UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação - ICMC-USP

Fabio Alves Martins Pereira (NUSP 7987435)

Naylor Garcia Bachiega (NUSP 5567669)

Trabalho 1: algoritmo sequencial e paralelo utilizando o método de Jacobi-Richardson

SÃO CARLOS

2015

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

A ciência da computação é uma área abrangente envolvendo vários aspectos nas mais variadas esferas do conhecimento. Ainda segundo Brookshear (2013):

> *A ciência da computação é uma disciplina que busca construir uma base científica para tópicos como projeto e programação de computadores, processamento de informação, soluções algorítmicas de problemas e o próprio processamento algorítmico.*

Dentro dessa área de conhecimento existem os algoritmos, os quais são importantes para resolver problemas ou criar soluções para os mais diversos paradigmas computacionais. Eles são um conjunto de passos que definem como uma ou mais tarefas serão realizadas (BROOKSHEAR, 2013).

Como o surgimento dos computadores e os algoritmos, o tempo de processamento das tarefas foi reduzido substancialmente em processadores de um núcleo. Com o acoplamento de mais núcleos no processador, algoritmos que dividem suas tarefas entre esses núcleos, tendem a ter um melhor desempenho, de acordo com o tipo de dados e sua possibilidade de paralelização (YANO, 2010).

Sendo assim, é importante que o algoritmo desenvolvido avalie todas as possibilidades de paralelização, para extrair um melhor tempo de execução. Atualmente, existem diversas linguagens de programação e bibliotecas que fornecem ferramentas para paralelização, entre elas pode-se citar *Pthreads*[1], *OpenMP* [2]e *MPI* [3](SATO, GUARDIA, 2013)

Além da paralelização de processos na *CPU*, é possível enviar trabalho para a *GPU* através de *CUDA*, que é uma plataforma de computação paralela e um modelo de programação desenvolvido pela NVIDIA. Ela permite aumentos significativos de desempenho computacional ao aproveitar a potência da Unidade de Processamento Gráfico (GPU) (NVIDIA, 2015).

Conforme pode ser observado na Figura 1, há um aumento significativo no tempo

[1] *Pthreads* são definidos como um conjunto de tipos de linguagem de programação C e chamadas de procedimento (LLNL, 2015).

[2] *OpenMP* é um conjunto de diretivas do compilador e bibliotecas chamadas através de rotinas para expressar o paralelismo de memória compartilhada (OPENMP, 2015)

[3] *MPI* é uma *API* padronizada normalmente utilizada para computação paralela e/ou distribuída.

de execução para algoritmos que utilizam processamento paralelo, tanto para trabalhos enviados para a *CPU* quanto para a *GPU*. Porém na *GPU*, o tempo de resposta foi menor, pelo desempenho da unidade.

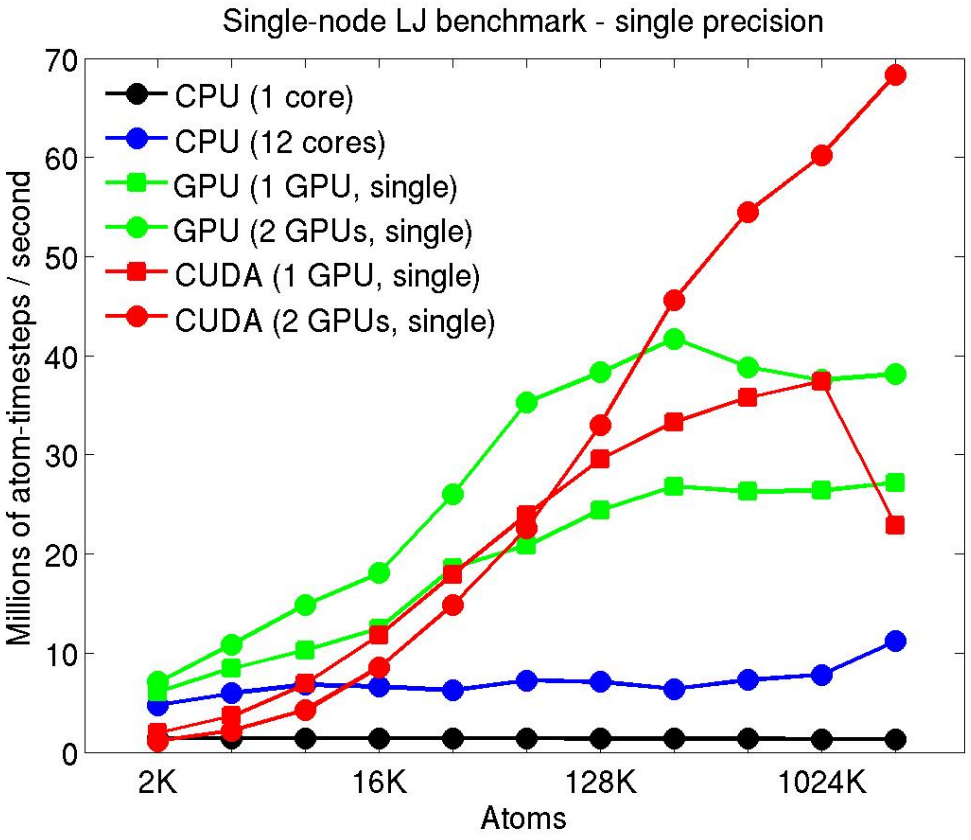

Figura 1 – Benchmark em um único computador (LAMMPS, 2015).

1.2 OBJETIVOS

Tendo em vista os benefícios da paralelização de algoritmos, esse trabalho mostra o desenvolvimento de algoritmos sequencias e paralelos para o processamento de matrizes lineares, utilizando o método de Jacobi-Richarson. Após o desenvolvimento, os resultados serão demonstrados através de gráficos e tabelas.

2 MÉTODO DE JACOBI-RICHARSON

Métodos numéricos para solução de sistemas de equações lineares são divididos principalmente em dois grupos (YONA, 2010):

• Métodos Exatos: são aqueles que forneceriam a solução exata, se não fossem os erros de arredondamento, com um número finito de operações.

• Métodos Iterativos: são aqueles que permitem obter a solução de um sistema com uma dada precisão através de um processo infinito convergente e que possui um erro de truncamento.

O método iterativo de Jacobi-Richardson se baseia inicialmente numa verificação de convergência, ou seja, verifica-se se a matriz é estritamente diagonalmente dominante. Se ela atender o critério de convergência, define-se a margem de erro máximo (truncamento) que o resultado terá. Assim, o método inicia-se através da solução inicial (no caso de um sistema linear, zero é uma das soluções possíveis), em seguida, calcula-se o resultado e ocorre a substituição no vetor resultado na próxima iteração, e assim em diante até que se tenha a solução que atenda a margem de erro (YONA, 2010).

2.2 DESCRIÇÃO DO MÉTODO

Dado um sistema quadrado de equações lineares n: $Ax = b$, onde:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Então, A pode ser decomposto numa diagonal D, e o restante de R:

$$A = D + R \qquad \text{where} \qquad D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \text{ and } R = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}.$$

A solução é, então, obtida iterativamente:

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - R\mathbf{x}^{(k)}),$$

Onde $\mathbf{x}^{(k)}$ é a aproximação de ordem *k* ou iteração de $\mathbf{x}$ e $\mathbf{x}^{(k+1)}$ é o próximo ou a iteração *k* + 1 de $\mathbf{x}$. A fórmula base é:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n.$$

2.3 ALGORITMO

Para o trabalho, foi utilizado o algoritmo básico, conforme pode ser visualizado:

Inicialmente, é estimado um valor inicial para $x^{(0)}$ para a solução $k = 0$

enquanto erro não é alcançado

para i := 1 até n faça

$\sigma = 0$

para j := 1 até n faça

se j ≠ i então

$\sigma = \sigma + a_{ij} x_j^{(k)}$

fim se

fim (j-loop)

$x_i^{(k+1)} = \frac{(b_i - \sigma)}{a_{ii}}$

fim (i-loop)

$k = k + 1$

repete (enquanto erro não é alcançado)

3 PARALELISMO

Nesse capítulo são expostas as ferramentas para paralelização utilizadas nesse trabalho.

3.2 PTHREADS

Pthread é uma maneira simples e eficaz de criar uma aplicação paralelizada. Quando uma *thread* é criada usando **pthread_create**, a *thread* original e a nova compartilham da mesma base de código e a mesma memória - é como fazer duas chamadas de função ao mesmo tempo (TIM, 2010).

Todos os programas em C usando *pthreads* precisam incluir o arquivo de *header* pthread.h (#include <pthread.h>). No sistema operacional Ubuntu Desktop 15.04 é necessário instalar um pacote através do *apt-get* (ferramenta de instalação e atualização de pacotes):

- sudo apt-get install build-essential
- sudo apt-get install libpthread-stubs0-dev

3.2.1 FUNCIONAMENTO

Há quatro etapas para a criação de um programa básico utilizando Pthreads (TIM, 2010):

- Definir a variável thread de referência: a variável do tipo pthread_t é uma forma de referenciar *threads*. É preciso haver uma variável pthread_t na existência de cada segmento que está sendo criado. Algo como pthread_t thread0.

- Criar um ponto de entrada para a *thread*: ao criar a *thread* usando *pthreads*, é necessário apontá-la para uma função para ela iniciar a execução. A função deve retornar void * e tomar um único argumento void *. Por exemplo, para que a função pegue um argumento inteiro, é necessário passar o endereço do inteiro. Um exemplo de função seria:

void * my_entry_function (void * param);

- Criar a thread: uma vez que a variável pthread_t foi definida e a função de ponto de entrada criada, deve criar o segmento usando pthread_create. Este método tem quatro argumentos: um ponteiro para a variável pthread_t, os atributos extras, um ponteiro para a função a ser chamada e o ponteiro que está sendo passado como argumento para a função. Esta chamada será algo parecido com pthread_create (&thread0, NULL, my_entry_function, e &parameter);

3.3 OPENMP

OpenMP é uma API (*Application Program Interface*), definida em conjunto por um grupo de grandes fornecedores de *hardware* e *software*. Fornece um modelo portátil, escalável para desenvolvedores de aplicações paralelas de memória partilhada. A API suporta C/C ++ e Fortran em uma ampla variedade de arquiteturas.

