UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE COMPUTAÇÃO

Programação Paralela e Distribuída

Projeto Final

Professor: Hermes Senger

Aluno: Guilherme de Souza Santiago, 790847

Aluno: Ivan Duarte Calvo, 790739

São Carlos, SP, 2023

1 Introdução

O código a ser paralelizado nesse projeto simula a propagação de uma onda acústica em um domínio 2D. A simulação funciona através da seguinte equação:

$$\frac{\partial^2 u}{\Delta t^2} = v^2(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2})$$

E sua forma discretizada:

$$p_{i,j}^{n+1} = 2p_{i,j}^{n} - p_{i,j}^{n-1} + \Delta t^2 \times v^2(\frac{y_{i,j}^{n} - 2p_{i,j}^{n} + p_{i-1,j}^{n}}{\Delta x^2} + \frac{p_{i,j+1}^{n} - 2p_{i,j}^{n} + y_{i,j-1}^{n}}{\Delta y^2})$$

O código recebe como entrada três parâmetros, são eles: N1, N2 e TIME. N1 e N2 representam as dimensões da matriz calculada, e TIME representa o tempo em milissegundos em que a onda foi propagada.

A partir da simulação, é gerada uma matriz que pode ser utilizada para a representação visual do comportamento da propagação da onda acústica. Essa representação visual pode ser vista na imagem a seguir, com uma matriz de tamanho 1000x1000 e com a onda acústica sendo propagada por 10000ms:

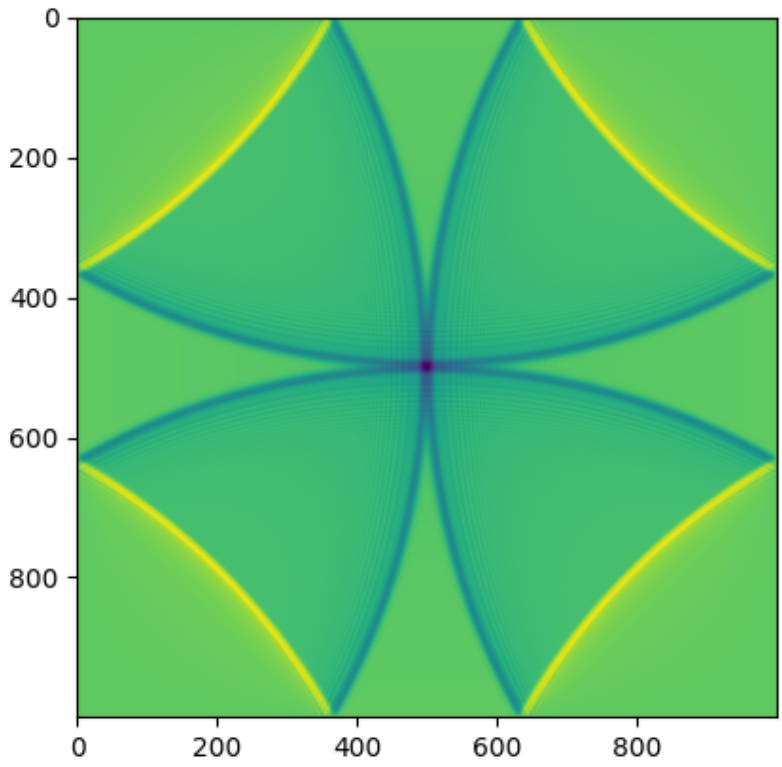

Figura 1: Representação visual da matriz calculada

2 Implementação

A forma escolhida para a implementação foi através da biblioteca *openMP*, devido ao seu desempenho satisfatório, simplicidade de implementação e também sua alta portabilidade.

