x)^2}, \tag{4}$$

onde $U(x_i, t_n)$ é a solução aproximada no ponto $(x_i, t_n)$ com erro de truncamento da ordem de $\mathcal{O}(\Delta t + (\Delta x)^2)$ [Cunha 2003],[Estacio 2005].

Reescrevendo (4) com $U(x_i, t_{n+1})$ explicitado em função dos demais termos, denotando $\frac{\alpha \Delta t}{(\Delta x)^2} = \lambda$ e simplificando a notação $U(x_i, t_n)$ por $U_i^n$, obtém-se

$$U_i^{n+1} = \lambda U_{i-1}^n + (1 - 2\lambda) U_i^n + \lambda U_{i+1}^n, \tag{5}$$

que nos fornece uma aproximação da temperatura da barra no $(n+1)$-*ésimo* passo de tempo, em $x_i$, em função das aproximações obtidas em $x_{i-1}, x_i$ e $x_{i+1}$, no *n-ésimo* passo de tempo.

### 2.1. Solução Numérica Paralelizável

O esquema numérico explícito expresso por (5) pode ser representado matricialmente por

$$U^{n+1} = AU^n + \lambda b^n, \tag{6}$$

onde

$$A = \begin{pmatrix} 1-2\lambda & \lambda & 0 & \cdots & 0 \\ \lambda & 1-2\lambda & \lambda & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \lambda & 1-2\lambda & \lambda \\ 0 & \cdots & 0 & \lambda & 1-2\lambda \end{pmatrix}$$

é matriz tridiagonal Toeplitz[1] $(r-1) \times (r-1)$ e

$$U^{n+1} = \begin{pmatrix} U_1^{n+1} \\ U_2^{n+1} \\ \vdots \\ U_{r-1}^{n+1} \end{pmatrix}, \qquad U^n = \begin{pmatrix} U_1^n \\ U_2^n \\ \vdots \\ U_{r-1}^n \end{pmatrix} \qquad \text{e} \qquad b^n = \begin{pmatrix} U_0^n \\ 0 \\ \vdots \\ 0 \\ U_r^n \end{pmatrix}$$

Observe que $b^n$ também deve possuir $r-1$ entradas, sendo no máximo a primeira e última não nulas [Cunha 2003],[Baolin and Wenzhi 1994].

Expressando (6) em função dos valores iniciais da temperatura ao longo da barra, dados por $U_i^0 = f(x_i)$, com $i = 1, ..., r-1$, e da evolução da temperatura nos extremos da barra, obtidos por meio de $U_0^n = g(t_n)$ e $U_r^n = h(t_n)$, obtém-se

$$U^{n+1} = A^{n+1} U^0 + \lambda \sum_{k=0}^{n} A^k b^{n-k}, \tag{7}$$

[1]Uma matriz quadrada M é dita matriz de Toeplitz se em cada diagonal todos os seus elementos são iguais.

em que $A^{n+1} = \underbrace{A \cdot A \cdot ... \cdot A}_{n+1}$, $A^1 = A$ e $A^0 = I$.

Observe que $A^p$, com $p = 0, ..., n+1$, representa uma potência matricial de $A$ enquanto $U^0$ representa o vetor completo de condições iniciais. Note também que $b^{n-k} = [g(t_{n-k}), 0, ..., 0, h(t_{n-k}))^T$.

Fica assim definida uma solução numérica, dada pela equação (7), que pode ser paralelizada, pois depende somente dos valores de temperatura dados por $U^0$ e por cada $b^{n-k}$, com $k = 0, ..., n$. Estes são valores obtidos por meio de funções supostamente conhecidas. No entanto, embora o esquema explícito expresso por (7) seja computacionalmente simples, ele apresenta uma restrição. O passo de tempo $\Delta t$ deve ser necessariamente muito pequeno para que o método seja convergente, uma vez que, conforme a condição de Courant-Friedrichs-Levy (ou condição CFL), o processo é estável somente para $\lambda = \frac{\alpha \Delta t}{(\Delta x)^2} \leq \frac{1}{2}$. Além disso, o particionamento em $x$ deve ser suficientemente fino – gerando um $\Delta x$ muito pequeno – para que as aproximações das derivadas em $x$, por meio de diferenças finitas, possuam suficiente precisão [Cunha 2003],[de Moura 2002],[Juncosa and Young 1957].