Programas *OpenMP* realizam paralelismo exclusivamente através da utilização de *threads*. Um *thread* é a menor unidade de processamento que pode ser programada por um sistema operacional. Normalmente, o número de *threads* coincide com o número de processadores/núcleos. *OpenMP* é um modelo de programação explícita (não automático), oferecendo ao programador controle total sobre a paralelização. (BARNEY, 2015)

3.3.1 FUNCIONAMENTO

Uma das coisas úteis sobre *OpenMP* é que ele permite aos usuários a opção de usar o mesmo código-fonte para compiladores normais quanto para compiladores compatíveis com o *OpenMP*. Isto é possível, utilizando suas diretivas *OpenMP* e comandos ocultos para compiladores normais.

Na Figura 2 é demonstrada uma forma muito simples de programa paralelo *multi-threaded*, escrito em C que irá imprimir "*Hello World*", exibindo o número do *thread* em cada nível de processamento. (KIESSLING, 2009)

```
#include 'omp.h'
void main()
{
#pragma omp parallel
  {
    int ID = omp_get_thread_num()
    printf('Hello(%d) ',ID);
    printf('World(%d) \n',ID);
  }
}
```

Figura 2 – Exemplo de algoritmo utilizando *OpenMP* (LAMMPS, 2015).

Conforme pode ser visto, a primeira linha é a biblioteca do *OpenMP*. A região paralela é colocada entre a diretiva **#pragma omp parallel** {...}. A instrução omp_get_thread_num() retorna o ID *thread* para o programa.

O *OpenMP* possui uma série de diretivas para tornar o paralelismo customizável pelo programador. Essa e demais opções podem ser encontradas no site e documentação oficial da ferramenta.

4 DESENVOLVIMENTO E METODOLOGIA

Esse capítulo aborda o desenvolvimento dos algoritmos e a metodologia utilizada. Para esse trabalho, foi utilizada linguagem C e Code::Blocks como interface de desenvolvimento.

4.1 REPOSITÓRIO DO CÓDIGO

Como o trabalho foi desenvolvido em grupo, o GitHub foi utilizado para compartilhar o código e controlar o versionamento. O repositório pode ser acessado pelo link: **https://github.com/naylor/trabalho-prog-conc**

4.2 HARDWARE

A Tabela 1 mostra o hardware utilizado nos experimentos, para os algoritmos sequencial e paralelo.

TABELA 1 – ESPECIFICAÇÃO DO HARDWARE UTILIZADO.

COMPONENTE	MODELO
Placamãe	ASUSTeK COMPUTER INC. S550CA
Sistema Operacional	Linux Ubuntu 14.04.3 LTS
Processador	Intel Core i7 3537U @ 2.00GHz
Memória RAM	8,00GB Dual-Channel DDR3 798MHz
HardDrive	465GB Western Digital WDC WD5000LPVX-80V0TT0 (SATA)

4.2 PROGRAMA: TRABALHO-PROG-CONC

O programa principal foi criado para listar as matrizes disponíveis no diretório “matrizes/” e disponibilizar a opção de escolha para o usuário. O usuário, após escolher a matriz, pode especificar se a execução será sequencial ou paralela. No caso da execução paralela, ainda é possível digitar a quantidade desejada de *thread* (Figura 3).

O sistema foi dividido conforme descrito a seguir:

- **main.c**: faz a inicialização do sistema, carrega o menu e as escolhas do usuário.
- **menu.c**: o menu do usuário.
- **funcao.c**: funções importantes para o sistema, como enviar o resultados para a tela e imprimir os resultados no arquivo.
- **matriz.c**: checar as matrizes disponíveis, carregar a matriz, limpar a memória após execução e checar os critérios de parada.
- **p_pthread.c**: faz as chamadas das threads para execução do algoritmo paralelo, utilizando Pthreads.
- **p_omp.c**: faz as chamadas das threads para execução do algoritmo paralelo, utilizando *OpenMP*.
- **sequencial.c**: instruções do algoritmo sequencial.
- **timer.c**: função para imprimir o tempo de execução dos algoritmos.

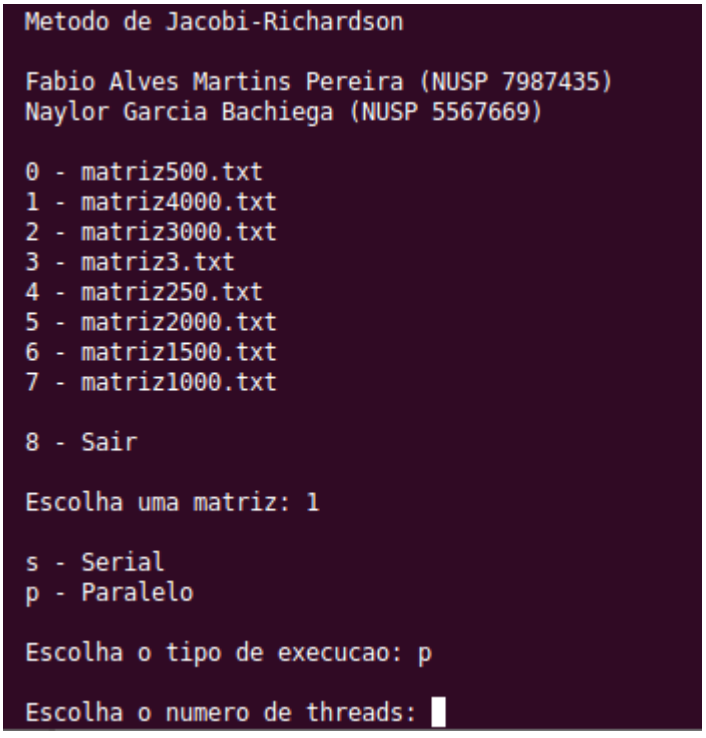

Figura 3 – Tela inicial do programa.

Além da opção por menu, o sistema permite a execução por linha de comando, conforme pode ser observado na Figura 4. Deve-se informar a matriz, o algoritmo e a quantidade de threads, caso paralelo.

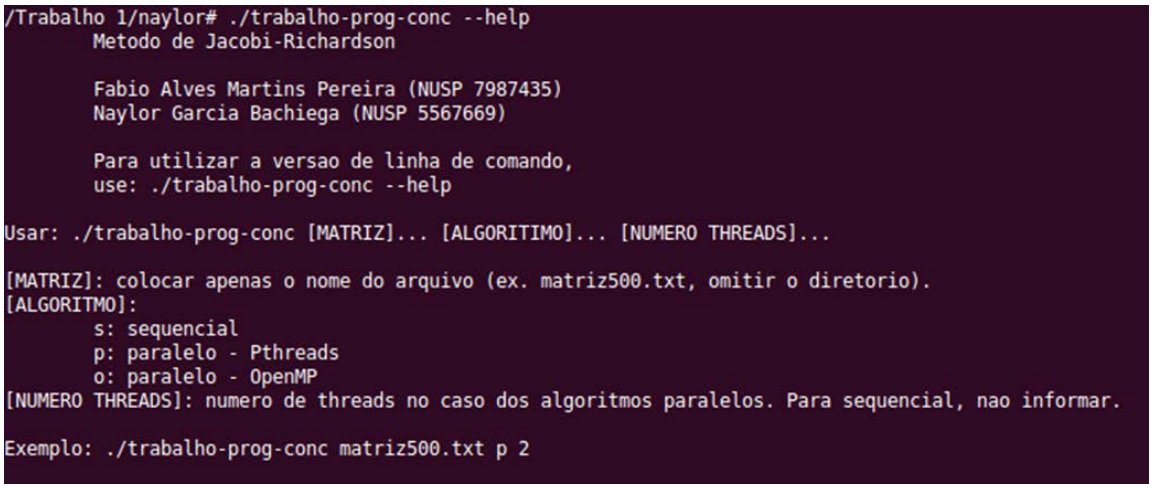

Figura 4 – Ajuda do programa.

4.3 README.MD

O arquivo README.md contém informação sobre outros arquivos de um projeto ou sistema, como por exemplo autores, procedimentos de instalação, agradecimentos, *bugs*, entre outros.

Abaixo segue o arquivo referente ao sistema desenvolvimento para esse trabalho.

```
trabalho-prog-conc
==================

Exemplo de como utilizar o programa.

### Dependências
1. Necessária instalação da libpthread:

sudo apt-get install build-essential
sudo apt-get install libpthread-stubs0-dev

### Instalação

1. Faça o clone deste projeto:
      git clone https://github.com/naylor/trabalho-prog-conc.git`

2. Entre na pasta do projeto

3. Rode o comando "make"


### Executando a aplicação

1. Utilizando os menus do programa
  usar: ./trabalho-prog-conc

2. Executando pelo terminal
  usar: ./trabalho-prog-conc --help
  ou
```

```
  usar: ./trabalho-prog-conc [MATRIZ]... [ALGORITIMO]... [NUMERO THREADS]...

3. Os resultados são gravados na pasta: resultados

4. Matrizes disponíveis na pasta: matrizes
  Obs.: Arquivos de matrizes são carregados automaticamente,
     assim, mais arquivos podem ser adicionados.
```

4.4 MAKEFILE

O *make* é utilitário *Unix* que é projetado para iniciar a execução de um *makefile*. Um *makefile* é um arquivo especial, contendo comandos *shell*, geralmente instruções necessárias para a compilação do programação. Para o trabalho, o *Makefile* foi configurado conforme a seguir:

```
# MAKEFILE #

#INFORMANDO O COMPILADOR,
#DIRETÓRIOS E O
#NOME DO PROGRAMA
CC=gcc
G=g++
SRCDIR=src/
SRCEXT=c
OBJEXT=o
PROG=trabalho-prog-conc

# FLAGS NECESSARIAS
# PARA COMPILACAO
CFLAGS=-Wall -Wextra
LIB=-lpthread

#------------------------------
# CARREGA AUTOMATICAMENTE OS
# ARQUIVOS .C E .H
#------------------------------
SOURCES=$(wildcard $(SRCDIR)*.c)
HEADERS=$(wildcard $(SRCDIR)*.h)


all: $(PROG)

$(PROG): $(SOURCES:.c=.o)
	$(G) -o $@ $^ $(LIB)

%.o: %.c $(HEADERS)
	$(CC) -g -c $< -o $@

clean:
	rm -f $(SRCDIR)*.o
	rm -f $(PROG)
```

4.5 PARTICIONAMENTO DA MATRIZ

Um algoritmo paralelo é derivado a partir de uma escolha de distribuição de dados. A distribuição de dados deve ser equilibrada, alocar (aproximadamente) o mesmo número de entradas para cada processador; e deverá minimizar a comunicação. A Figura 5 ilustra a distribuição utilizada nesse trabalho, em que a matriz é sujeita a partição em uma dimensão de acordo com o número de *threads* informado.

