Trio Elétrico

Arthur Briganti Gini 213253
Leonardo Alberto da Ponte 220007
Lorenzo Chaves Vilas Boas 220467

Rotaciona Matriz

06 de junho de 2019

Visão geral

Este projeto visa a criação de um programa que utilize múltiplas threads (2, 4, 8 e 16) para rotacionar uma matriz N x M (N linhas por M colunas) 90º no sentido horário. O programa deverá ser escrito para o sistema operacional Linux e obrigatoriamente utilizar a biblioteca POSIX Threads (pthreads.h).

Objetivos

1. **Proposta:** Considerar uma matriz N x M (N linhas por M colunas) que contém valores em ponto flutuante, positivos ou negativos. O programa deverá utilizar múltiplos threads para rotacionar essa matriz em 90º conforme o exemplo ilustrado na figura a seguir para uma matriz 4 x 4.

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \Rrightarrow \begin{bmatrix} a_{3,0} & a_{2,0} & a_{1,0} & a_{0,0} \\ a_{3,1} & a_{2,1} & a_{1,1} & a_{0,1} \\ a_{3,2} & a_{2,2} & a_{1,2} & a_{0,2} \\ a_{3,3} & a_{2,3} & a_{1,3} & a_{0,3} \end{bmatrix}$$

2. **Produtos esperados:**

2.1. O código fonte do programa completo, documentado e pronto para ser compilado em sistemas Linux.

2.2. Um vídeo, mostrando o código fonte do programa, a compilação do programa, um trecho do arquivo de entrada e a execução do programa para 2, 4, 8 e 16 threads.

2.3. Um relatório contendo a descrição da solução do problema; instruções para compila-lo; gráficos com os tempos de execução do programa para 2, 4, 8 e 16 threads; e as conclusões a respeito dos resultados obtidos.

