UNICAMP – UNIVERSIDADE ESTADUAL DE CAMPINAS

FT – Faculdade de Tecnologia

RELATÓRIO PROJETO I

"Divisão da matriz com múltiplas *threads*"

Grupo KGBcomM

Matheus Percário Bruder *RA: 222327*
Kevin Barrios *RA: 219643*
Gustavo T. S. Pereira *RA: 159740*

O projeto apresentado na Faculdade de Tecnologia de Limeira, da disciplina de sistemas operacionais, fará parte do processo de avaliação.

Disciplina TT304 – Sistemas Operacionais

Limeira – 2019

INTRODUÇÃO

O projeto tem como objetivo a criação de um programa, a partir da linguagem C, que utilize múltiplas *threads* para dividir uma matriz NxN, em que N representa as dimensões da matriz, respectivamente, linhas e colunas, em outras duas matrizes também NxN de tal forma que a primeira seja composta por elementos a partir da diagonal principal e acima; e a segunda matriz com elementos abaixo da diagonal principal.

O programa foi escrito para o sistema operacional Linux e utilizou a biblioteca *POSIX Threads*.

MATERIAL E MÉTODO

A. Instrução para a compilação do código

Para compilar o código, foi utilizado o sistema Linux e o comando:

gcc -pthread divideMat.c -o divideMat

B. Explicação do código em alto nível

Ao executar o programa, é passado por linha de comando, a dimensão da matriz o número de *threads* com qual o programa será executado e o arquivo de entrada, em que está a matriz que será processada. O número de *threads* pode ser 2, 4, 8 ou 16 e a matriz em questão é quadrática.

Se a quantidade de *threads* for respeitada, há a alocação das três matrizes, isto é, a matriz principal, a matriz diagonal 1 e a matriz diagonal 2. Caso contrário, é informado ao usuário que o número de *threads* é inválido. Após as alocações das matrizes, abre-se o arquivo de entrada que possui os dados da matriz principal, ou seja, a matriz que será processada. Se não houver erro na

abertura, inicia-se leitura dos dados do arquivo e, ao terminar, o arquivo é fechado. Vale ressaltar, que logo após o término da leitura começa a marcação

do tempo inicial, como se fosse o início do cronômetro. A partir disso, o próximo passo é a criação das *threads*, com o comando:

```
"pthread_create(&vThread[i], NULL, threadDividirMatriz, (void *)&info[i]);"
```

Esse comando está dentro de um FOR, o qual tem como limite o número de *threads*. O comando é responsável pela criação de uma *thread*, bem como, pela chamada da função que tem como responsabilidade realizar a divisão da matriz principal em matriz diagonal 1 e matriz diagonal 2. Também há um FOR responsável para que todas as *threads* retornem ao mesmo tempo, por meio do comando:

```
"pthread_join(vThread[i], NULL);".
```

Após todas as *threads* terminarem seus respectivos processamentos, há a marcação do tempo final. Então ao realizar a subtração do tempo final pelo tempo inicial tem-se o tempo total de execução.

Por fim, abre-se o arquivo de destino e se não houver erro de abertura, grava-se a matriz diagonal 2 no arquivo e o fecha. O mesmo procedimento ocorre para matriz 1 diagonal 1.

C. Tempos de execução em cada computador

Para realizar os cálculos referentes ao tempo de execução, foi realizado testes em quatro diferentes máquinas com cinco arquivos de entrada diferentes, mas todos eles relativos a uma matriz quadrática de dimensão 1000 x 1000. Todos os tempos estão em milissegundos (*ms*).

O primeiro computador em que os foram realizados os testes possuía as seguintes configurações: Processador i7-8650H 2.20GHz, 16,0GB RAM, Ubuntu 18.4 (Tabela 1).

	Número de Threads			
Testes	**2**	**4**	**8**	**16**
T1	59,213	55,635	61,184	69,240
T2	54,371	42,407	47,899	68,324
T3	52,709	48,238	54,050	71,521
T4	52,799	50,882	49,151	75,026
T5	52,162	50,645	56,453	77,286
Média	**54,251**	**49,561**	**53,747**	**72,279**
Desvio Padrão	**2,893**	**4,815**	**5,432**	**3,810**

Tabela 1 - Referente aos tempo de execução, em milissegundos do primeiro computador, utilizando 2, 4, 8 e 16 'threads'

O segundo computador em que os foram realizados os testes possuía as seguintes configurações: Processador i7 7500U 2.70GHz, 3,0GB RAM, Ubuntu 18.4 (Tabela 2).

