Thales Oliveira Arakawa

Aplicação do OpenMP ao Método das Diferenças Finitas

Universidade Federal Fluminense - UFF

Campus Volta Redonda

Instituto de Ciências Exatas

Brasil

15 de setembro de 2021

Resumo

Este trabalho vem propor uma solução quanto as atividades práticas laboratoriais, pedagógicas, diante a necessidade de ou da realidade do distanciamento social. Para que isso seja possível usufruiremos de uma grande ferramenta, usada e abusada por todos, a internet. Junto a um novo conceito que vem ganhando espaço no cotidiano o IoT (*Internet of Things*), ou Internet das Coisas, este trabalho propiciará as práticas laboratoriais de forma remota, não só com o intuito de perpetuar as atividades exigidas pela grade curricular mas também como uma possibilidade de expandir esse acesso àqueles que não o possuem.

Palavras-chaves: Distanciamento social, IoT, Internet das Coisas, laboratório remoto.

Sumário

1 Introdução

Muita das vezes, nas ciências exatas, não é possível obter soluções analíticas para os problemas. Diante deste fato, muitos matemáticos buscam compreender e desenvolver ferramentas para obter um resultado aproximado.

Um exemplo é o método das diferenças finitas, que faz o uso das séries de Taylor; que é talvez a ferramenta de aproximação matemática mais disseminada no meio. Este método busca obter uma solução, não exata, para as equações diferenciais parciais. Porém a solução para uma EDP abrange os infinitos pontos em que esta é valida e da forma que o método das diferenças finitas é proposto deveríamos calcular o maior número de pontos possíveis, de forma que através de lápis e papel seja um processo árduo.

É para esses métodos recursivos e massivos que o computador se torna uma solução. Podendo realizar milhões de cálculos por segundo este é capaz de resolver este trabalho árduo num curto espaço de tempo.

Com a evolução dos processadores, que regem essas máquinas, chegamos ao ponto em que se desenvolve a computação paralela. Procedimento este em que existem diversos componentes realizando cálculos simultaneamente.

Para implementar a computação paralela foram desenvolvidas API's (Application Programming Interface), como o OpenMP, que são capazes de serem implementados aos códigos, ditos seriais, e possibilitar a paralelização dos procedimentos matemáticos realizados pelo mesmo.

2 Objetivos

Como a resolução de EDP's, via métodos numéricos, é de grande importância para para a ciência e indústria e também por esse procedimento exigir tempo para obtenção de solução, dependendo do tamanho do sistema, se faz necessário a implementação de computação paralela para ganho de performance.

Este trabalho vem com o intuito de analisar a escalabilidade do método das diferenças finitas quanto ao número de núcleos de processamento virtuais (*threads*) disponíveis e comparar a eficiência atingida quanto ao tempo de processamento gasto.

3 Fundamentos Teóricos

3.1 Método das Diferenças Finitas

3.1.1 A Diferença Finita

A Diferença Finita nada mais é do que uma aproximação, para as derivadas, utilizando-se da série em expansão de Taylor, ou seja:

$$f(x_i + h_x) = \sum_{n}^{\infty} f^n(x_i)\frac{h_x^n}{n!} = f(x_i) + h_x f'(x_i) + h_x^2\frac{f''(x_i)}{2} + ... \tag{3.1}$$

onde $h_x = x - x_i$. A Equação 3.1 é dita progressiva, temos também a sua forma regressiva, dada por:

$$f(x_i - h_x) = \sum_{n}^{\infty} f^n(x_i)\frac{(-h_x)^n}{n!} = f(x_i) - h_x f'(x_i) + h_x^2\frac{f''(x_i)}{2} + ... \tag{3.2}$$

Da diferença entre as Equações 3.1 e 3.2, obtemos a foma centrada (Eq. 3.3) que tem como característica uma convergência mais abrupta, uma vez que os termos h_x de ordem par se cancelam.

$$f(x_i + h_x) - f(x_i - h_x) = 2h_x f'(x_i) + 2h_x^3\frac{f'''(x_i)}{6} + ... \tag{3.3}$$

Determinando $h_x < 1$, de forma que $h_x^3 \ll 1$, teremos então uma aproximação de $f'(x_i)$ dada por:

$$f(x_i + h_x) - f(x_i - h_x) \approx 2h_x f'(x_i) \Rightarrow f'(x_i) \approx \frac{f(x_i + h_x) - f(x_i - h_x)}{2h_x}. \tag{3.4}$$

Definiremos então, a diferença finita centrada de ordem 1 (UFRGS, 2020), como:

$$D_{i,h_x} f(x_i) := \frac{f(x_i + h_x) - f(x_i - h_x)}{2h_x} \tag{3.5}$$

Para a derivada de segunda ordem, façamos a soma entre as Equações 3.1 e 3.2, obtendo-se assim:

$$f(x_i + h_x) + f(x_i - h_x) = 2f(x_i) + h_x^2 f''(x_i) + h_x^4\frac{f''''(x_i)}{12} + ..., \tag{3.6}$$

uma vez que $h_x < 1$ e, portanto, $h_x^4 \ll 1$, eliminaremos os termos de quarta ordem e superiores, obtendo a aproximação:

$$f''(x_i) \approx \frac{f(x_i + h_x) - 2f(x_i) + f(x_i - h_x)}{h_x^2} \tag{3.7}$$

que é por onde definiremos a diferença finita de ordem 2 (UFRGS, 2020), como:

$$D^2_{i,h_x} f(x_i) := \frac{f(x_i + h_x) - 2f(x_i) + f(x_i - h_x)}{h_x^2} \tag{3.8}$$