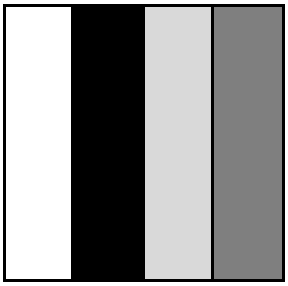

Figura 5 – Distribuição dos dados da matriz.

5 RESULTADOS E DISCUSSÕES

Aqui são apresentados os resultados e discussões com base nos algoritmos sequencial e paralelo.

5.1 ALGORITMO SEQUENCIAL

Inicialmente foram executadas as matrizes de ordem *n x n* utilizando o algoritmo sequencial, obtendo os resultados conforme a Tabela 2:

TABELA 2 – EXECUÇÃO DAS MATRIZES PELO MÉTODO SEQUENCIAL.

ORDEM ***n x n***	**MÉDIA (segundos)**	**DESVIO (segundos)**
250	0,9556787	0,001737338
500	5,2710011	0,002590176
1000	58,2954194	0,973853303
1500	194,5482731	3,243080053
2000	479,9629639	0,15378519
3000	1615,233688	0,165632233
4000	➢ 999999	0

Foram realizadas as médias das dez execuções para cada ordem da matriz (ANEXO A). Não foi possível determinar o tempo de execução da matriz de ordem 4000. Após nove horas de execução, o algoritmo foi interrompido e um valor aproximado foi inserido apenas para efeitos de comparação. Nesse caso, essa matriz foi desconsiderada em alguns gráficos comparativos.

A Figura 6 mostra como o tempo aumenta, consideravelmente, quando a ordem da matriz é acrescida.

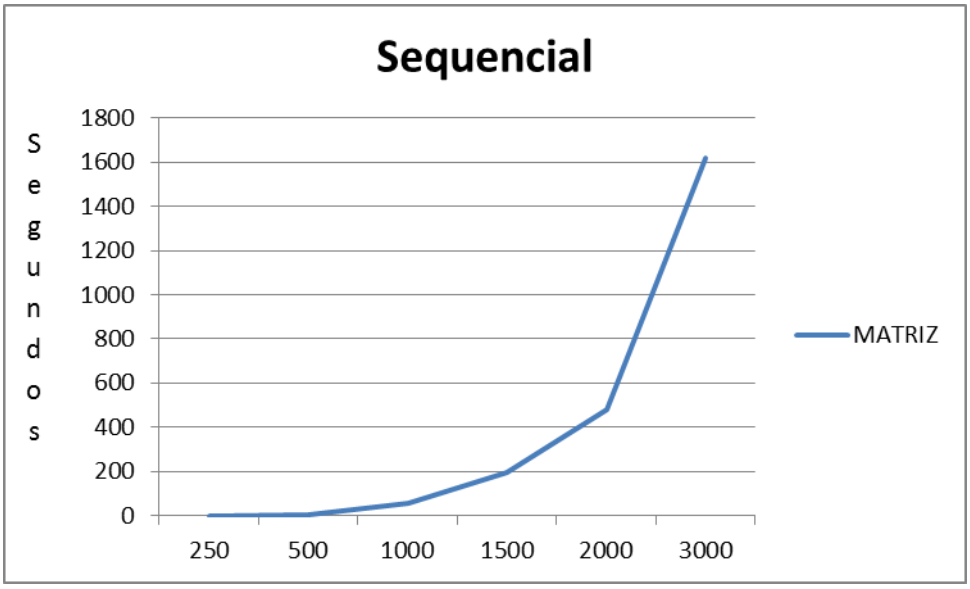

Figura 6 – Execução das matrizes pelo método sequencial.

Apesar de adicionado o desvio padrão no gráfico, não é possível visualizar pela grandeza das diferenças.

5.2 ALGORITMO PARALELO: PTHREADS

Os mesmos testes foram realizados para o algoritmo paralelo com *Pthreads*. Foi observado que, em um determinado momento, aumentar a quantidade de *threads* também aumentava o tempo de execução. Sendo assim, na Tabela 3 são apresentados os melhores resultados para cada ordem de matriz e suas respectivas quantidades de *threads* para o melhor tempo de execução.

TABELA 3 – EXECUÇÃO DAS MATRIZES POR *PTHREADS*.

ORDEM ***n x n***	**THREADS**	**MÉDIA (segundos)**	**DESVIO (segundos)**
250	5	0,0193803	0,000175189
500	5	0,0569816	0,000645895
1000	5	0,1887689	0,002332534
1500	5	0,3959849	0,004570521
2000	5	0,7129205	0,010868422
3000	7	1,7038281	0,018810684
4000	10	2,9953613	0,021264158

Na Figura 7 são apresentados os resultados com os tempos de execução de cada ordem de matriz e a quantidade de *threads* que proporcionou o melhor resultado.

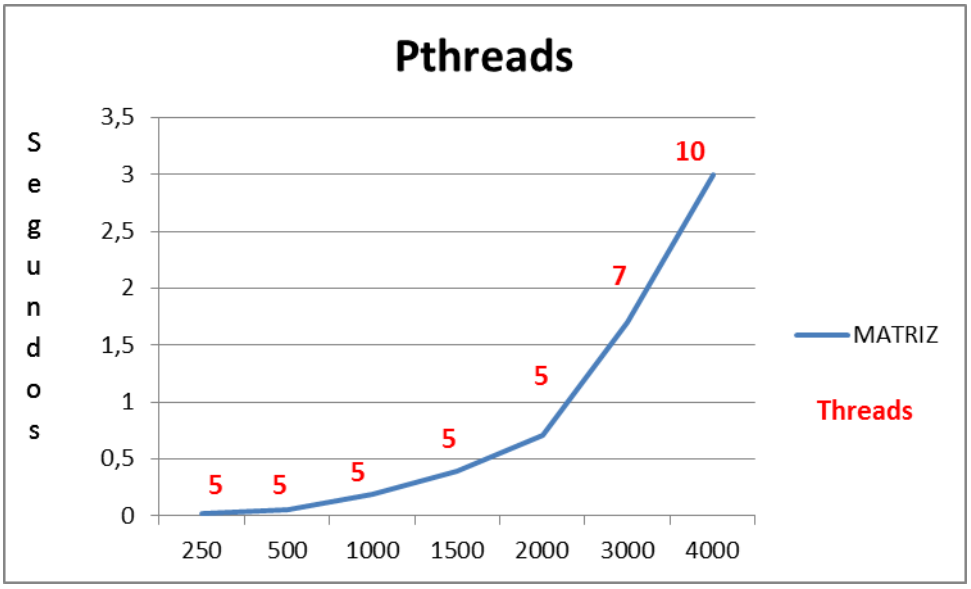

Figura 7 – Execução das matrizes por *Pthreads*.

5.2.1 PROCESSAMENTO DAS MATRIZES

Na Figura 8 é possível observar o comportamento da execução de acordo com a quantidade de *threads* informada ao programa.

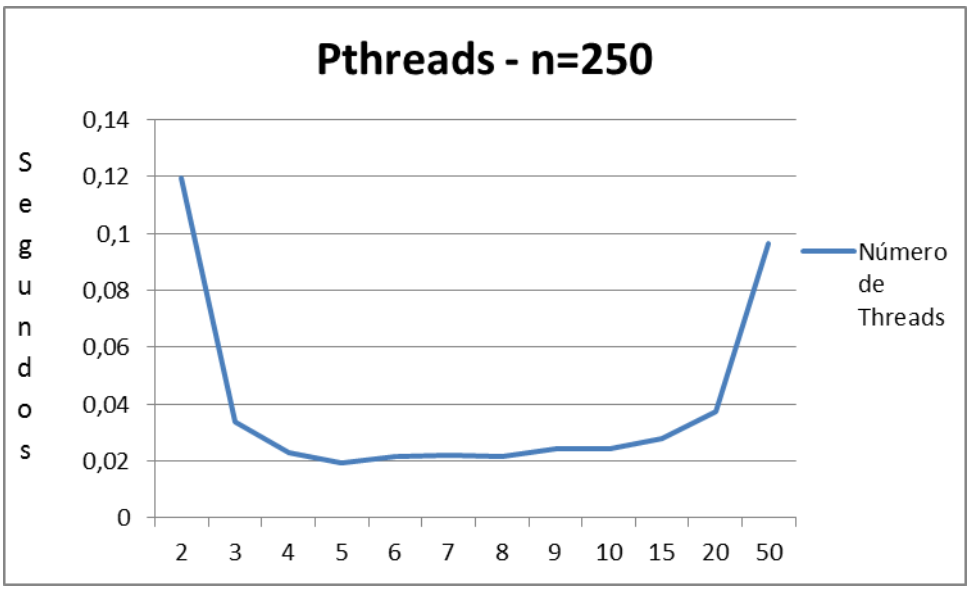

Figura 8 – Execução da matriz de ordem n=250 por *Pthreads*.