## 3. O Método Explícito em Paralelo

Quanto ao algoritmo para calcular as potências de $A$, foi adotado o algoritmo usual de multiplicação de matrizes. Tal escolha, embora seja conhecido que esse algoritmo é da ordem de $\mathcal{O}(n^3)$ e que existem algoritmos mais rápidos como o algoritmo de Strassen (da ordem de $\mathcal{O}(n^{2,807})$) [Koç and Gan 1992],[Hattori et al. 2005],[Gilbert et al. 2010], é motivada pela simplicidade em paralelizar esse algoritmo e pela significativa redução no número de multiplicações e adições efetuadas devido ao fato de ser a matriz $A$ tridiagonal e portanto esparsa.

Com relação às potências de $A$, expressas pelo produto matricial $A^k \times A$, com $k = 0, ..., n$, não é difícil mostrar que cada *j-ésimo* elemento de $A_i^k \times A$ pode ser expresso por

$$\begin{cases} (A_i^k \times A)_j = \lambda \cdot a_{i(j-1)}^{(k)} + (1 - 2\lambda) \cdot a_{ij}^{(k)} + \lambda \cdot a_{i(j+1)}^{(k)} \\ \\ j = 1, ..., r-1, \quad i = 1, ..., r-1, \quad a_{i0}^{(k)} = 0 \quad \text{e} \quad a_{ir}^{(k)} = 0 \end{cases} \tag{8}$$

em que $a_{ij}^{(k)}$ é o *j-ésimo* elemento da matriz linha $A_i^k$.

Com base em (8) é imediato notar que o produto $A^k \times A$ é da ordem de $\mathcal{O}(n^2)$.

Além do produto matriz-matriz há também na equação (7) o produto matriz-vetor, dado por $A^k b^{n-k}$. Também não é difícil mostrar que cada elemento de $A^k b^{n-k}$ pode ser expresso por

$$\begin{cases} A_i^k b^{n-k} = a_{i1}^{(k)} \cdot U_0^{n-k} + a_{i(r-1)}^{(k)} \cdot U_r^{n-k} \\ \\ i = 1, ..., r-1, \quad 1 \leq k \leq n \end{cases} \tag{9}$$

em que $U_0^{n-k} = g(t_{n-k})$ e $U_r^{n-k} = h(t_{n-k})$.

Conhecendo (8) e (9) é possível expressar (7) de maneira pontual. Com efeito, é possível expressar $U_i^{n+1}$ por meio de

$$U_i^{n+1} = \lambda b_i^n + \lambda \sum_{k=1}^{n} (a_{i1}^{(k)} \cdot U_0^{n-k} + a_{i(r-1)}^{(k)} \cdot U_r^{n-k}) + A_i^{n+1} U^0, \tag{10}$$

com

$$\begin{cases} i = 1, ..., r-1 \\ \\ b_1^n = U_0^n, \quad b_2^n = ... = b_{r-2}^n = 0 \quad \text{e} \quad b_{r-1}^n = U_r^n \\ \\ U_0^{n-k} = g(t_{n-k}) \quad \text{e} \quad U_r^{n-k} = h(t_{n-k}) \end{cases}$$

A grande vantagem do esquema numérico encontrado é que, em virtude de seu caráter pontual, pode-se calcular apenas as soluções que, de fato, sejam procuradas, sem precisar calcular as demais soluções, como acontece em (5).

### 3.1. O Algoritmo Paralelo

A ideia básica no algoritmo paralelo proposto é fazer com que cada processador calcule o mesmo número de soluções $U_i^{n+1}$. Assim, o cálculo de cada solução $U_i^{n+1}$, por seu respectivo processador, pode ser expresso da seguinte maneira:

1. Calcule $\lambda b_i^n$. Na verdade, quando $i = 2, ..., r-2$, basta considerar $\lambda b_i^n = 0$ enquanto $\lambda b_1^n = \lambda \cdot g(t_n)$ e $\lambda b_{r-1}^n = \lambda \cdot h(t_n)$;
2. Calcule $\lambda \sum_{k=1}^{n} (a_{i1}^{(k)} \cdot U_0^{n-k} + a_{i(r-1)}^{(k)} \cdot U_r^{n-k})$ e some a $\lambda b_i^n$, observando que para $k = 2, ..., n$, deve-se executar primeiro (8), calculando cada $(A_i^{k-1} \times A)_j$ e encontrando então $A_i^k$;
3. Por fim, conhecendo $A_i^n$, do item anterior, deve-se executar novamente (8) e obter $A_i^{n+1} = A_i^n \times A$, afim de encontrar $A_i^{n+1} U^0$, onde $U_i^0 = f(x_i)$, e somar ao resultado obtido parcialmente no item anterior, determinando então a solução $U_i^{n+1}$.