3.1.2 O Método

O método das diferenças finitas é utilizado para resolução de EDP's, principalmente, quando esta não possui solução analítica. O objetivo é obter uma solução aproximada através da implementação das diferenças finitas. Tomemos como exemplo o operador laplaciano para duas dimensões, onde

$$\nabla^2 f = \frac{\partial f}{\partial x}^2 + \frac{\partial f}{\partial y}^2 . \tag{3.9}$$

Substituindo as derivadas parciais pela diferença finita, dada pela equação 3.8, teremos então:

$$\begin{aligned} \nabla^2 f \approx & \frac{f(x_i + h_x, y_j) - 2f(x_i, y_j) + f(x_i - h_x, y_j)}{h_x^2} \\ & + \frac{f(x_i, y_j + h_y) - 2f(x_i, y_j) + f(x_i, y_j - h_y)}{h_y^2}. \end{aligned} \tag{3.10}$$

Para a equação de Laplace, teríamos então:

$$\begin{aligned} & \frac{f(x_i + h_x, y_j) - 2f(x_i, y_j) + f(x_i - h_x, y_j)}{h_x^2} \\ & + \frac{f(x_i, y_j + h_y) - 2f(x_i, y_j) + f(x_i, y_j - h_y)}{h_y^2} \approx 0 \end{aligned} \tag{3.11}$$

Uma vez que buscamos a solução de uma EDP, nosso objetivo então é determinar $f(x, y)$, logo o isolamos e obtemos:

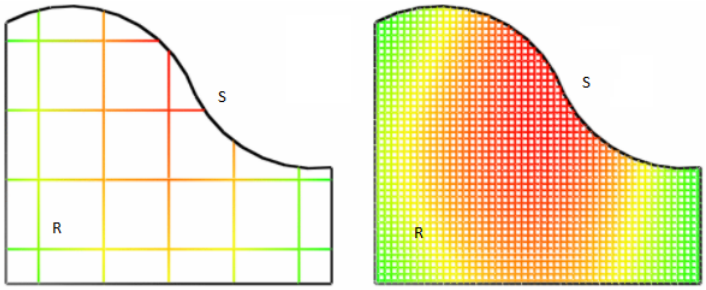
$$f(x_i, y_j) \approx \frac{h_y^2 \left[f(x_i + h_x, y_j) + f(x_i - h_x, y_j)\right] + h_x^2 \left[f(x_i, y_j + h_y) + f(x_i, y_j - h_y)\right]}{2(h_x^2 + h_y^2)}. \tag{3.12}$$

Como para toda EDP, esta possuí região($\mathbf{R}$) em que a solução é válida e também as condições de contorno($\mathbf{S}$). Uma vez que desconhecemos o comportamento da $f(x, y)$,

poderíamos atribuir quaisquer valores a ela e para cada ponto utilizaríamos a equação 3.12 para atualizar o valor relativo a este ponto. Será necessário então realizar uma varredura em todos os pontos da região em que a EDP se encontra até que, por convergência, a solução seja obtida.

O processo de determinar quais pontos serão computados, é chamado de discretização. É construído uma malha sobre região de tal forma que seu espaçamento horizontal seja de h_x e vertical de h_y

Figura 1 – Discretização de uma Região.



Fonte: Alto Qi.

3.1.3 OpenMP

O OpenMP é uma API, criada em 1997 para Fortran, que possibilita a paralelização de um código serial. Através de diretivas compostas por construtores e cláusulas adicionadas ao código e um compilador que possua suas bibliotecas o programa poderá ter sua atividade dividida entre os processadores virtuais (*threads*) disponíveis para computação (OPENMP, 2012a).

Atualmente o OpenMP possui suporte à Fortran, C/C++, Java, Python e outras linguagens mais (OPENMP, 2012b).

Figura 2 – Exemplo de criação de região paralela.

```
#pragma omp parallel
{
        do{
                erro=opDif(map,data,erro);
        }while(tol<erro);
}
```

Fonte: Autor.

A partir do momento que uma diretiva "#pragma omp parallel"é inserida no programa, é criada uma região paralela. Nesta região poderemos utilizar algumas funções

da biblioteca do OpenMP de forma a obter a quantidade de *threads* disponíveis e o número de identificação de cada uma. Fora dessa região o programa irá se comportar de forma serial. Além dessas funções básicas o OpenMP fornece diretivas mais inteligentes baseadas em algoritmos canônicos, como a varredura de vetores e matrizes através de *loops*.