O trecho paralelizado de código foi o seguinte:

```
// wavefield modeling
for(int n = 0; n < iterations; n++) {
    #pragma omp parallel for collapse(2)
    for(int i = HALF_LENGTH; i < rows - HALF_LENGTH; i++) {
        for(int j = HALF_LENGTH; j < cols - HALF_LENGTH; j++) {
            // index of the current point in the grid
            int current = i * cols + j;

            //neighbors in the horizontal direction
```

```
10              float value = (prev_base[current + 1] - 2.0 * prev_base[
    current] + prev_base[current - 1]) / dxSquared;
11
12              //neighbors in the vertical direction
13              value += (prev_base[current + cols] - 2.0 * prev_base[
    current] + prev_base[current - cols]) / dySquared;
14
15              value *= dtSquared * vel_base[current];
16
17              next_base[current] = 2.0 * prev_base[current] - next_base[
    current] + value;
18          }
19      }
20
21      // swap arrays for next iteration
22      swap = next_base;
23      next_base = prev_base;
24      prev_base = swap;
25  }
```

A aplicação da paralelização foi realizada na linha 3, onde foi utilizada a cláusula *collapse(2)* para abranger ambos os laços *for* que percorrem as linhas e colunas da matriz, pois no momento da compilação do código com *gcc -fopenmp* o compilador irá realizar a melhor implementação convertendo esses dois laços em um único laço mais otimizado e realizando uma separação de execução para cada *thread*.

3 Hardware

O código foi colocado para execução no *cluster* da UFSCar, e através do comando *lscpu* é possível analisar qual a CPU utilizada na execução:

```
1 Architecture:          x86_64
2 CPU op-mode(s):        32-bit, 64-bit
3 Byte Order:            Little Endian
4 CPU(s):                128
5 On-line CPU(s) list:   0-127
6 Thread(s) per core:    2
7 Core(s) per socket:    32
```

```
Socket(s):           2
NUMA node(s):        8
Vendor ID:           AuthenticAMD
CPU family:          23
Model:               49
Model name:          AMD EPYC 7452 32-Core Processor
Stepping:            0
CPU MHz:             2345.560
BogoMIPS:            4691.12
Virtualization:      AMD-V
L1d cache:           32K
L1i cache:           32K
L2 cache:            512K
L3 cache:            16384K
NUMA node0 CPU(s):   0-7,64-71
NUMA node1 CPU(s):   8-15,72-79
NUMA node2 CPU(s):   16-23,80-87
NUMA node3 CPU(s):   24-31,88-95
NUMA node4 CPU(s):   32-39,96-103
NUMA node5 CPU(s):   40-47,104-111
NUMA node6 CPU(s):   48-55,112-119
NUMA node7 CPU(s):   56-63,120-127
```

4 Resultados

Para calcular os valores para *Speedup* e eficiência foram utilizadas as seguintes equações:

$$Speedup_n = \frac{T_n}{T_{seq}}$$

$$Eficiencia_n = \frac{Speedup_n}{n\_threads}$$

O algoritmo paralelizado foi submetido para execução em configurações de 1, 2, 5, 10, 20 e 40 *threads* e os parâmetros passados para tamanho da matriz e tempo de propagação foram: 5000x5000 e 10000ms respectivamente. Os resultados obtidos com os tempos de execução, *Speedup* e Eficiência estão contidos na tabela e imagens a seguir:

Tabela 1: Métricas

nº de *threads*	T Sequencial	T Paralelizado	*Speedup*	Eficiência
1	390.226868	395.520006	0.986617	0.986617
2	-	198.003766	1.970805	0.985403
5	-	80.620912	4.840269	0.968054
10	-	41.644891	9.370342	0.937034
20	-	32.803918	11.895740	0.594787
40	-	18.348641	21.267344	0.531684