Este algoritmo, quando executado sequencialmente, tem complexidade computacional da ordem de $\mathcal{O}(n^3)$.

Deve-se lembrar que há uma restrição ($\lambda = \frac{\alpha \Delta t}{(\Delta x)^2} \leq \frac{1}{2}$), inerente ao método explícito dado por (5), para que se tenha a garantia de estabilidade do esquema numérico proposto em (10).

Sabendo que $\Delta x = \frac{L}{r}$, $\Delta t = \frac{T}{s}$ e $\frac{\alpha \Delta t}{(\Delta x)^2} \leq \frac{1}{2}$, não é difícil mostrar que

$$s \geq \frac{2\alpha T r^2}{L^2} \tag{11}$$

Se $\alpha = 1$, $T = 20$, $r = 4000$ e $L = 50$, por exemplo, então $s \geq 256000$.

## 4. Resultados e Discussões

Nesta seção é apresentado um problema de valores inicial e de contorno para uma EDP da difusão do calor. Nesse problema constam os dados explícitos de entrada que devem ser usados no programa baseado no algoritmo apresentado. Há também uma breve descrição dos ambientes de execução usados para desenvolver e submeter o programa do esquema numérico proposto. Alguns resultados obtidos com a execução do programa baseado no algoritmo apresentado na seção anterior são apresentados e analisados nesta seção.

### 4.1. O Problema Proposto

Considere o seguinte problema de valores inicial e de contorno para a EDP que representa a difusão do calor numa barra composta de material homogêneo, de secção reta uniforme e comprimento $L = 50$ cm, com constante de difusividade térmica $\alpha = 1\ \frac{\text{cm}^2}{\text{s}}$, que depende apenas do material:

$$\begin{cases} u_t = u_{xx}, \quad (x,t) \in (0,50) \times (0,+\infty) \\ u(x,0) = f(x) = 40 - 2x + 0,04x^2, \quad x \in [0,50] \\ u(0,t) = g(t) = 40 + t, \quad t \geq 0 \\ u(50,t) = h(t) = 40 + 0,5t, \quad t \geq 0 \end{cases}$$

Na implementação do algoritmo referente ao esquema numérico proposto foram considerados os valores mínimos de $s$ (veja inequação 11), que é o caso em que $\lambda = \frac{\alpha \Delta t}{(\Delta x)^2} = \frac{1}{2}$. Nesse caso, a diagonal principal da matriz $A$ é zerada e, sendo assim, é possível (e conveniente) simplificar a relação dada por (8). Neste caso, é possível reescrever cada elemento $A_i^k \times A$ por

$$\begin{cases} (A_i^k \times A)_j = \lambda \cdot (a_{i(j-1)}^{(k)} + a_{i(j+1)}^{(k)}) \\ \\ j = 1, ..., r-1, \quad i = 1, ..., r-1, \quad a_{i0}^{(k)} = 0 \quad \text{e} \quad a_{ir}^{(k)} = 0 \end{cases} \tag{12}$$

### 4.2. Ambientes de Programação

O programa baseado no algoritmo apresentado na Seção 3, em suas versões paralela e sequencial, foi desenvolvido na linguagem *C*, utilizando a biblioteca de programação paralela MPI [Quinn 2003]. A complexidade computacional da versão sequencial é da ordem de $\mathcal{O}(n^3)$. O programa foi desenvolvido num *cluster* do Instituto de Matemática e Estatística da Universidade do Estado do Rio de Janeiro (IME/UERJ), com 8 processadores.

Já os testes para investigar o desempenho computacional do algoritmo proposto foram realizados no computador Uranus (SGI Altix ICE 8400) do Núcleo Avançado de Computação de Alto Desempenho da Universidade Federal do Rio de Janeiro (NACAD-UFRJ). No Uranus, os arquivos das versões sequencial e paralela foram compilados com o GCC e a diretiva de compilação $-$O.

O Uranus possui 128 CPUs com 640 *cores* e 64 nós de processamento. Utiliza os compiladores Intel e GNU com suporte às principais versões MPI como OpenMPI. Os testes de desempenho foram realizados em 1, 2, 4, 8, 16, 32, 64 e 128 *cores* utilizando a biblioteca OpenMPI.