4 Metodologia

4.1 A Equação Diferencial Parcial

A EDP utilizada para implementação do método de diferenças finitas, será a equação de Laplace (Eq. 3.9). Esta é uma equação diferencial que pode ser encontrada em todas as áreas da física, uma vez que esta procura determinar o fluxo de um campo de forças conservativo, ou seja um campo tal que possua função potencial. O trabalho abordará o contexto da eletrostática, onde:

$$\nabla^2\Phi = \nabla \cdot \vec{\nabla}\Phi = -\nabla \cdot \vec{E} = 0, \tag{4.1}$$

onde Φ é o potencial eletrostático e a Equação 4.2 é a equação de Laplace para este potencial. Vale lembrar que, das equações de Maxwell, que:

$$\nabla^2\Phi = -\nabla \cdot \vec{E} = -\frac{\rho}{\epsilon_0}, \tag{4.2}$$

portanto no problema a ser desenvolvido estará ausente de cargas. Portanto para a implementação do método das diferenças finitas, a equação utilizada (Eq. 3.12) será aquela abordada na sessão 3.1.1

4.2 Discretização

Sendo a região a ser trabalhada bidimensional então uma estrutura matricial seria de grande conveniência. Porém, ao invés de se utilizar na sua forma convencional, foi criada uma variável do tipo *struct*, através da linguagem C, onde essa estrutura é apresentada na Figura 3.

Essa variável nada mais é que um monômero para uma estrutura maior, que se dará como uma matriz. De forma análoga a uma lista duplamente encadeada, a partir do momento que esta variável é criada, basta realizar o encadeamento entre as estruturas, gerando a matriz dada pela Figura 4.

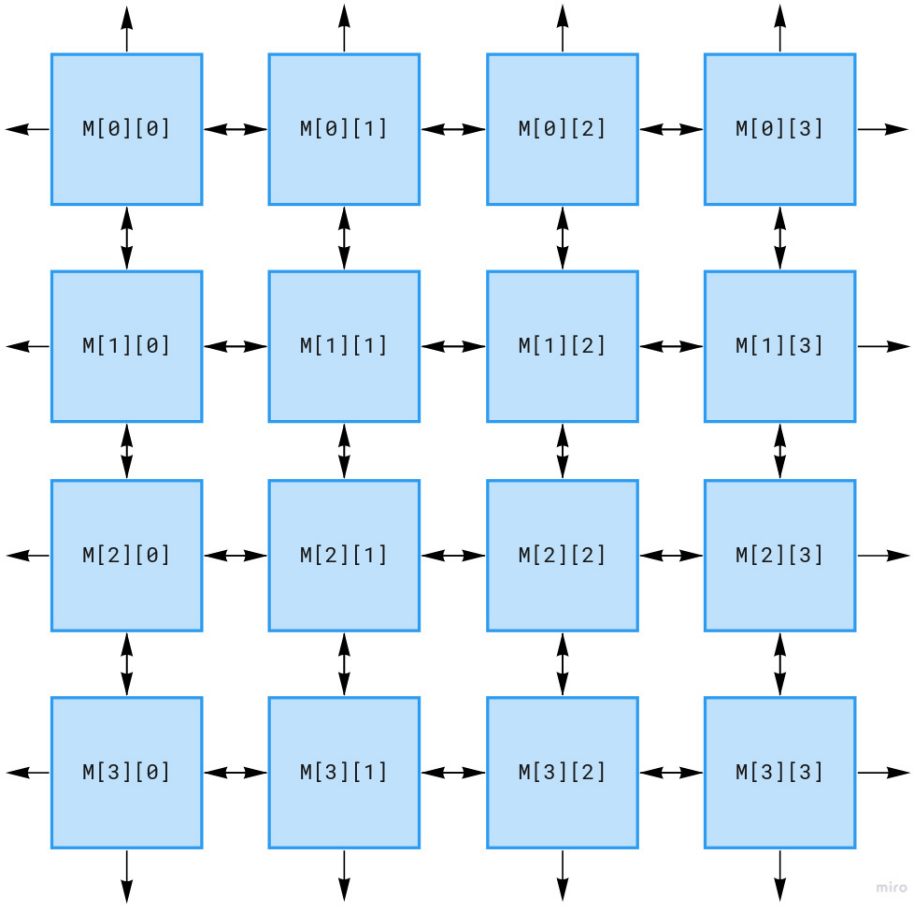
A vantagem que essa abordagem fornece é a velocidade de acesso à memória sem a necessidade de algumas *flags* de compilação, uma vez que a manipulação se dá através de ponteiros; a capacidade de se tornar uma malha fechada pelas laterais, pelo simples encadeamento das mesmas; e por cada monômero deste poder carregar informações (como status de condição de contorno) quaisquer que seja a necessidade. A desvantagem deste tipo de aplicação é uma maior necessidade de memória.

Figura 3 – Monômero da estrutura matricial.

```
struct quad
{
        struct quad *up;
        struct quad *down;
        struct quad *left;
        struct quad *right;
        .
        .
        .
};
```

Fonte: Autor.

Figura 4 – Matriz duplamente encadeada.



Fonte: Autor.

4.3 Método da Relaxação

Para solucionar a EDP, será utilizado o método da relaxação. Uma vez que já possuímos a relação matemática que cada ponto da malha deve obedecer, dado pela equação 3.12, bastará realizar varreduras sobre a mesma e ir atualizando os valores pontuais, sendo

esperado a convergência, quando a variação dos valores atualizados forem suficientemente pequenos a solução numérica será entregue.

4.4 Paralelização - OpenMP

Uma vez que a malha será construída de forma não canônica, as diretivas específicas do OpenMP para os *loop's* não serão utilizadas. Portando realizaremos uma paralelização explícita.

Criando a região paralela para o *loop* da relaxação, determinaremos as variáveis **map** (que contém a malha com os valores de f), **data** (que contém as informações sobre dimensão da malha), **chunk** (que é a divisão inteira de número de linhas por thread) e **nt** (que informará o número de threads) como compartilhadas (*shared*); e as variáveis **aux** (que determinará o trabalho de cada thread), **p** (que possui o endereço inicial de cada thread), **tid** (que possuí a identificação de cada thread) e o **erro** (relativo a cada região da malha).