Como mencionado, com quatro *threads* o sistema obteve um melhor tempo de

execução. O mesmo pode ser observado nas figuras a seguir.

Execução da matriz de ordem n=500.

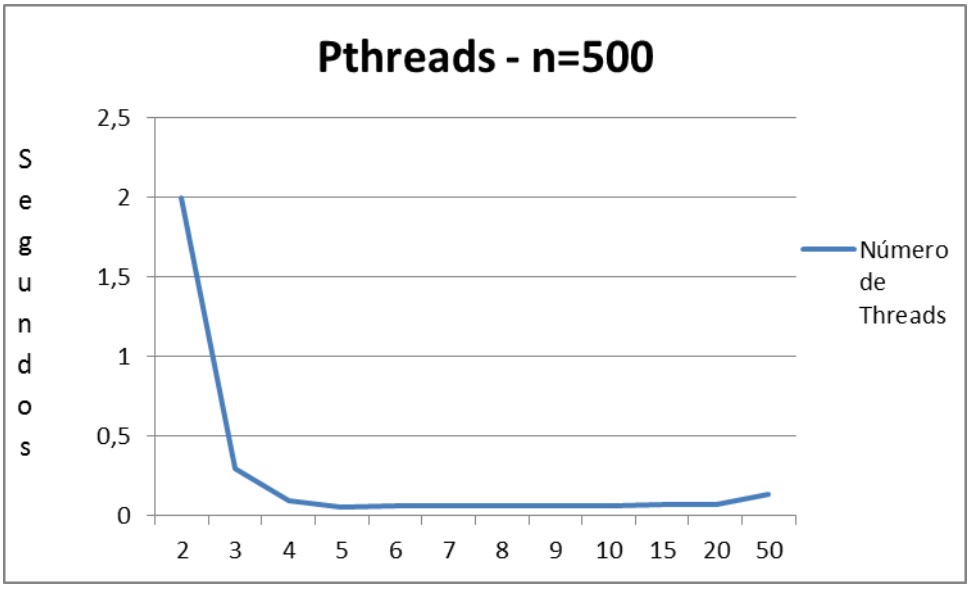

Figura 9 – Execução da matriz de ordem n=500 por *Pthreads*.

Execução da matriz de ordem n=1000.

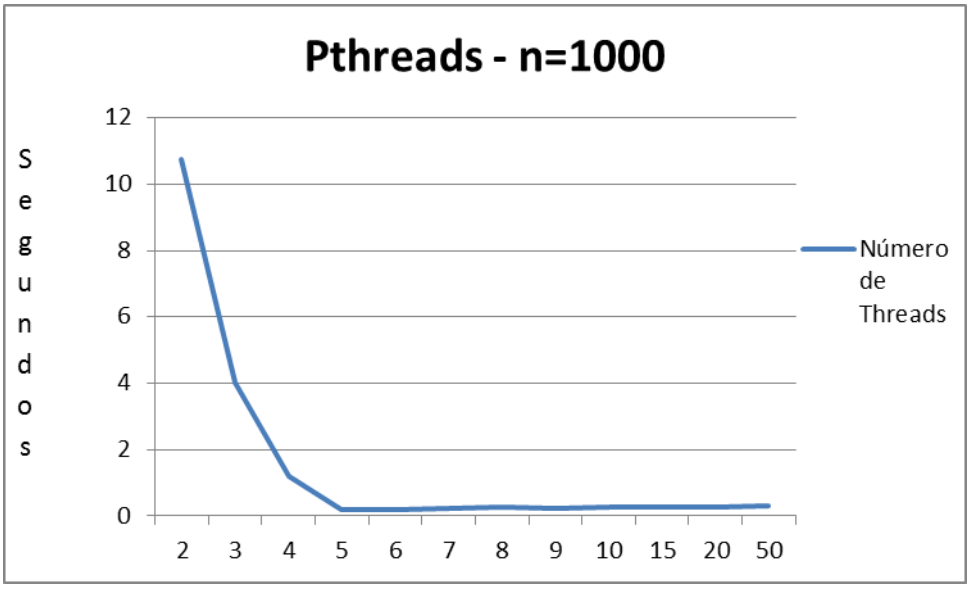

Figura 10 – Execução da matriz de ordem n=1000 por *Pthreads*.

Execução da matriz de ordem n=1500.

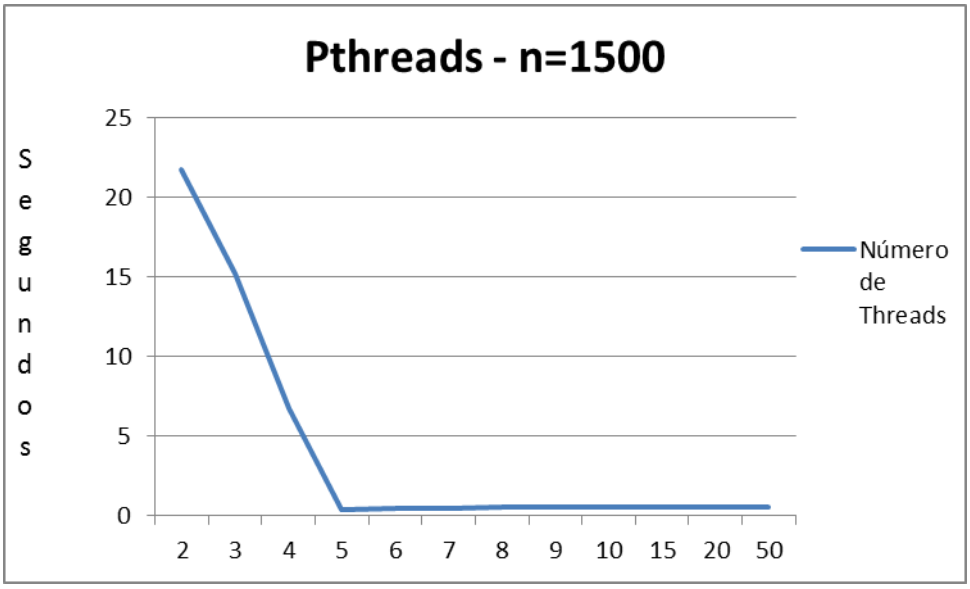

Figura 11 – Execução da matriz de ordem n=1500 por *Pthreads*.

Execução da matriz de ordem n=2000.

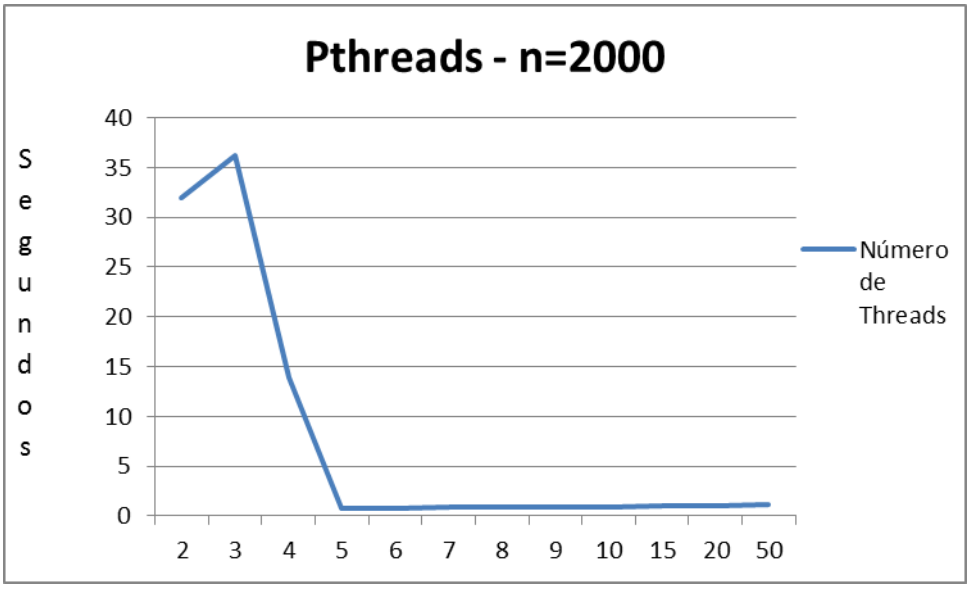

Figura 12 – Execução da matriz de ordem n=2000 por *Pthreads*.

Execução da matriz de ordem n=3000.

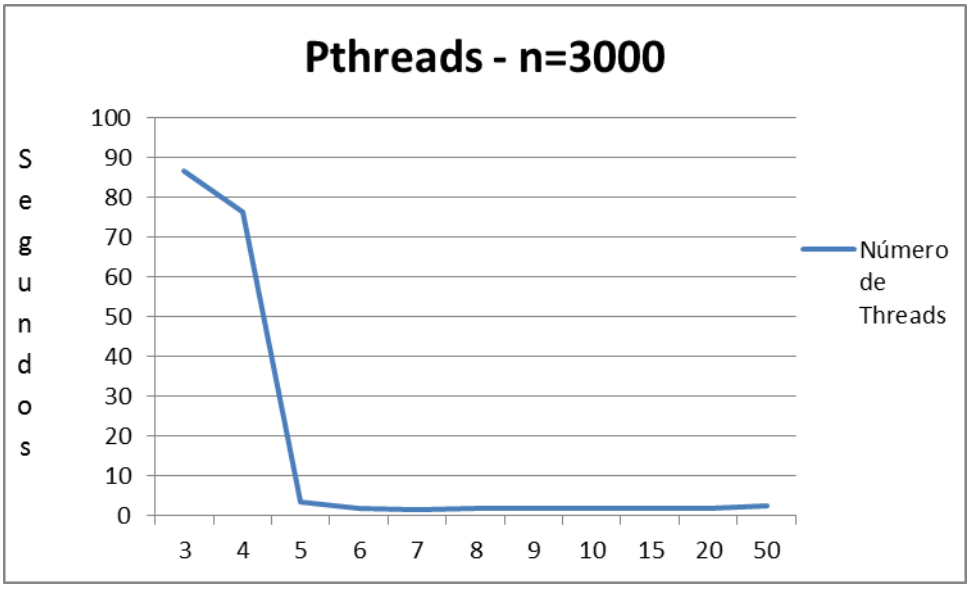

Figura 13 – Execução da matriz de ordem n=3000 por *Pthreads*.

