UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE TECNOLOGIA

Missão: Formar e aperfeiçoar cidadãos e prestar serviços atendendo às necessidades tecnológicas da sociedade com agilidade, dinâmica e qualidade.

Sistemas Operacionais

Projeto “Divide Matriz”

Grupo: Threads FTW

Larissa Benevides Vieira	RA: 200805
Juliana Almeida Morroni	RA: 200372
Gabriel Domingues Ferreira	RA: 216207

Limeira - SP
2019

SUMÁRIO

1. OBJETIVO

Este projeto visa a criação de um programa que utilize múltiplas threads para dividir uma matriz N x N (N linhas por N colunas) em outras duas matrizes também N x N de tal forma que seja composta por elementos a partir da diagonal principal e acima; e a segunda matriz com elementos abaixo da diagonal principal. O programa deverá ser escrito para o sistema operacional Linux e obrigatoriamente utilizar a biblioteca POSIX Threads.

2. DESCRIÇÃO DO PROBLEMA A SER RESOLVIDO

Considere uma matriz N x N (N linhas por N colunas) que contém valores em ponto flutuante, positivos ou negativos. O programa deverá utilizar múltiplos threads para dividir essa matriz em outras duas.

A matriz original deve ser dividida em outras duas matrizes também N x N de tal forma que a primeira matriz seja composta por elementos a partir da diagonal principal e acima; e a segunda matriz com elementos abaixo da diagonal principal. Os dados da matriz original devem vir de um arquivo e as matrizes resultantes devem ser gravadas em arquivos com as extensões diag1 para os dados da primeira matriz e diag2 para os dados da segunda matriz. O programa deve ser testado para 2, 4, 8 e 16 threads, com matrizes 1000 x 1000.

3. ENTRADAS E SAÍDAS DE VALORES PARA O PROGRAMA

Entradas: os valores N, T e Arquivo -- respectivamente, as dimensões da matriz (N x N); o número de threads; e o arquivo onde estão os dados -- devem ser informados pelo usuário no início do programa. Para os testes, considere valores grandes (maiores ou iguais a 100) para N.

Saídas: Arquivos com as matrizes diagonal superior e diagonal inferior. Os arquivos gerados devem ter o mesmo nome do arquivo original e as extensões diag1 para a primeira matriz e diag2 para a segunda matriz.
Como sugestão, construa o programa de modo que os valores de entrada sejam lidos da linha de comando. Por exemplo:
./divideMat 1000 16 matriz.dat

Onde:

- ./divideMat é o nome do programa;
- 1000 é a dimensão da matriz (1000 x 1000);
- 16 é o número de threads;
- matriz.dat é o arquivo que contém os dados da matriz.

4. CÓDIGO FONTE

Código disponível no seguinte diretório apresentado no github:
<https://github.com/Laribene/Sistemas-Operacionais/blob/master/thread.c>

5. VÍDEO DO CÓDIGO

Vídeo explicativo do código e apresentação dos resultados:
<https://youtu.be/ycccneFk25c>

6. INSTRUÇÕES SOBRE O PROGRAMA

6.1 Descrição da solução do problema

O objetivo deste projeto, assim como fora explicado, é criar uma matriz quadrada de ordem N com o intuito de dividi-la em duas partes (superior e inferior) e criar múltiplas threads (2, 4, 8, 16) para percorrerem a matriz e efetuar sua leitura, assim analisando o tempo de execução para cada uma das threads.

O programa seguiu o seguinte algoritmo para ser realizado:

Primeiramente, foram incluídas bibliotecas padrão para a criação de threads, e também para o uso de uma função que calcula o tempo de processamento em milissegundos, analisando seu desempenho de forma mais assertiva (automatizada).

A declaração de variáveis fora de grande importância para a realização do código, inclusive a criação da estrutura que serviu como base para passar os parâmetros as threads. No entanto, dentro da função principal também foram declaradas variáveis para armazenar o número de threads passado pelo usuário e a ordem da matriz.

Sendo que fora criado uma condição para ter um número máximo de threads para que não ocorra erros significativos, como por exemplo, o número de threads deve ser sempre menor ou igual a ordem da matriz.

Foi disponibilizado um programa a parte, gerado pelo professor, que promoveu a criação de uma matriz, garantindo a aleatoriedade de seus valores.

A partir daí, no nosso código, será feita a abertura do arquivo para a utilização da matriz aleatória gerada, para assim ser realizada a divisão da mesma. Para isso, será utilizada a alocação de memória dinâmica para a matriz principal, assim como para as matrizes auxiliares (superior e inferior).

Posteriormente foram feitas as criações das múltiplas threads. Assim seus argumentos serão definidos a partir da atribuição dos valores das variáveis, como o número de threads, ordem da matriz, matriz principal, entre outras.

Dentro do laço de repetição, houve o cálculo do resto da divisão que é calculado caso haja situações do tipo: uma matriz com o número total de elementos sendo ímpar, e o número de threads usadas sendo par, sobrará um elemento no final sem ser processado. Por exemplo, 25 elementos para 2 threads, seria 12 elementos utilizados para cada thread, usando esta condição, quando a última thread é processada será mandado o número de elementos (12) + o resto (1), sendo processados 13 elementos. Ou seja, nenhum elemento fica fora do processo.

A partir disso, na divisão, a matriz será separada em duas partes (parte superior e inferior). Este recurso é feito a partir das threads criadas que percorrerão a matriz linha por linha para a identificação da metade desta matriz. Sendo que a thread 1 percorrerá até a primeira metade da matriz, e a thread 2 percorrerá da metade até o fim da mesma.

Logo após, fora feita a junção das threads criadas, para que a função principal seja travada até que todas as threads terminem e o programa possa continuar.