4.5 Determinando Condição de Contorno

Para determinar a condição de contorno para a equação de Laplace, para a eletrostática, utilizou-se de um recurso que converte uma imagem em ASCII (protocolo de codificação). O recurso se encontra disponível na internet (<https://www.dcode.fr/binary-image>), basta realizar o *upload* da imagem e escolher a escala relativa, desejada. Neste trabalho foi utilizado a logo da UFF (Universidade Federal Fluminense), que na modelagem se encontrará aterrada e enclausurada a uma caixa a $100V$.

Figura 5 – Condição de contorno fora de escala.

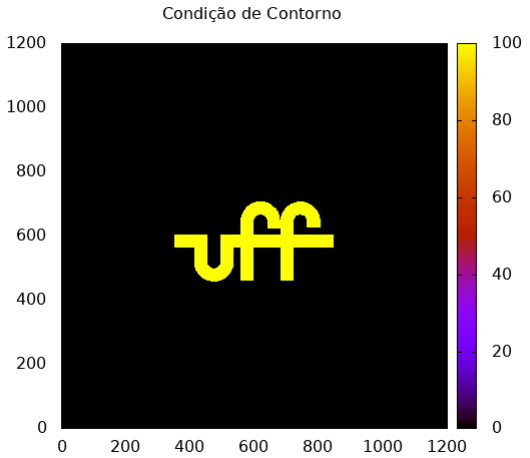


Figura 6 – Fonte: Autor.

4.6 Computadores

Para testar os programas e observar a escalabilidade quanto ao paralelismo, utilizamos:

1) Nós do supercomputador Santos Dumont (SDumont) do Laboratório Nacional de Computação Científica(LNCC,).

   a) B710

      * 2 x CPU Intel Xeon E5-2695v2 Ivy Bridge, 2,4Ghz
      * 24 núcleos (12 por CPU), totalizando de 12.096 núcleos
      * 64GB DDR3 RAM

   b) Sequana

      * 2x Intel Xeon Cascade Lake Gold 6252
      * 48 núcleos (24 por CPU)
      * 384Gb de memória RAM

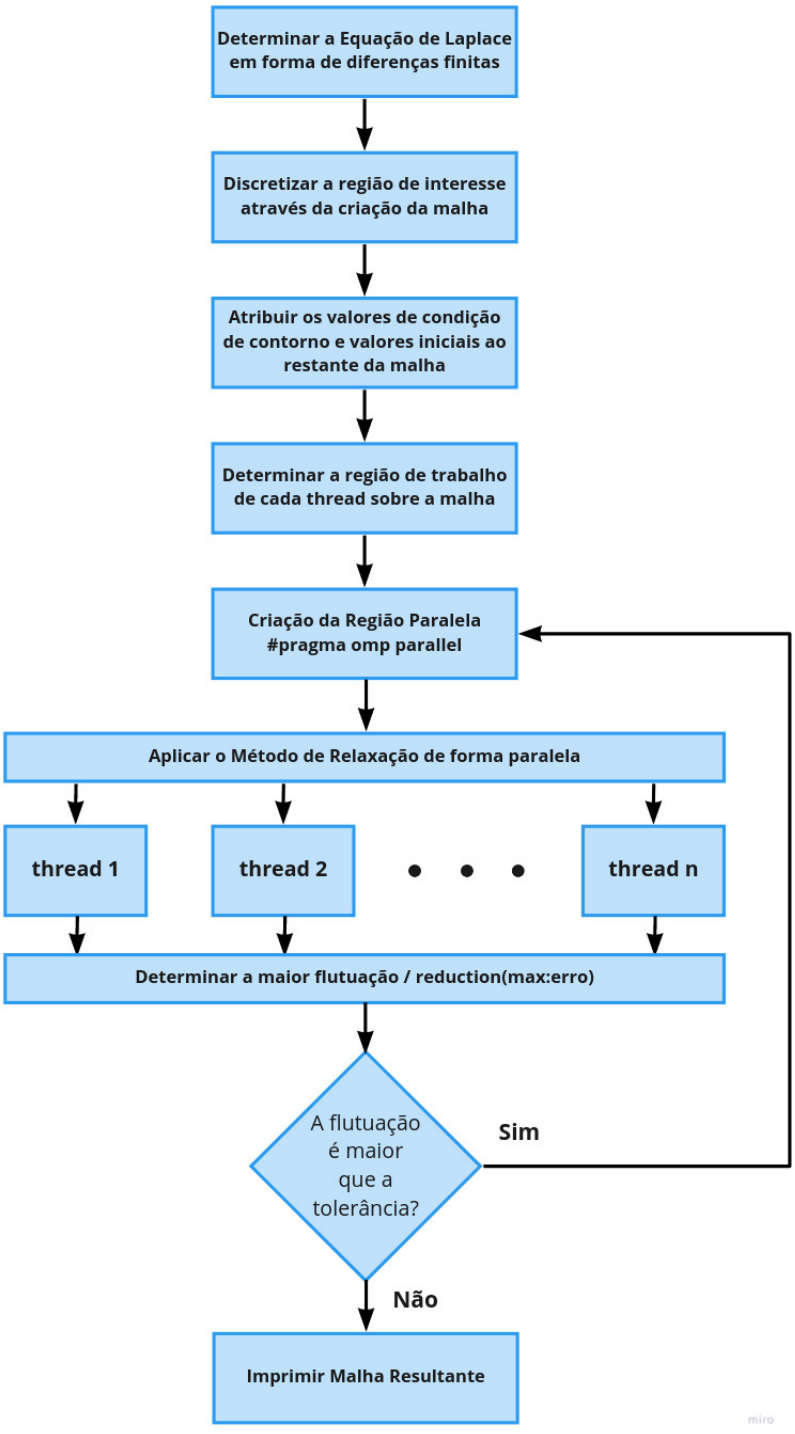
2) Computador do laboratório 107 da Universidade Federal Fluminense (UFF).