Execução da matriz de ordem n=4000.

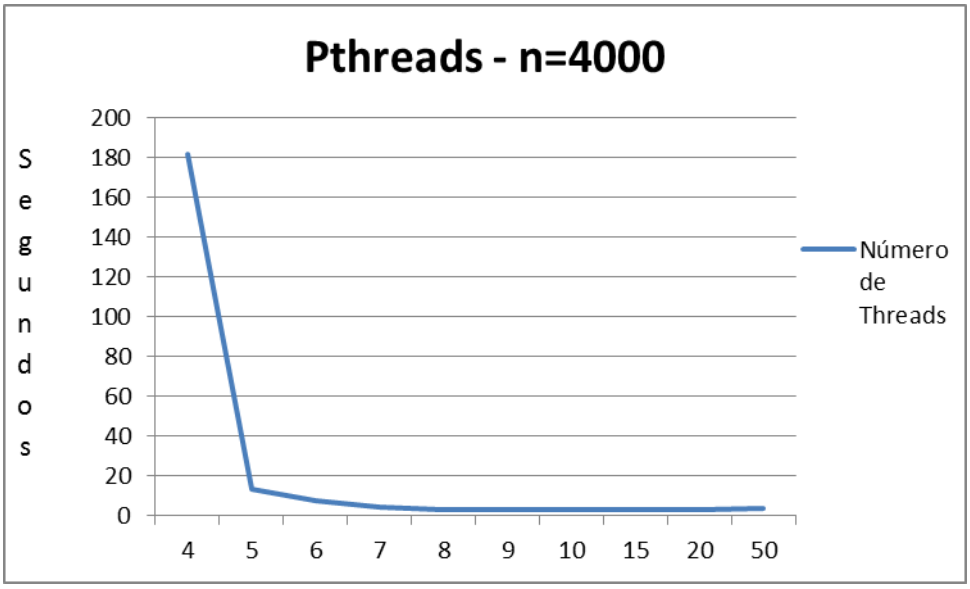

Figura 14 – Execução da matriz de ordem n=4000 por *Pthreads*.

5.3 ALGORITMO PARALELO: *OPENMP*

Aqui são demonstrados os testes com o *OpenMP*. Também foi observado que, em um determinado momento, aumentar a quantidade de *threads* matinha o resultado

estável em relação ao tempo de execução. Sendo assim, na Tabela 3 são apresentados os melhores resultados para cada ordem de matriz e suas respectivas quantidades de *threads* para o melhor tempo de execução.

TABELA 4 – EXECUÇÃO DAS MATRIZES PELO *OPENMP*.

ORDEM ***n x n***	**THREADS**	**MÉDIA (segundos)**	**DESVIO (segundos)**
250	5	0,0243607	3,29274E-05
500	5	0,0826408	7,89174E-05
1000	5	0,3148744	0,000220465
1500	6	0,6800495	0,000296371
2000	2	1,1732628	0,003280427
3000	3	2,6221947	0,001255182
4000	2	4,4473706	0,004068199

Na Figura 7 são apresentados os resultados com os tempos de execução de cada ordem de matriz e a quantidade de *threads* que proporcionou o melhor resultado.

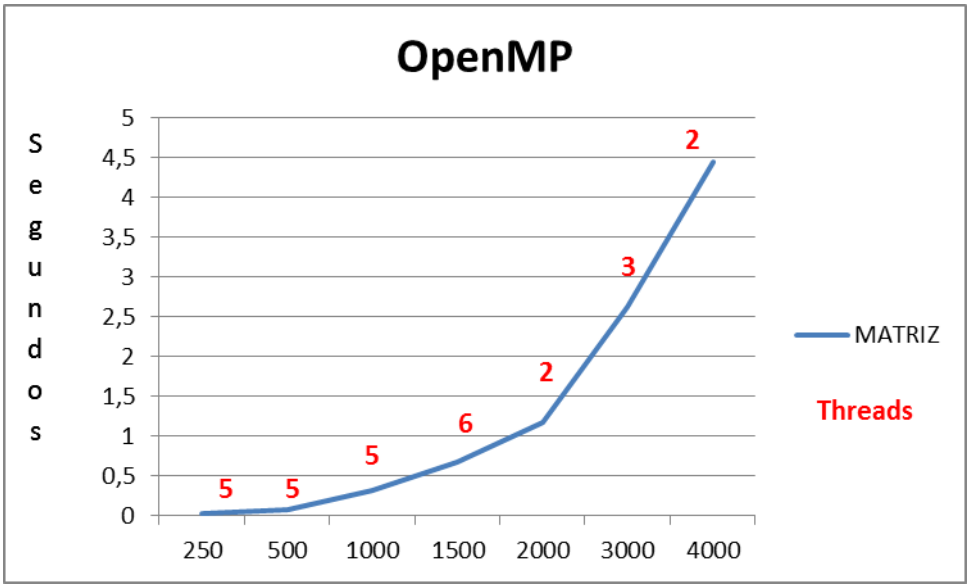

Figura 15 – Execução das matrizes pelo *OpenMP*.

5.3.1 PROCESSAMENTO DAS MATRIZES

Na Figura 8 é possível observar o comportamento da execução de acordo com a quantidade de *threads* informada ao programa.

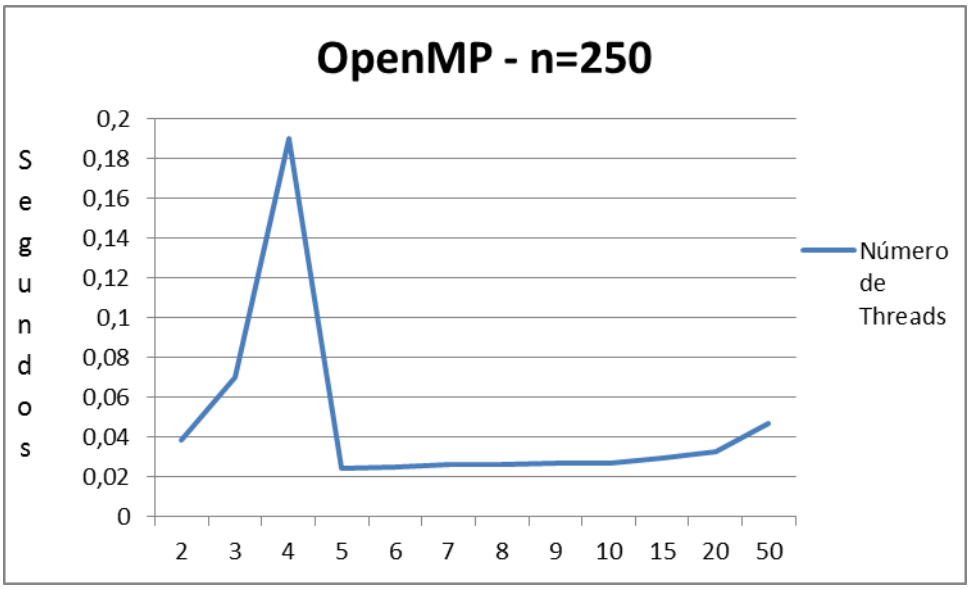

Figura 16 – Execução da matriz de ordem n=250 pelo *OpenMP*.

Como mencionado, com quatro *threads* o sistema obteve um melhor tempo de execução. O mesmo pode ser observado nas figuras a seguir.

Execução da matriz de ordem n=500.

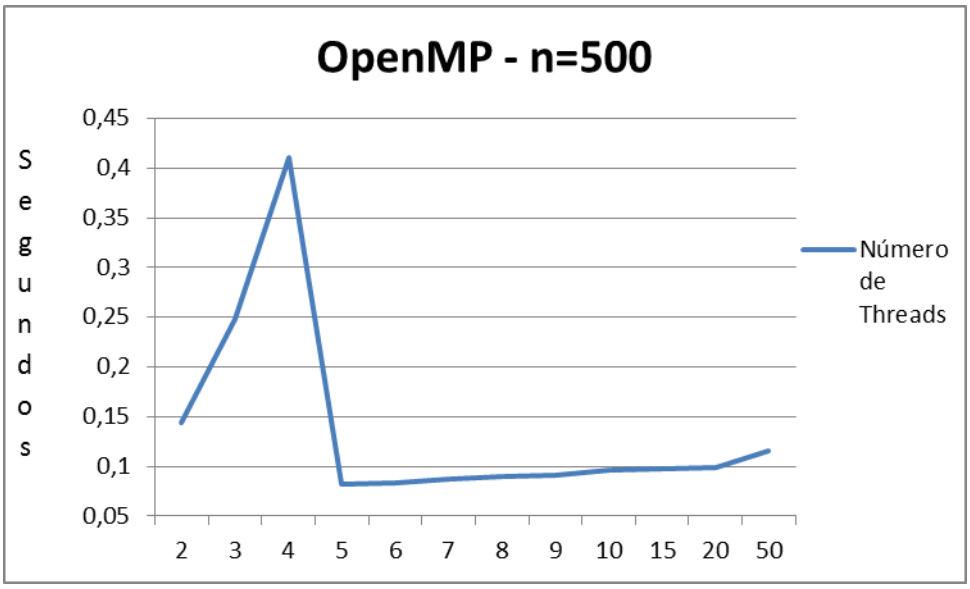

Figura 17 – Execução da matriz de ordem n=500 pelo *OpenMP*.

Execução da matriz de ordem n=1000.