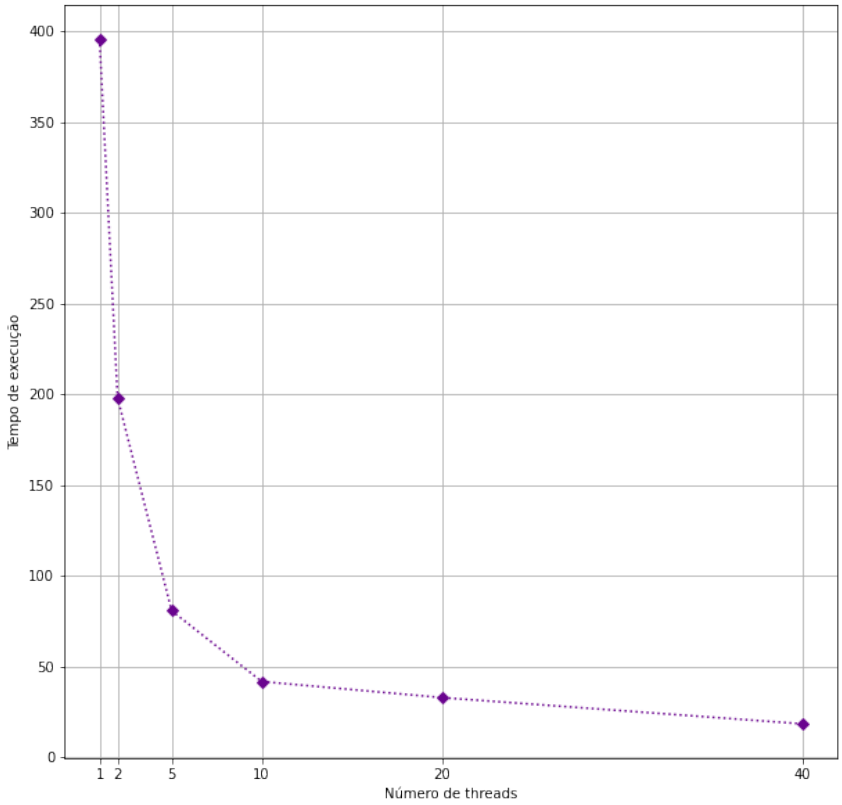

Figura 2: Gráfico com os tempos de execução

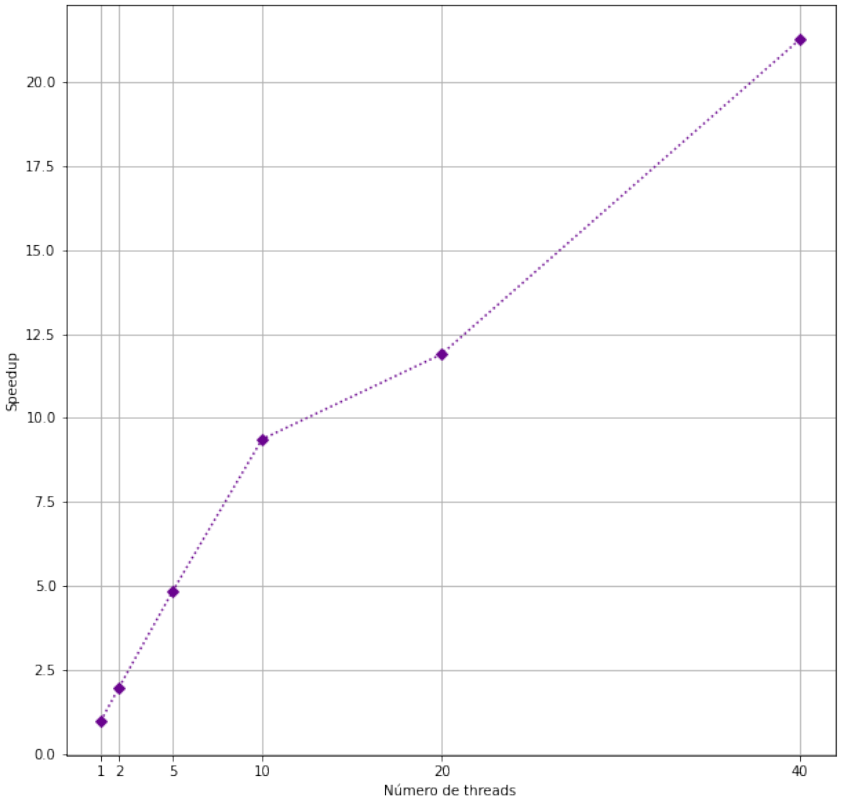

Figura 3: Gráfico com os valores de *speedup*

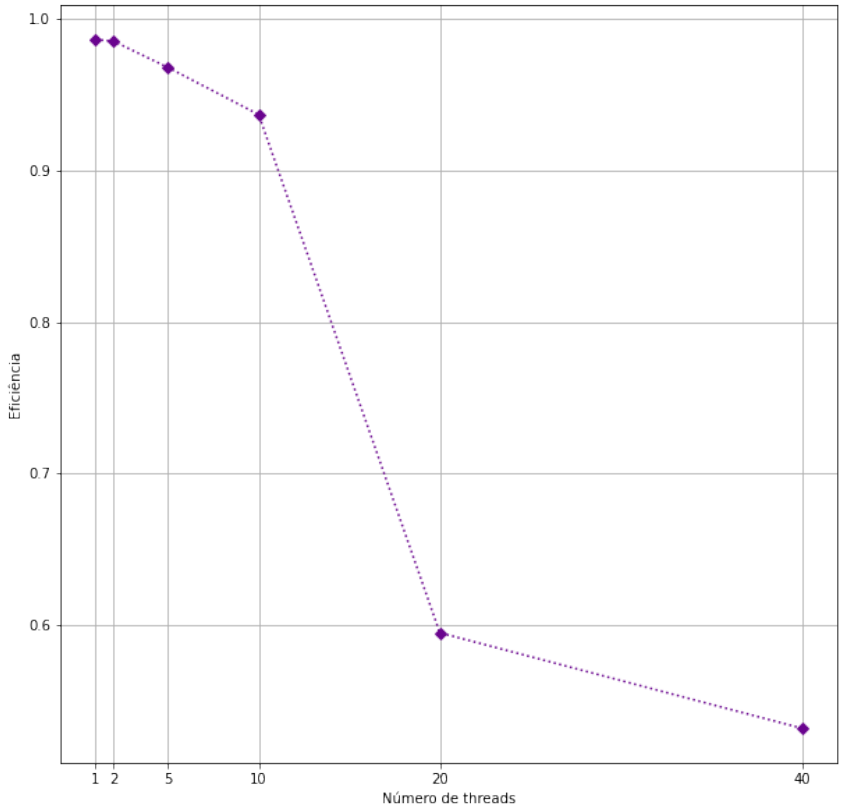

Figura 4: Gráfico com os valores de eficiência

5 Conclusão

Como esperado, a implementação da paralelização com a biblioteca *OpenMP* se mostrou muito vantajosa, possuindo uma implementação de pouca complexidade e retornando um desempenho aceitável. Com a utilização de 20 e 40 *threads* é possível notar uma queda na eficiência, contudo, os valores ainda se encaixam em uma faixa completamente aceitável.

Uma opção de possível melhoria no código seria a paralelização, através do *OpenMP*, das criações, inicializações e arquivamento das matrizes que são usadas no processamento, contudo, essas paralelizações teriam um ganho de desempenho ínfimo, pois o maior peso de trabalho presente no código está concentrado nos cálculos das ondas. Com isso, pode-se denotar que com poucas linhas de código a biblioteca *OpenMP* trouxe o máximo de proveito e desempenho e com a possibilidade de ser exportado para outros sistemas com poucas ou até nenhuma alteração no código.

É possível ainda que, com a utilização de outros meios, como *pthreads*, *MPI* ou até

mesmo *CUDA* se alcance valores de *Speedup* e *Eficiência* melhores, entretanto todas essas tecnologias possuem implementações relativamente mais complexas, sendo assim possível concluir que a aplicação de *OpenMP* se mostra uma boa alternativa para se obter um ganho de desempenho no problema proposto.