   - Intel I7-200 Little Endian, 3.4Ghz
   - 8 núcleos
   - 4Gb de memória RAM

5 Fluxograma

A Figura 7 apresenta como se dará a implementação do programa.

Figura 7 – Fluxograma do método de diferenças finitas, com o uso da relaxação e paralelizado.



Fonte: Autor.

6 Análise do Programa Serial

6.1 Determinando as melhores *flags*

Foram testadas, apenas, as *flags* de otimização padrão do compilador *GNU Compiler Collection* (GCC). Uma vez que os *loop's* não se dão de forma canônica, não faria sentido utilizar as *flags* destinadas para cálculo vetorial e matricial.

Benchmark Serial	
Flag	Tempo (s)
O0	399,824
O1	352,534
O2	324,367
O3	323,481

Tabela 1

6.2 Perfil do Programa Serial

Como esperado, devido sua construção, a função responsável pela varredura por trás do método da relaxação, chamada de *opDif*, é a que mais representa para o tempo total de execução, 99.9%.

Figura 8 – Perfil gerado pelo programa GNU profiler.

```
index % time    self  children    called     name
                                             <spontaneous>
[1]    100.0    0.02  387.71                main [1]
              387.27    0.00   22654/22654       opDif [2]
                0.17    0.00       1/1           printcc [3]
                0.13    0.00       1/1           simetria [4]
                0.10    0.00       1/1           printmap [5]
                0.04    0.00       1/1           createmap [6]
                0.00    0.00       1/1           importCC [7]
-----------------------------------------------
              387.27    0.00   22654/22654       main [1]
[2]     99.9  387.27    0.00   22654         opDif [2]
-----------------------------------------------
                0.17    0.00       1/1           main [1]
[3]      0.0    0.17    0.00       1         printcc [3]
-----------------------------------------------
                0.13    0.00       1/1           main [1]
[4]      0.0    0.13    0.00       1         simetria [4]
-----------------------------------------------
                0.10    0.00       1/1           main [1]
[5]      0.0    0.10    0.00       1         printmap [5]
-----------------------------------------------
                0.04    0.00       1/1           main [1]
[6]      0.0    0.04    0.00       1         createmap [6]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[7]      0.0    0.00    0.00       1         importCC [7]
-----------------------------------------------
```

Fonte: Autor.

6.3 Otimização do Programa Serial

Como podemos ver o perfil do programa através da Figura 8, $99,9\%$ do programa se concentrava em uma função. Sendo necessária a varredura completa da matriz e a atualização da mesma, logo não haveria nenhuma condição de saída prévia.

Quanto aos cuidados sobre a otimização das operações matemática, estas já haviam sido implementadas, uma vez que é relativamente comum problemas dado pela divisão e multiplicação de uma variável de ponto flutuante por uma inteira. Portanto todas essas operações já estavam dadas por decimais.

No mais a alteração feita, foi a remoção de funções que auxiliavam a verificar o funcionamento do programa, os comentários com o mesmo fim e a união das funções *printmap* e *printcc* em uma só.

7 Implementação do OpenMP

Uma vez que a estrutura matricial não foi implementada de maneira convencional, ou seja, não é possível o acesso direto as células dessa matriz através de índice. O princípio utilizado, uma vez que todas as células tinham acesso às informações de suas vizinhas e das condições de contorno, bastava-se caminhar pela malha e atualizar seus valores. Com base nesse processo, foi construído uma paralelização explícita, como mostrado na Figura 9.

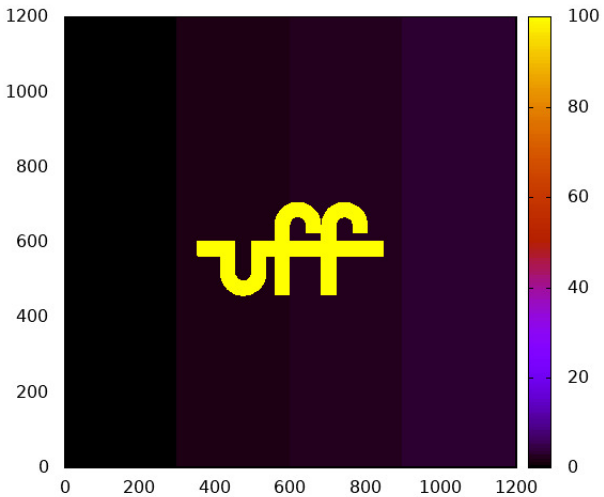
Figura 9 – Diretivas e funções do OpenMP utilizadas para paralelização do programa.

```
        //Variáveis para paralelilzação
        int nt=omp_get_max_threads();
        int chunk = d->nLin/nt;
        int resto=d->nLin%nt;
        int trabalho;
        int andamento;


#pragma omp parallel private(i,j,tid,ini,ult,andamento,trabalho) shared(p,nt, chunk,resto) reduction(max:erro)
{
        ini=p;
        ult=p;

        tid=omp_get_thread_num();
        trabalho=(tid < resto) ? chunk + 1 : chunk;
        andamento=(tid>resto) ? tid*chunk + resto : tid*chunk + tid ;

        for(i=0;i<andamento;i++){
             ult=ult->down;
        }
```

Fonte: Autor.