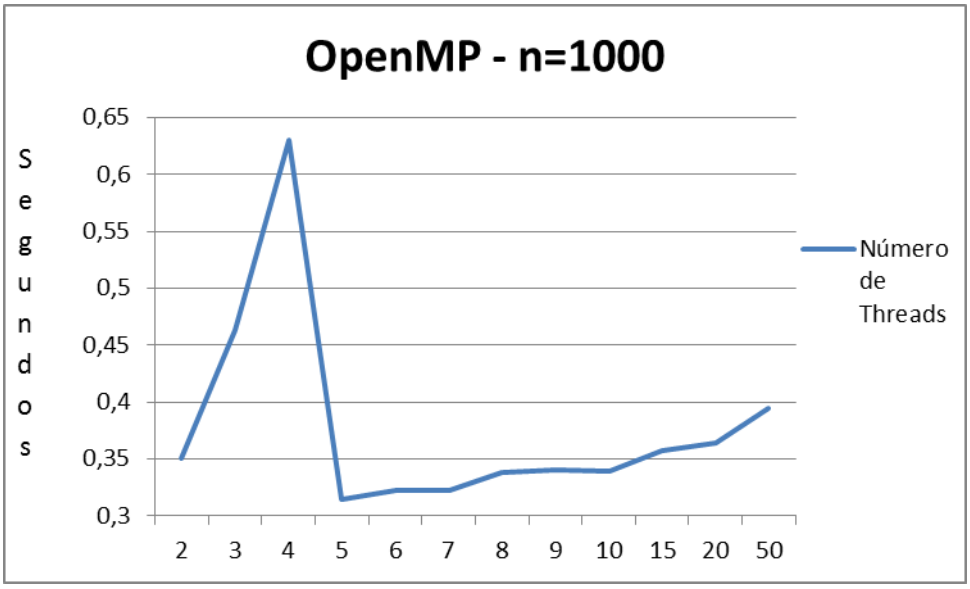

Figura 18 – Execução da matriz de ordem n=1000 pelo *OpenMP*.

Execução da matriz de ordem n=1500.

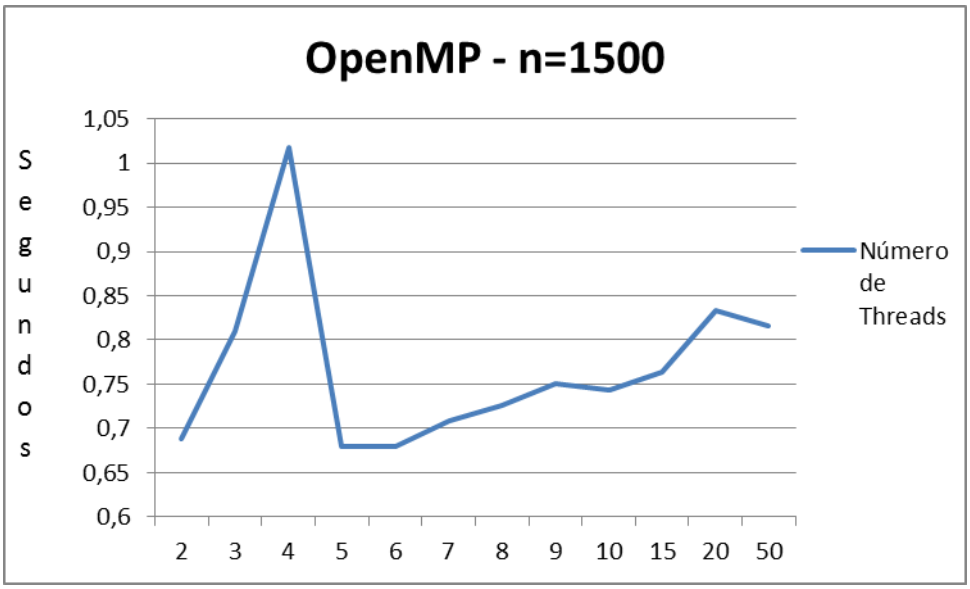

Figura 19 – Execução da matriz de ordem n=1500 pelo *OpenMP*.

Execução da matriz de ordem n=2000.

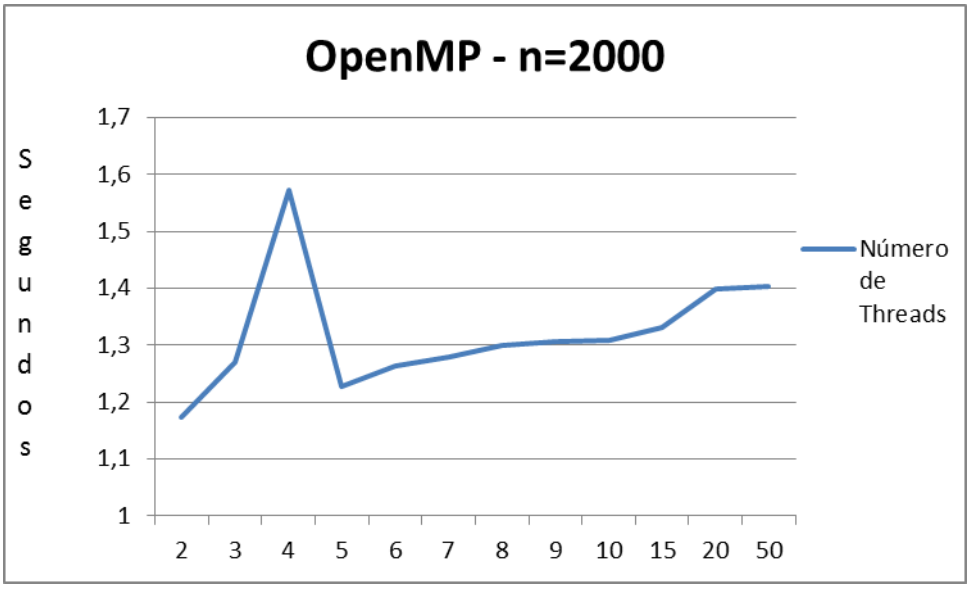

Figura 20 – Execução da matriz de ordem n=2000 pelo *OpenMP*.

Execução da matriz de ordem n=3000.

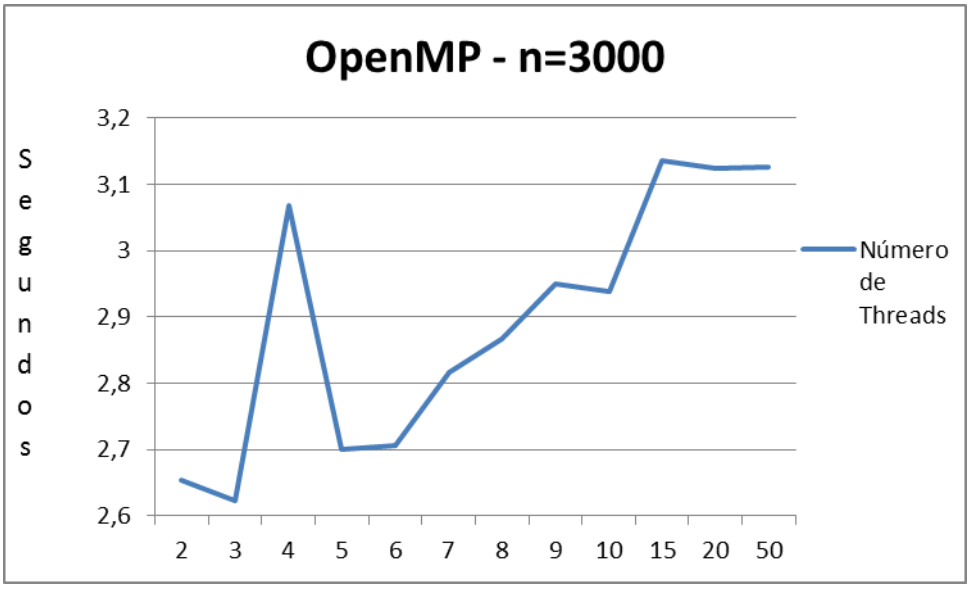

Figura 21 – Execução da matriz de ordem n=3000 pelo *OpenMP*.

Execução da matriz de ordem n=4000.

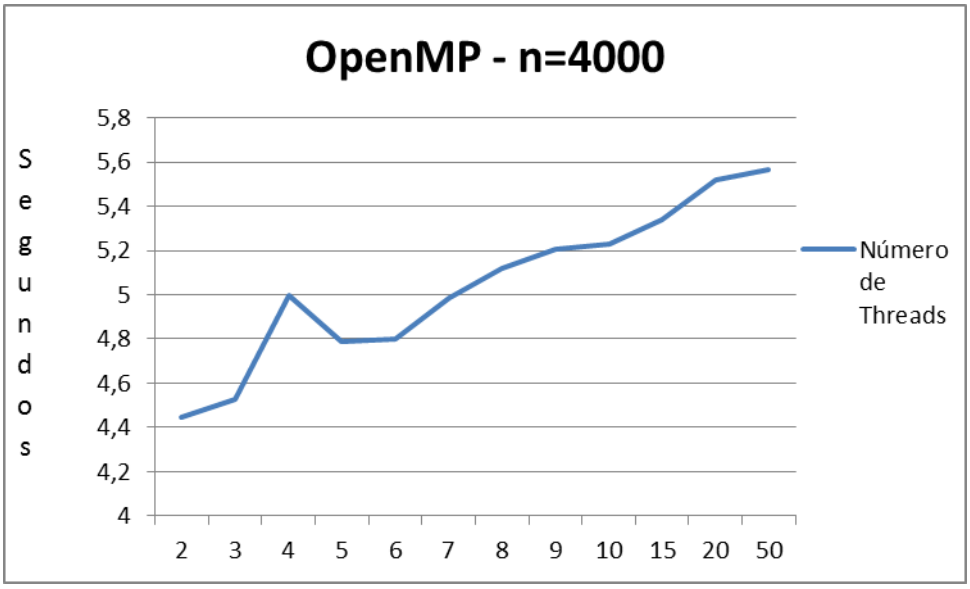

Figura 22 – Execução da matriz de ordem n=4000 pelo *OpenMP*.