Com isso, usa-se a função de tempo de processamento novamente para que seja armazenado o tempo final em uma variável, e então calculado a diferença de tempo que durou a divisão da matriz, e assim, analisando seu desempenho final.

Concluindo, os dados do resultado da divisão que estavam na memória serão escritos no arquivo, passando os valores da diagonal superior e diagonal inferior. Logo os arquivos são fechados, assim finalizando o processo.

6.2 Instruções para compilação

Para a compilação do programa, é aconselhável a utilização do sistema Linux, as instruções aqui descritas serão baseadas neste ambiente:

1- Primeiramente é necessário acessar o repositório disponível no seguinte endereço do Github: <https://github.com/Laribene/Sistemas-Operacionais>

2- Entrando nele, deve ser feito o clone do arquivo ou então baixar os arquivos lá postados;

3- Após isso, deve-se acessar o terminal de seu computador para acessar a pasta na qual os arquivos foram baixados (etapa 2), inserindo assim, os seguintes comandos respectivamente:

```
gcc -pthread thread.c -o divideMat
./divideMat 1000 X matriz.dat
```

X sendo o número de threads (2, 4, 8, 16)
1000 sendo a ordem da matriz, designada pelo professor

4- Assim, o código será executado conforme as solicitações dadas pelo professor, mostrando assim, o tempo utilizado em milisegundos.

7. RESULTADOS OBTIDOS

Foi executado o programa com 2, 4, 8 e 16 threads em um computador com processador Intel Core i5-7200U de 2.5GHz , placa de video NVIDIA GeForce 940MX, memória de 8gb RAM DDR4, com HD de 1TB da fabricante ACER, modelo A515-51G.

7.1 Gráfico

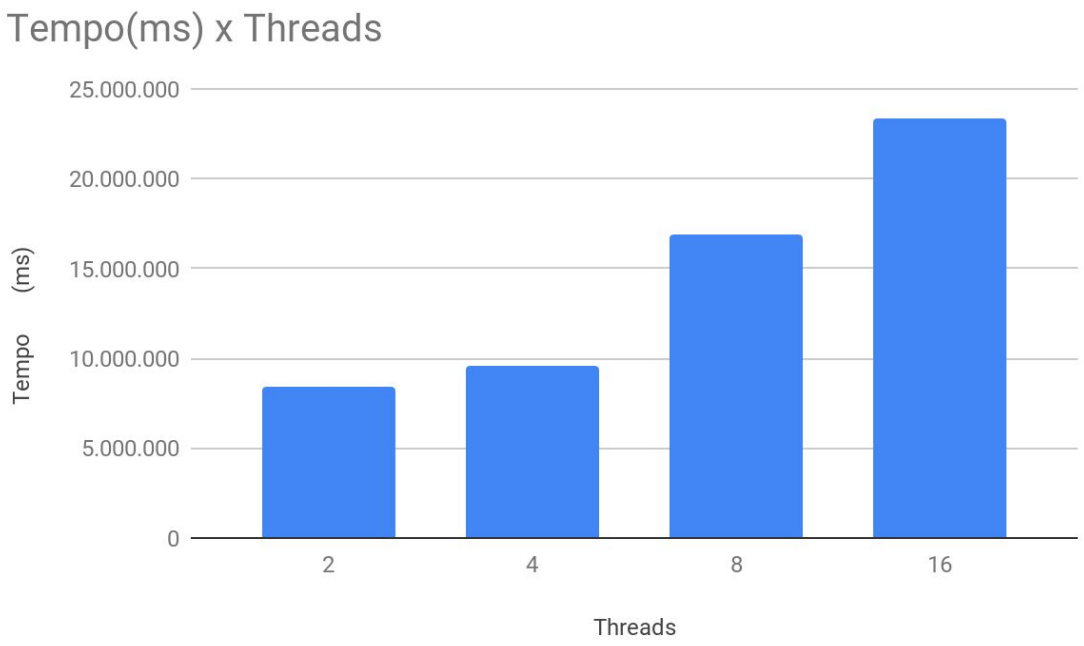

Figura 1 - Tempo de execução x número de threads em matriz 1000x1000

Threads	Tempo (ms)
2	8.459000
4	9.630000
8	16.903999
16	23.384001

Tabela 1 - Tempo de execução de cada thread em matriz 1000x1000

8. CONCLUSÃO

Os dados obtidos nos fornecem as seguintes conclusões:

Exemplificando de uma maneira geral, com o uso de 2 threads, cada thread trabalharia com metade da matriz, usando se 4 threads, cada uma ficaria com ¼ da matriz, e 8 threads com ⅛ e assim sucessivamente. Neste sentido, poderíamos inferir que quanto mais threads mais rápido seria o processamento, mas na verdade não podemos afirmar isto, pois a quantidade de dados processados pode ser relativamente pequena, e o número de processadores também pode interferir neste âmbito, assim podendo perder sua eficiência. Pois o processo sendo pequeno gastaria um maior tempo de criação de threads que seriam desnecessárias.

A partir dos dados observados em nosso teste, podemos concluir que o processamento com 2 threads se mostrou mais eficiente por obter um menor tempo de processamento, mas como o tempo de execução da divisão da matriz é variável, (dependente dos processos que estão em execução no computador no momento) algumas vezes o tempo menor é com 4 threads, já que o tempo de processamento de 2 e 4 threads são bem próximos.

Portanto, no teste realizado no vídeo, a execução com 2 threads foi melhor, comparada com o uso de 4, 8 e 16 threads. Demonstrando que quanto mais threads não necessariamente o tempo de execução do código será mais satisfatório.