Basicamente há a abertura da região paralela e a determinação do tamanho do trabalho para cada *thread*. Na Figura 10 podemos ver a região divida em 4 *threads*.

Figura 10 – Separação da Malha em *threads*.



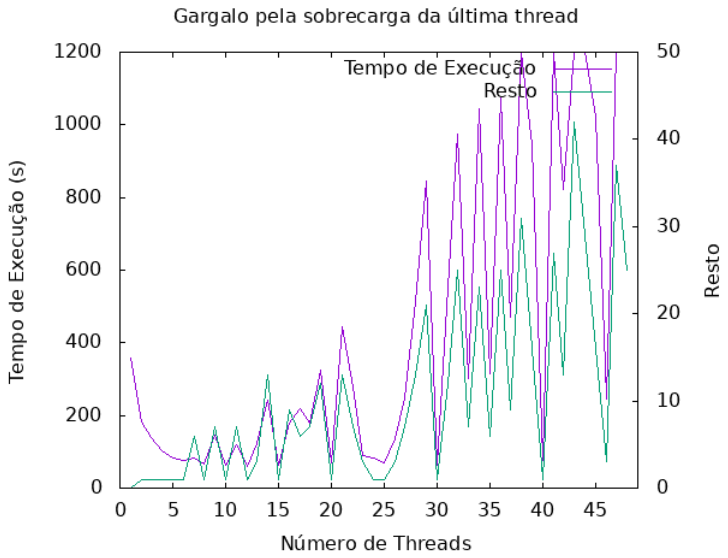
Fonte: Autor.

A variável *trabalho* é a que de fato indicará qual o tamanho da região a ser trabalhada. A cláusula *reduction* junto do parâmetro *max:erro* determinará a maior flutuação encontrada em uma varredura e será comparada posteriormente com a tolerância para avaliar a necessidade de interrupção do método da relaxação.

8 Resultados

A primeira versão do programa dividia o número de linhas (que graficamente se encontram na vertical) pelo número de *threads* e se houvesse resto nesta divisão este era todo acumulado na última *thread*. Dessa forma havia uma sobrecarga na última *thread* e portanto um gargalo proporcional ao resto dessa divisão, como podemos ver na Figura 11 o tempo de execução no SDumont (LNCC), correspondente ao Sequana.

Figura 11 – Ausência de escalabilidade devido a má divisão das regiões da malha. Pontos acima de 20 minutos foram execuções interrompidas por tempo limite.



Fonte: Autor. Dados gerados no Sequana/LNCC.

Foi elaborado outro algoritmo, mostrado nas seções anteriores na Figura 10, para corrigir a sobrecarga da última *thread* obtendo-se a escalabilidade do programa, mostrado na Figura 12.

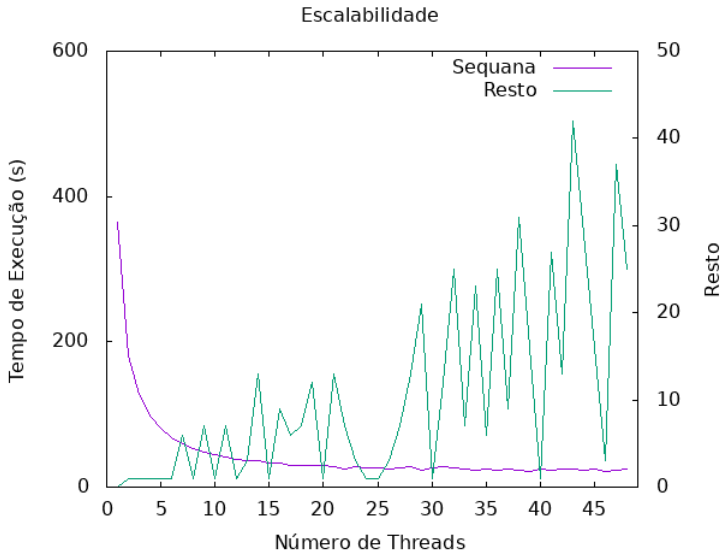
Para comparar a performance dos computadores disponíveis, foi gerado o gráfico dado pela Figura 13.

Tomando como parâmetro o tempo serial do nó B710, obtivemos o comportamento dado pelo gráfico da Figura 14. Note que o tempo de uma *thread* para o nó B710 foi maior que em sua versão serial para este mesmo nó.

É de grande importância, ressaltar que os dados respectivos ao tempo de execução e eficiência, foram gerados sobre uma matriz 600x600.