5.4 SPEEDUP

Para medir o aumento de desempenho do processamento utiliza-se o cálculo chamado de *Speedup* que determina a relação existente entre o código executado em *threads* e sequencial. A medida tem por objetivo determinar a relação existente entre o tempo dispensado para executar um algoritmo em um único processador (T1) e o tempo gasto para executá-lo em p processadores (Tp): Speedup = T1/Tp (ROHDE et al, 2015). A Figura 23 mostra essa relação.

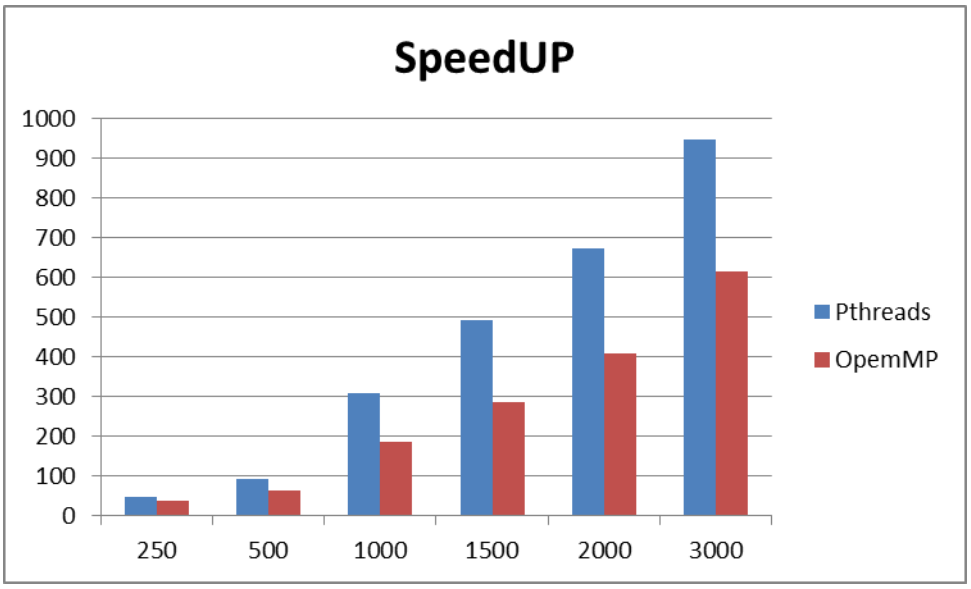

Figura 23 – SpeedUp entre os algoritmos paralelos e sequencial.

5.5 CONSIDERAÇÕES FINAIS

Conforme pode ser observado ao longo desse trabalho, paralelizar aplicações permitiu um ganho de tempo substancial em vista de algoritmos sequenciais. Deve-se ressaltar que nem todas as aplicações são paralelizáveis e que o método de abordagem do programa paralelizado também influencia no tempo de execução.

Além disso, foram encontradas algumas dificuldades no desenvolvimento desse trabalho, como entender o funcionamento das *Pthreads*, o modo como *OpenMP* distribui as tarefas, problemas com testes na matriz de ordem n=4000 e construir o algoritmo paralelo pensando na melhor forma de execução.

REFERÊNCIAS

ABREU, Luciano. **Calculo Numérico - Método de Jacobi-Richardson**. Disponível em: <https://www.youtube.com/watch?v=2AORqeCrQEc>. Acesso em: 30 ago. 2015

ALVES, Carlos J. S. **Métodos Iterativos para Sistemas Lineares**. Disponível em: < http://www.math.ist.utl.pt/~calves/cursos/SisLin-Iter.htm>. Acesso em: 30 ago. 2015

ANDRETTA, Marina. **Sistemas lineares - Método Iterativo de Jacobi-Richardson**. São Carlos, 2008.

BALBO, Antonio Roberto. **Métodos iterativos de solução de sistemas lineares**. Disponível em: <http://wwwp.fc.unesp.br/~arbalbo/Iniciacao_Cientifica/sistemaslineares/teoria/jacobi_richardson.pdf>. Acesso em: 26 ago. 2015.

BARNEY, Blaise. **OpenMP**. Disponível em: < https://computing.llnl.gov/tutorials/openMP/>. Acesso em: 03 out. 2015.

BROOKSHEAR, J. Glenn. **Ciência da computação: uma visão abrangente**. 11ª ed. Porto Alegre: Bookman, 2013.

KIESSLING, Alina. **An Introduction to Parallel Programming with OpenMP**. Disponível em: <http://www.roe.ac.uk/ifa/postgrad/pedagogy/2009_kiessling.pdf>. Acesso em: 04 out. 2015

LAMMPS, Molecular Dynamics Simulator. ***GPU and USER-CUDA package benchmarks on Desktop system with Fermi GPUs***. Disponível em: <https://computing.llnl.gov/tutorials/pthreads/#Pthread>. Acesso em: 05 set. 2015

LLNL. ***POSIX Threads Programming***. Disponível em:

<http://lammps.sandia.gov/bench.html>. Acesso em: 02 set. 2015

NVIDIA. **O que é *CUDA*?**. Disponível em: <http://www.nvidia.com.br/object/cuda_home_new_br.html>. Acesso em: 02 set. 2015

ROHDE, M. Tiago; DESTEFANI, Luciano; FERRARI, Edilaine; MARTINS, Rogério. **As diferentes técnicas de implementação paralela de algoritmos recursivos em C.** Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/erad-rs/2012/0034.pdf>. Acesso em: 02 set. 2015.

SATO, Liria Matsumoto; GUARDIA, Hélio Crestana. **Programando para múltiplos processadores: *Pthreads, OpenMP* e *MPI*.** Disponível em: <http://erad.dc.ufscar.br/mc/eradsp2013-multiproc-3.pdf>. Acesso em: 02 set. 2015.

SOUZA, Marcone Jamilson Freitas. **Sistemas Lineares**. Disponível em: <http://www.decom.ufop.br/marcone/Disciplinas/CalculoNumerico/Sistemas.pdf>. Acesso em: 03 set. 2015.

OPENMP. ***What is OpenMP*?**. Disponível em: <http://www.openmp.org/mp-documents/paper/node3.html>. Acesso em: 02 set. 2015.

OPEN-MPI. ***General information about the Open MPI Project***. Disponível em: <https://www.open-mpi.org/faq/?category=general>. Acesso em: 05 set. 2015.

RICARDO, Luis; PAULINO, Diogo; CARVALHO, Paulo. **Paralelização do algoritmo do Método Jacobi**. Disponível em: <https://drive.google.com/file/d/0B639uUhZ62fgV012UkZvSDdZeWs/view>. Acesso em: 30 ago. 2015.

TIM, C. ***Pthreads in C – a minimal working example***. Disponível em: <http://timmurphy.org/2010/05/04/pthreads-in-c-a-minimal-working-example/>. Acesso em: 30 ago. 2015.

YANO, Luis Gustavo Abe. **Avaliação e comparação de desempenho utilizando**

tecnologia CUDA. São José do Rio Preto: 2010.

ANEXO A – EXECUÇÕES SEQUENCIAIS

TABELA 5 – EXECUÇÃO DA MATRIZ DE ODEM 250 PELO MÉTODO SEQUENCIAL.

ORDEM	THREADS	DATA	TEMPO (MILISSEGUNDOS)
250	0	05/09/15 14:35	937355
250	0	05/09/15 14:37	944146
250	0	05/09/15 14:41	969399
250	0	05/09/15 14:45	941395
250	0	05/09/15 14:45	944782
250	0	05/09/15 14:45	966939
250	0	05/09/15 14:45	973535
250	0	05/09/15 14:46	983961
250	0	05/09/15 14:46	934186
250	0	05/09/15 14:46	961089

TABELA 6 – EXECUÇÃO DA MATRIZ DE ODEM 500 PELO MÉTODO SEQUENCIAL.

ORDEM	THREADS	DATA	TEMPO (MILISSEGUNDOS)
500	0	04/10/15 09:45	5261713
500	0	04/10/15 09:45	5312198
500	0	04/10/15 09:45	5267882
500	0	04/10/15 09:45	5293331
500	0	04/10/15 09:46	5298443
500	0	04/10/15 09:46	5289794
500	0	04/10/15 09:46	5246556
500	0	04/10/15 09:46	5240019
500	0	04/10/15 09:46	5260129
500	0	04/10/15 09:46	5239946

TABELA 7 – EXECUÇÃO DA MATRIZ DE ODEM 1000 PELO MÉTODO SEQUENCIAL.

ORDEM	THREADS	DATA	TEMPO (MILISSEGUNDOS)
1000	0	05/09/15 17:21	30592120
1000	0	05/09/15 17:23	61448066
1000	0	05/09/15 17:25	60549921
1000	0	05/09/15 17:26	61599126
1000	0	05/09/15 17:32	61432035
1000	0	05/09/15 17:34	61482450
1000	0	05/09/15 17:35	61447323
1000	0	05/09/15 17:37	61587433
1000	0	05/09/15 17:39	61402718
1000	0	05/09/15 17:40	61413002

TABELA 8 – EXECUÇÃO DA MATRIZ DE ODEM 1500 PELO MÉTODO SEQUENCIAL.

ORDEM	THREADS	DATA	TEMPO (MILISSEGUNDOS)
1500	0	05/09/15 17:42	102248984
1500	0	05/09/15 17:46	204817838
1500	0	05/09/15 17:49	204655042

1500	0	05/09/15 17:53	204801189
1500	0	05/09/15 18:00	204747024
1500	0	05/09/15 18:11	204735267
1500	0	05/09/15 18:16	204966621
1500	0	05/09/15 18:20	204843678
1500	0	05/09/15 18:25	204931821
1500	0	05/09/15 18:32	204735267

TABELA 9 – EXECUÇÃO DA MATRIZ DE ODEM 2000 PELO MÉTODO SEQUENCIAL.