	Número de Threads			
Testes	**2**	**4**	**8**	**16**
T1	34,909	25,262	47,752	41,520
T2	24,776	26,717	30,636	38,563
T3	29,433	26,590	28,104	39,579
T4	26,756	31,758	31,037	38,249
T5	23,635	24,362	27,994	49,405
Média	**27,902**	**26,938**	**33,105**	**41,463**
Desvio Padrão	**4,493**	**2,866**	**8,307**	**4,620**

Tabela 2 - Referente aos tempo de execução, em milissegundos do segundo computador, utilizando 2, 4, 8 e 16 'threads'

O terceiro computador em que os foram realizados os testes possuía as seguintes configurações: Processador i7 8750H 2.20GHz, 2,0GB RAM, Ubuntu 18.4 (Tabela 3).

	Número de Threads			
Testes	**2**	**4**	**8**	**16**
T1	24,639	30,010	26,931	33,557
T2	25,323	26,512	27,39	35,674
T3	28,255	41,003	25,445	36,418
T4	24,408	25,197	25,191	47,261
T5	25,636	28,203	27,31	36,485
Média	**25,652**	**30,185**	**26,453**	**37,879**
Desvio Padrão	**1,538**	**6,312**	**1,055**	**5,377**

Tabela 3 - Referente aos tempo de execução, em milissegundos do terceiro computador, utilizando 2, 4, 8 e 16 'threads'

O quarto computador em que os foram realizados os testes possuía as seguintes configurações: Processador i7 87500H 2.20GHz, 4,0GB RAM, Ubuntu 18.4(Tabela 4).

	Número de Threads			
Testes	**2**	**4**	**8**	**16**
T1	16,779	17,827	22,932	35,214
T2	17,531	17,340	22,045	33,996
T3	16,670	16,425	25,457	33,809
T4	16,937	19,544	28,528	32,850
T5	15,859	17,305	22,114	32,977
Média	**16,755**	**17,688**	**24,215**	**33,769**
Desvio Padrão	**0,601**	**1,154**	**2,780**	**0,950**

Tabela 4 - Referente aos tempo de execução, em milissegundos do quarto computador, utilizando 2, 4, 8 e 16 'threads'

RESULTADOS

A partir das tabelas anteriores, com os tempos de execução em cada um dos computadores, é possível criar a tabela referente a média de todos os computadores e então gerar um tempo de execução médio dos resultados entre todas as máquinas (Tabela 5).

	Número de Threads			
	2	**4**	**8**	**16**
PC Lab 03	54,251	49,561	53,747	72,279
PC - Matheus	27,902	26,938	33,105	41,463
PC - Kevin	25,652	30,185	26,453	37,879
PC - Gustavo	16,755	17,688	24,215	33,769
Média dos tempos *(ms)*	**31,140**	**31,093**	**34,380**	**46,348**
Desvio Padrão *(ms)*	**16,141**	**13,402**	**13,452**	**17,571**

Tabela 5 - Referente aos tempo de execução médio, em milissegundos, de quatro computadores diferentes, utilizando 2, 4, 8 e 16 'threads'

A partir da Tabela 5 pode-se plotar um gráfico relativo à média dos tempos de execução dos quatro computadores. No gráfico, além da média, também será plotado o desvio padrão com o intuito de exibir a variação entre cada um desses computadores.