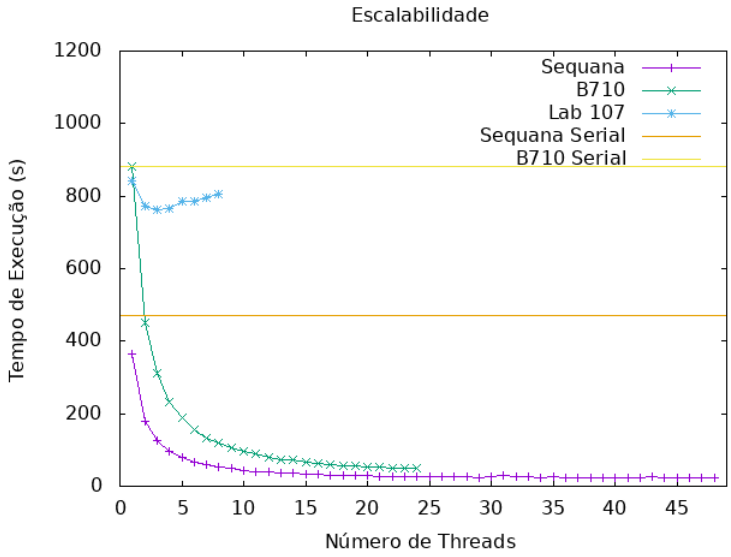
Partindo de um condição de contorno dada pela Figura 6, obtivemos os resultados dados pelas Figuras 15, 16 e 17, onde, respectivamente, suas tolerâncias quanto à flutuação

Figura 12 – Escalabilidade do programa, após correção.



Fonte: Autor. Dados gerados no Sequana/LNCC.

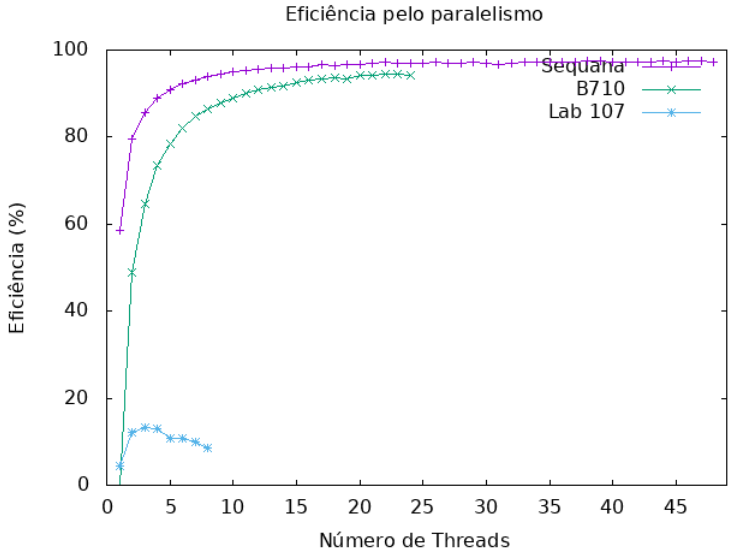
Figura 13 – Benchmark dos computadores do Laboratório 107(UFF) dos nós Sequana e B710(LNCC).



Fonte: Autor.

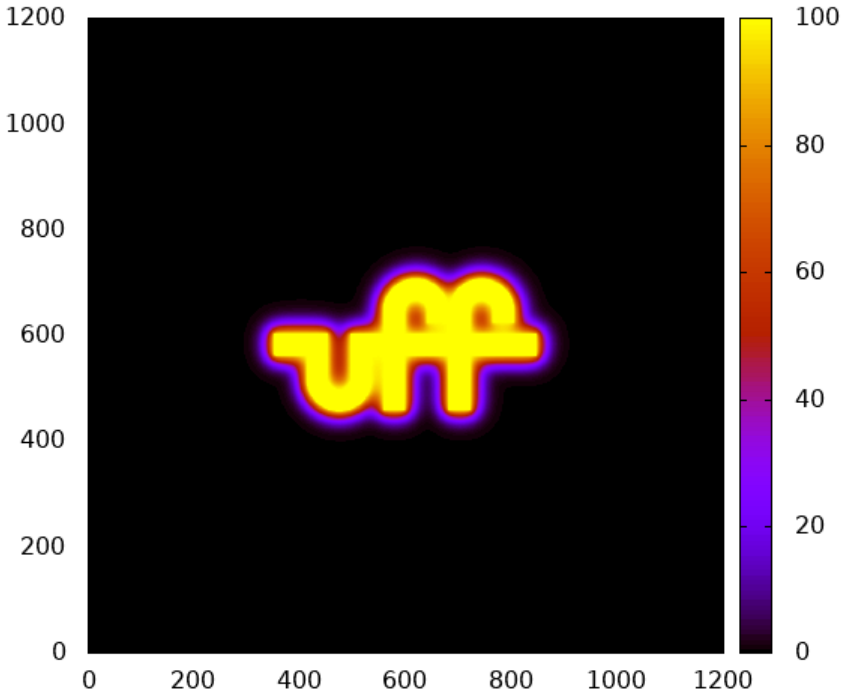
eram de $10^{-}1V$, $10^{-}3V$ e $10^{-}5V$. Para esses resultados a matriz utilizada foi de dimensão 1200x1200.

Figura 14 – Eficiência tendo como referência o tempo serial do nó B710.