### 4.3. Resultados Obtidos

Nesta seção são apresentados alguns resultados obtidos com a implementação do método numérico proposto. Os resultados completos podem ser vistos em [Silva 2014].

A Tabela 1 apresenta os valores dos tempos totais de execução ($T_T$) em segundos, de *speedup* ($S(p)$) e da *eficiência* ($\mathcal{E}$ (%)) para um particionamento $r = 4000$ no comprimento da barra e tempo estimado $T = 20$ segundos. Neste caso, o particionamento

temporal deve ser $s = 256000$, gerando uma malha $4000 \times 256000$. Os tempos de *speedup* e *eficiência* foram obtidos comparando-se o tempo de computação paralela com a execução do algoritmo paralelo usando apenas um *core*.

**Tabela 1. $T_T$, $S(p)$ e $\mathcal{E}$ para $r = 4000$ e $T = 20$ segundos.**

| | Número de *cores* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
| $T_T$ | 31234,28 | 15625,59 | 7826,96 | 3917,37 | 1958,43 | 979,98 | 494,10 | 254,10 |
| $S(p)$ | | 2,00 | 3,99 | 7,97 | 15,95 | 31,87 | 63,21 | 126,86 |
| $\mathcal{E}$ (%) | | 100,00 | 99,75 | 99,62 | 99,69 | 99,59 | 98,77 | 99,11 |

Observando a Tabela 1 podemos destacar os seguintes resultados:

- à medida que dobrou-se o número de *cores* o tempo total de execução foi reduzido aproximadamente à metade do tempo anterior;
- os valores de *speedup* foram muitos próximos do número de *cores* correspondentes;
- a *eficiência*, a menos de décimos, foi muito próxima de 100%.

Esses resultados mostram que para uma malha em que $r = 4000$ o algoritmo tem *speedup* linear.

As Figuras 1, 2 e 3 se referem as linhas da Tabela 1 e ilustram essas conclusões.

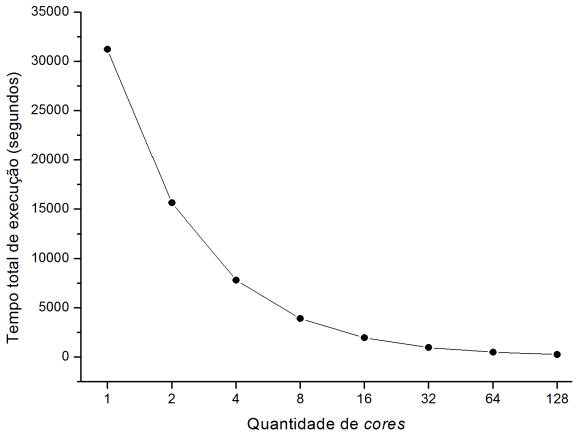


**Figura 1. Tempos totais de execução para $r = 4000$ e $T = 20$ segundos.**

## 5. Conclusões

Nesse artigo foi apresentado um método numérico com paralelismo no tempo para aproximar soluções de EDP's do calor. Com esse esquema numérico obtivemos as soluções aproximadas de um dado problema de condição inicial e de contorno para uma EDP do calor. Devido ao caráter pontual deste esquema numérico, expresso por (10), não foi difícil elaborar e implementar o correspondente algoritmo paralelo.