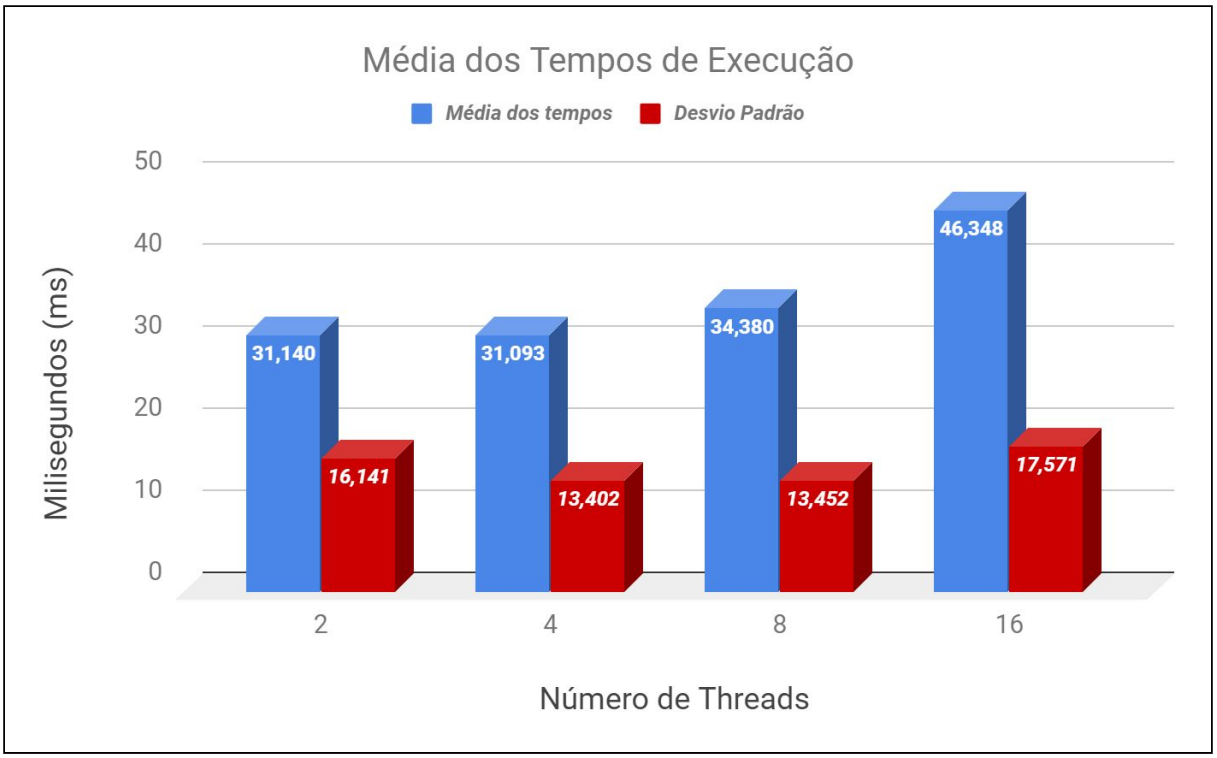

Figura 1 - Gráfico referente ao tempo de execução médio, em milissegundos, utilizando 2, 4, 8 e 16 'threads'

A partir do gráfico da Figura 1, infere-se que o melhor resultado foi obtido com a utilização de quatro *threads*. Nota-se que a diferença entre o tempo de execução médio utilizando duas e quatro *threads* é muito pequeno, ou seja, utilizando as quatro *threads* (melhor resultado), foi atingido um tempo de processamento de 47 microssegundos mais rápido.

As demais diferenças entre os tempos de execução são discrepantes e, portanto, respeitam a ordem de milissegundos. Por exemplo, ao dividir a matriz utilizando oito *threads* o tempo de execução foi 11,986 milissegundos mais rápido do que quando foram usadas dezesseis *threads* para processar a divisão da mesma matriz.

CONCLUSÃO

No início do projeto, era esperado que o tempo médio de execução fosse inversamente proporcional ao número de *threads,* ou seja, quanto maior fosse o número de *threads,* menor seria o tempo de execução. Esse resultado era esperado, pois, ao alocar dezesseis *threads* trabalhando simultaneamente, espera-se que desempenho seja maior do que duas *threads* trabalhando simultaneamente na mesma tarefa.

Entretanto, os resultados alcançados mostraram que para o programa desenvolvido, o ideal é trabalhar com quatro *threads*, isso porque é um programa de baixa complexidade. Logo, a criação de cada uma das *threads* demanda muito tempo e isso faz com que o desempenho fique menor.

Contudo, para programas com uma alta complexidade, possivelmente o tempo médio de execução será inversamente proporcional ao número de *threads.*

LINK GITHUB

Repositório KGBcomM - GitHub

LINK GOOGLE DRIVE

Pasta Google Drive com vídeo
Pasta Google Drive com as planilhas dos tempos de execução