ORDEM	THREADS	DATA	TEMPO (MILISSEGUNDOS)
2000	0	05/09/15 15:44	484082565
2000	0	05/09/15 15:53	478247389
2000	0	05/09/15 16:01	480237392
2000	0	05/09/15 16:10	479556307
2000	0	05/09/15 16:18	479689229
2000	0	05/09/15 16:31	479342916
2000	0	05/09/15 16:55	479548405
2000	0	05/09/15 17:04	479640614
2000	0	05/09/15 17:12	479334809
2000	0	05/09/15 17:20	479950013

TABELA 10 – EXECUÇÃO DA MATRIZ DE ODEM 3000 PELO MÉTODO SEQUENCIAL.

ORDEM	THREADS	DATA	TEMPO (MILISSEGUNDOS)
3000	0	05/09/15 19:03	1613572110
3000	0	05/09/15 19:39	1617364883
3000	0	05/09/15 20:09	1615317931
3000	0	05/09/15 21:03	1613572110
3000	0	05/09/15 21:31	1617364883
3000	0	05/09/15 22:00	1615317931
3000	0	05/09/15 22:36	1613572110
3000	0	05/09/15 23:11	1617364883
3000	0	05/09/15 23:50	1615317931
3000	0	06/09/15 00:33	1613572110

ANEXO B – EXECUÇÕES PARALELAS: *PTHREADS*

TABELA 11 – EXECUÇÃO DA MATRIZ DE ODEM 250 POR *PTHREADS*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
250	2	0,005844909	0,1193635
250	3	0,001697169	0,0338745
250	4	0,000699474	0,0229959
250	5	0,000175189	0,0193803
250	6	0,000337322	0,0216497
250	7	0,000246099	0,0218962
250	8	0,000244771	0,0217701
250	9	0,000204559	0,0240882
250	10	0,00020909	0,0243559
250	15	0,000197461	0,0277417
250	20	0,000341694	0,0374482
250	50	0,001177456	0,0962887

TABELA 12 – EXECUÇÃO DA MATRIZ DE ODEM 500 POR *PTHREADS*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
500	2	0,030735347	2,001331
500	3	0,012561123	0,29431
500	4	0,001868237	0,0932098
500	5	0,000645895	0,0569816
500	6	0,000751651	0,0611769
500	7	0,000943233	0,0656162
500	8	0,000988676	0,0650518
500	9	0,00056376	0,0651392
500	10	0,000624722	0,0658769
500	15	0,000498003	0,0695445
500	20	0,000565597	0,0675954
500	50	0,001827839	0,1362438

TABELA 13 – EXECUÇÃO DA MATRIZ DE ODEM 1000 POR *PTHREADS*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
1000	2	0,151144293	10,7570312
1000	3	0,15401344	4,0049701
1000	4	0,047888413	1,1855495
1000	5	0,002332534	0,1887689
1000	6	0,002525459	0,2018273
1000	7	0,001979472	0,2424278
1000	8	0,002088994	0,263045
1000	9	0,002350543	0,2341624
1000	10	0,001965605	0,2629905

1000	15	0,001965948	0,2543275
1000	20	0,001991757	0,2500758
1000	50	0,002732976	0,2992521

TABELA 14 – EXECUÇÃO DA MATRIZ DE ODEM 1500 POR *PTHREADS*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
1500	2	0,356756841	21,7196997
1500	3	0,235295018	15,1989657
1500	4	0,385301527	6,7724884
1500	5	0,004570521	0,3959849
1500	6	0,005380396	0,4523447
1500	7	0,003714276	0,5043017
1500	8	0,006416579	0,5604127
1500	9	0,004841777	0,5634784
1500	10	0,004278872	0,5597507
1500	15	0,005361765	0,5881193
1500	20	0,004137842	0,5821005
1500	50	0,003772527	0,5719834

TABELA 15 – EXECUÇÃO DA MATRIZ DE ODEM 2000 POR *PTHREADS*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
2000	2	0,276105207	31,9962087
2000	3	0,813639209	36,1758708
2000	4	0,676481274	13,9111002
2000	5	0,010868422	0,7129205
2000	6	0,006755462	0,744742
2000	7	0,007042875	0,8955244
2000	8	0,009752613	0,9208012
2000	9	0,009307143	0,936028
2000	10	0,008589566	0,9163774
2000	15	0,006416224	0,9771568
2000	20	0,007335715	1,0605202
2000	50	0,006444757	1,1164191

TABELA 16 – EXECUÇÃO DA MATRIZ DE ODEM 3000 POR *PTHREADS*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
3000	2	2,534311793	384,0293968
3000	3	1,108038448	86,5759956
3000	4	1,425758076	76,3255225
3000	5	0,026958126	3,5252359
3000	6	0,013318767	1,7827783
3000	7	0,018810684	1,7038281
3000	8	0,012875233	1,7294243
3000	9	0,011639586	1,8574677
3000	10	0,013544456	1,8728928

3000	15	0,021190148	1,8790246
3000	20	0,01012595	2,0332112
3000	50	0,015153726	2,3587592

TABELA 17 – EXECUÇÃO DA MATRIZ DE ODEM 4000 POR *PTHREADS*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
4000	2	6,702310421	1258,059046
4000	3	3,72768057	284,267874
4000	4	3,633183493	181,7880312
4000	5	0,112139899	13,462303
4000	6	0,037009648	7,3738155
4000	7	0,046051486	4,4916316
4000	8	0,031419668	3,0813353
4000	9	0,028077313	3,0283891
4000	10	0,021264158	2,9953613
4000	15	0,015352764	3,3155479
4000	20	0,035937811	3,2145234
4000	50	0,021273221	3,9147345

ANEXO C – EXECUÇÕES PARALELAS: *OPENMP*

TABELA 18 – EXECUÇÃO DA MATRIZ DE ODEM 250 PELO *OPENMP*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
250	2	0,00021646	0,0383757
250	3	0,000628303	0,0701944
250	4	0,008024047	0,19019
250	5	3,29274E-05	0,0243607
250	6	0,000131192	0,0247212
250	7	0,000101744	0,0260262
250	8	5,88187E-05	0,0264609
250	9	7,3506E-05	0,0268512
250	10	5,40789E-05	0,0270525
250	15	8,20358E-05	0,0297134
250	20	0,000327338	0,0325976
250	50	0,000209633	0,047059

TABELA 19 – EXECUÇÃO DA MATRIZ DE ODEM 500 PELO *OPENMP*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
500	2	0,000141217	0,1436495
500	3	0,001230141	0,2488983
500	4	0,000556295	0,4114003
500	5	7,89174E-05	0,0826408
500	6	9,47682E-05	0,0831002
500	7	0,000107461	0,0870604
500	8	7,13836E-05	0,0891213
500	9	0,000142679	0,0904144
500	10	0,001177023	0,0960052
500	15	0,000152508	0,0970359
500	20	0,000106334	0,0988332
500	50	0,000123694	0,1160879

TABELA 20 – EXECUÇÃO DA MATRIZ DE ODEM 1000 PELO *OPENMP*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
1000	2	0,000193063	0,3501861
1000	3	0,0003422	0,4631207
1000	4	0,000686309	0,6308179
1000	5	0,000220465	0,3148744
1000	6	0,000200477	0,3228454
1000	7	0,000220983	0,3227508
1000	8	0,000782032	0,3385973
1000	9	0,000709274	0,3405521
1000	10	0,000282927	0,3388063

1000	15	0,000266086	0,357204
1000	20	0,000274323	0,3644371
1000	50	0,000292011	0,3943974

TABELA 21 – EXECUÇÃO DA MATRIZ DE ODEM 1500 PELO *OPENMP*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
1500	2	0,000489791	0,6887524
1500	3	0,002399717	0,8101095
1500	4	0,002543567	1,017067
1500	5	0,000262572	0,6802059
1500	6	0,000296371	0,6800495
1500	7	0,000456466	0,7088291
1500	8	0,00054612	0,7257934
1500	9	0,00194818	0,7511781
1500	10	0,000551944	0,7429973
1500	15	0,000746868	0,7631534
1500	20	0,011868508	0,8328276
1500	50	0,000560257	0,8162348

TABELA 22 – EXECUÇÃO DA MATRIZ DE ODEM 2000 PELO *OPENMP*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
2000	2	0,003280427	1,1732628
2000	3	0,0008538	1,2708791
2000	4	0,007101083	1,5736235
2000	5	0,001138713	1,226486
2000	6	0,018626818	1,264386
2000	7	0,004655617	1,2793232
2000	8	0,004412968	1,2993697
2000	9	0,000829727	1,3053968
2000	10	0,00080161	1,3088045
2000	15	0,000775741	1,3321501
2000	20	0,000934982	1,3995464
2000	50	0,000960965	1,4036731

TABELA 23 – EXECUÇÃO DA MATRIZ DE ODEM 3000 PELO *OPENMP*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
3000	2	0,039994628	2,6532659
3000	3	0,001255182	2,6221947
3000	4	0,009022777	3,0681601
3000	5	0,002562502	2,6994174
3000	6	0,002740349	2,7065048
3000	7	0,001913408	2,8164997
3000	8	0,001188279	2,8660061
3000	9	0,005271193	2,949759
3000	10	0,002717784	2,937724

3000	15	0,045620909	3,1358197
3000	20	0,002815769	3,1235518
3000	50	0,001771854	3,1249703

TABELA 24 – EXECUÇÃO DA MATRIZ DE ODEM 4000 PELO *OPENMP*.

ORDEM	THREADS	DESVIO PADRÃO	MÉDIA (MILISSEGUNDOS)
4000	2	0,004068199	4,4473706
4000	3	0,002029364	4,5262573
4000	4	0,005399699	4,9987777
4000	5	0,00435111	4,7895119
4000	6	0,003169563	4,7986727
4000	7	0,001875792	4,9871919
4000	8	0,002362364	5,1206465
4000	9	0,004011034	5,2086168
4000	10	0,00516372	5,2305637
4000	15	0,004097391	5,3386516
4000	20	0,003916922	5,521651
4000	50	0,005531817	5,5659338