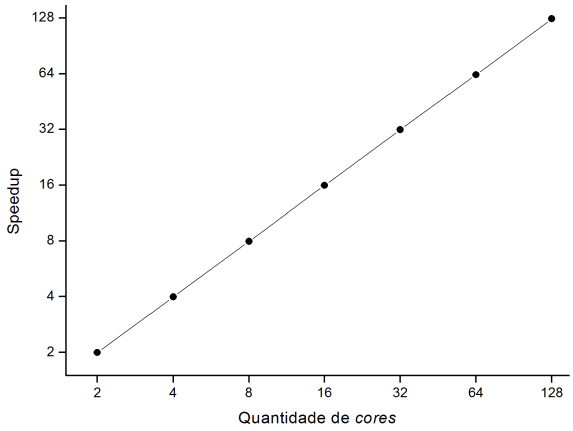


**Figura 2. Curva de *Speedup* para $r = 4000$ e $T = 20$ segundos.**

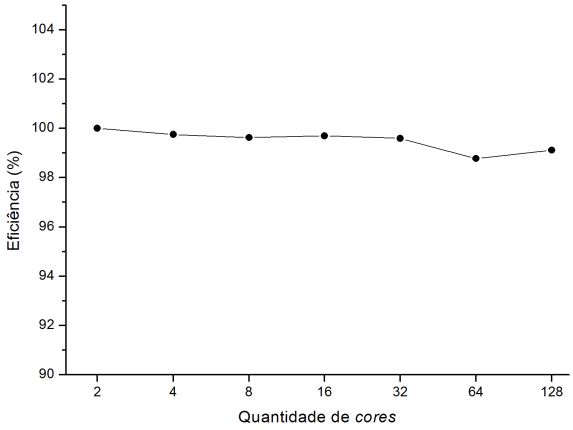


**Figura 3. Curva de *Eficiência* para $r = 4000$ e $T = 20$ segundos.**

Embora o método numérico apresentado esteja associado a um problema linear, no qual se concentram muitas pesquisas na área de métodos numéricos, podemos destacar pelo menos dois pontos importantes que justificam a apresentação e implementação do método proposto:

- dado um problema não-linear, é possível aproximá-lo, com diferentes esquemas numéricos, por problemas lineares;
- diferentemente dos esquemas numéricos para EDP's evolutivas tradicionalmente encontrados, o método numérico proposto considera o paralelismo no tempo.

A consequência imediata do que foi citado anteriormente, no segundo ponto, é que no cálculo das soluções aproximadas de um dado problema de EDP evolutiva, não é preciso obter as soluções intermediárias entre o tempo inicial $t_0 = 0$ e o tempo estimado $t_{n+1} = T$.

Quanto à comunicação (envio e recebimento de mensagens) entre os processos envolvidos na execução do programa baseado no algoritmo paralelo, esta se dá apenas entre o processo *master* e os outros processos (não há comunicação entre os outros processos).

Isto ocorre apenas no momento em que o *master* envia os dados de entrada aos demais processos e quando recebe as soluções obtidas pelos mesmos, não havendo comunicação enquanto realizam o processamento.

O esquema numérico expresso em (10) mostrou-se escalável ao ser implementado. À medida que o volume de dados aumenta, o algoritmo paralelo apresenta um comportamento linear.

Já com relação aos tempos totais de execução, eles aumentam consideravelmente à medida que o volume de dados aumenta. À medida que um particionamento $r$ é dobrado o tempo total de execução aumenta, aproximadamente, 16 vezes mais. Fato que deve-se à condição CFL.

## Referências

Baolin, Z. and Wenzhi, L. (1994). On alternating segment Crank-Nicolson scheme. *Parallel Computing*, 20(6):897–902.

Cunha, M. C. C. (2003). *Métodos numéricos*. Editora da UNICAMP.

de Moura, C. A. (2002). Parallel numerical methods for differential equations. In *Models for Parallel and Distributed Computation*, pages 279–313. Springer.

Estacio, K. C. (2005). *Solução Numérica de Equações Diferenciais a Derivadas Parciais por Diferenças Finitas*. EDUSP.

Gilbert, J. R. et al. (2010). Highly parallel sparse matrix-matrix multiplication. *arXiv preprint arXiv:1006.2183*.

Hattori, M., Ito, N., Chen, W., and Wada, K. (2005). Parallel matrix-multiplication algorithm for distributed parallel computers. *Systems and Computers in Japan*, 36(4):48–59.

Juncosa, M. L. and Young, D. (1957). On the Crank-Nicolson procedure for solving parabolic partial differential equations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 53(2):448–461.

Koç, Ç. K. and Gan, S. C. (1992). Parallel matrix multiplication on networked microcomputers. *Computers & Electrical Engineering*, 18(2):145–152.

Lions, J.-L., Maday, Y., and Turinici, G. (2001). Résolution d'edp par un schéma en temps «pararéel». *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, 332(7):661–668.

Maday, Y. and Turinici, G. (2002). A parareal in time procedure for the control of partial differential equations. *Comptes Rendus Mathematique*, 335(4):387–392.

Quinn, M. J. (2003). *Parallel Programming*, volume 526. TMH CSE.

Silva, W. S. (2014). *Um método numérico com paralelimo no tempo para aproximar soluções de EDP's*. 2014. 74 f. Dissertação (Mestrado em Ciências Computacionais). Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2014.