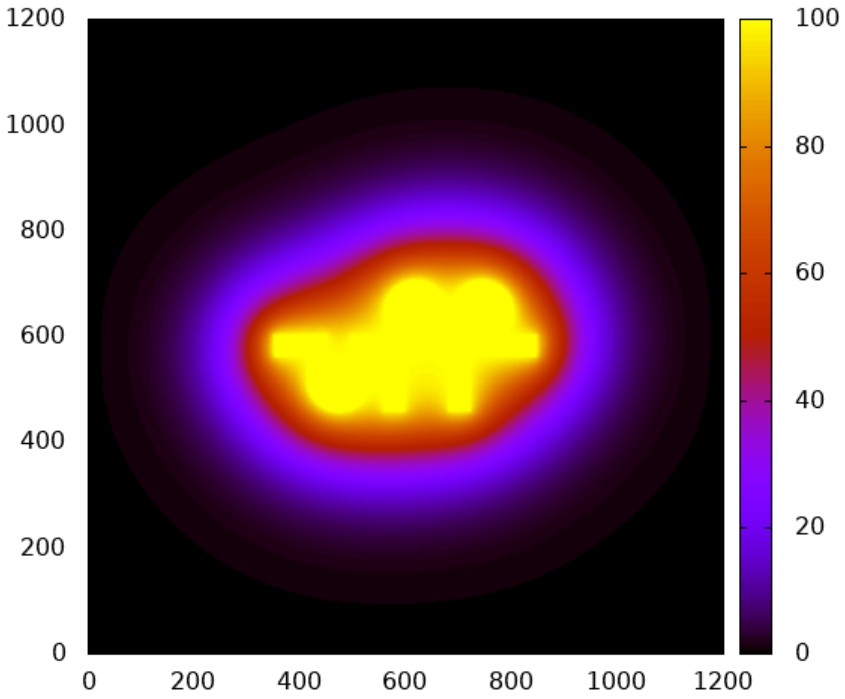
Fonte: Autor.

Figura 15 – Resultado com tolerância de $10^{-}1V$.



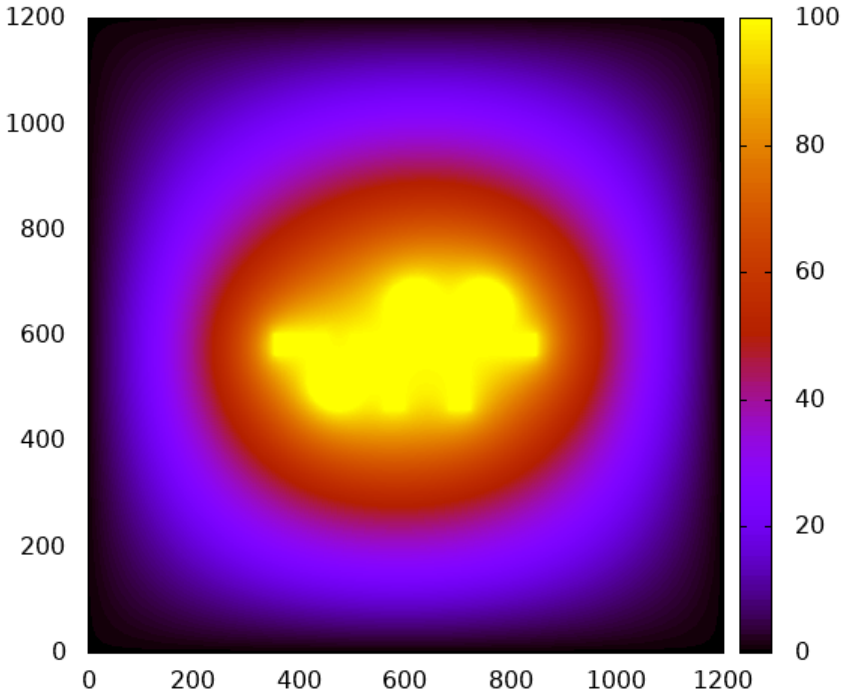
Fonte: Autor.

Figura 16 – Resultado com tolerância de $10^{-}3V$.



Fonte: Autor.

Figura 17 – Resultado com tolerância de $10^{-}5V$.



Fonte: Autor.

9 Conclusão

Apesar da anomalia, quanto a escalabilidade, dos processadores da UFF e também percebido no processador de quem elabora este trabalho, é notável a agilidade que pode ser obtida através da paralelização de um código, quando este assim o permite.

Construir um programa de forma organizada com planejamento, execução, verificação e correção, conhecido como ciclo PDCA, uma das ferramentas fundamentais da Gestão de Projetos.

Portanto o desenvolver de um método numérico deixa de ser um simples ato de eminência e passa a ser algo mais elaborado, que requer muita reflexão e análise a cerca do problema.

Referências

LNCC. *Supercomputador Santos Dumont (SDumont)*. Disponível em: <https://sdumont.lncc.br/support_manual.php?pg=support#1.1>. Acesso em: 10 set. 2021. Citado na página 13.

OPENMP. *About Us*. 2012. Disponível em: <https://www.openmp.org/about/about-us/>. Acesso em: 10 set. 2021. Citado na página 8.

OPENMP. *About Us*. 2012. Disponível em: <https://www.openmp.org/resources/openmp-compilers-tools/>. Acesso em: 10 set. 2021. Citado na página 8.

UFRGS. *Diferenças finitas*. 2020. Disponível em: <https://ciencia.ufla.br/reportagens/mercado/655-ufla-e-pioneira-em-laboratorio-remoto-para-ensino-de-fisica>. Acesso em: 2 set. 2021. Citado 2 vezes nas páginas 6 e 7.