Código Fonte:

```
/*==========================================================================
====

* O código a seguir foi construído para a disciplina de Sistemas
Operacionas(TT304A_2019S1) da Faculdade de Tecnologia (UNICAMP).

* Autores: Arthur Gini, Leonardo Ponte, Lorenzo Chaves.

* Professor: Leon Gradvolh .

* Última modificação: qui 06 Jun 2019

*

* O código recebe uma matriz em arquivo N x M gira ela em 90 graus e
retorna para um arquivo de saída.

* Para realizar a execução do código é necessário digitar os valores via
linha de comando

* Exemplo:

*

* Os parâmetros necessários são os seguintes:

*    ./rotacionaMat é o nome do programa;

*    1000 é o número de linhas da matriz;

*    500 é o número de colunas da matriz;

*    16 é o número de threads; e

*    matriz.dat é o arquivo que contém os dados de entrada da matriz.

*    matriz.rot é o arquivo que contém os dados da matriz rotacionada.
```

```
*

*Exemplo de execução via linha de comando:

*./rotacionaMat 1000 500 16 matriz.dat matriz.rot

*================================================================================
====

*/

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>//Biblioteca para a implementação das threads em
sistemas Linux

#include <time.h>//Biblioteca para a contagem do tempo de processamento


typedef struct atributos{


  int start;


}Argumentos;


int linhas, colunas, numThreads;//variaveis de definição da matriz

int z; //variavel auxiliar no for de passar thread
```

```
double **matriz;//ponteiro para o ponteiro da matriz inicial

double **bobby;//ponteiro para o ponteiro da matriz final


Argumentos Args[16];//Gera 16 structs


FILE *arquivo_entrada;

FILE *arquivo_saida;


//determinar tempo de execução

clock_t tempo;//variáveis para armazenar o tempo de processamento


void *tombaMatriz(void *args);//declaração da função gira matriz


int main(int argc, char *args[])

{

  linhas = atoi(args[1]);//entrada do número de linhas via command line

  colunas = atoi(args[2]);//entrada do número de colunas via command line

  numThreads = atoi(args[3]);//entrada do número de threads via command
line

  arquivo_entrada = fopen(args[4], "r");//capturar o arquivo de entrada a
partir da linha de comando
```

```
  arquivo_saida = fopen(args[5],"w");//captura o arquivo de saída a
partir da linha de comando


  pthread_t tids[numThreads];//Alocação da quantidade de threads


  //Alocação dinâmica da matriz principal

  matriz = (double **) malloc(linhas * sizeof(double *));     //alocação
dinâmica das linhas da matriz principal

  for(int i = 0; i < linhas; i++){                            //alocação
dinâmica das colunas da matriz principal

      matriz[i] = (double*) malloc(colunas * sizeof(double));

  }



  //Alocação dinâmica da matriz auxiliar

  bobby = (double **) malloc(colunas * sizeof(double *));

  for(int i = 0; i < colunas; i++){

      bobby[i] = (double*) malloc(linhas * sizeof(double));

  }


   //leitura da matriz por arquivo

  for(int i=0;i<linhas;i++){
```

```
        for(int j=0;j<colunas;j++){
            fscanf(arquivo_entrada,"%lf", &matriz[i][j]);
        }
    }

    //Inicia a contagem do tempo de processamento
    tempo = clock();

    //criação das threads
    for(z = 0; z < numThreads; z++){
        pthread_create(&tids[z], NULL, tombaMatriz, &Args[z]);//Criação de z threads
    }
     //Join das threads
    for(z = 0; z < numThreads; z++){
        pthread_join(tids[z], NULL);
    }

    tempo = clock() - tempo; //Calcula quanto tempo o programa leva para executar o giro utilizando threads
    printf("Tempo de execucao: %lf milisegundos\n", ((double)tempo)/((CLOCKS_PER_SEC/1000))); //conversão para double e tempo para milissegundos
```

```
    //Impressão da matriz gerada no arquivo
    for(int i=0;i<colunas;i++){
        for(int j=0;j<linhas;j++){
            fprintf(arquivo_saida,"%2.f",bobby[i][j]);
        }
        fprintf(arquivo_saida,"\n");
    }
  fclose(arquivo_entrada);//fecha o arquivo de entrada da matriz
  fclose(arquivo_saida);//fecha o arquivo de saída da matriz
    return 0;
}

//Função que gira a matriz em 90 graus
void *tombaMatriz(void *arg)
{
  double auxiliar=0;
  int aux = linhas-1;
  for(int j=0;j<colunas;j++){
      for(int i=0;i<linhas;i++){
       auxiliar=matriz[aux-i][j];
```

```
            bobby[j][i]=auxiliar;
          }
    }
}
```

Algoritmo de alto nível:

Segue abaixo o Algoritmo em alto nível da resolução do projeto:

1.Receber as entradas da linha de comando, e alocá-las em suas respectivas variáveis.

2. Aloca a variável “matriz” dinamicamente, e logo após aloca a matriz auxiliar “bobby” também dinamicamente.

3. Passa os dados do arquivo “matriz.dat” para a variável “matriz”.

4.Inicia a contagem de tempo de processamento através do comando “clock()”

5. Laço para criação de um vetor de threads, que chamam a função que irá rotacionar a matriz.

6. A função “tombaMatriz” gira a matriz em 90° graus para o sentido horário.

7. Laço na função “tombaMatriz”’ que a percorre a variável “matriz” e vai atribuindo à matriz auxiliar (bobby), que será a rotacionada.

8. Com a matriz auxiliar já rotacionada pelas threads, é declarado um laço que chama a função “pthread_join” no decorrer do vetor de threads.

9. Laço de impressão da matriz rotacionada “bobby” no arquivo de saída “matriz.rot”.

10. Fechamento do arquivo de entrada com o comando “fclose()”

11. Fecha o arquivo de saída também com o comando “fclose()”

Instruções para compilação:

Os parâmetros necessários são os seguintes:

Para compilar o código o usuário deve obedecer às seguintes instruções:

Abrir o terminal do Linux.

Abra a pasta onde se localiza os arquivos.

Digite o seguinte comando:

```
gcc Main.c -o rotacionaMat -pthread
```

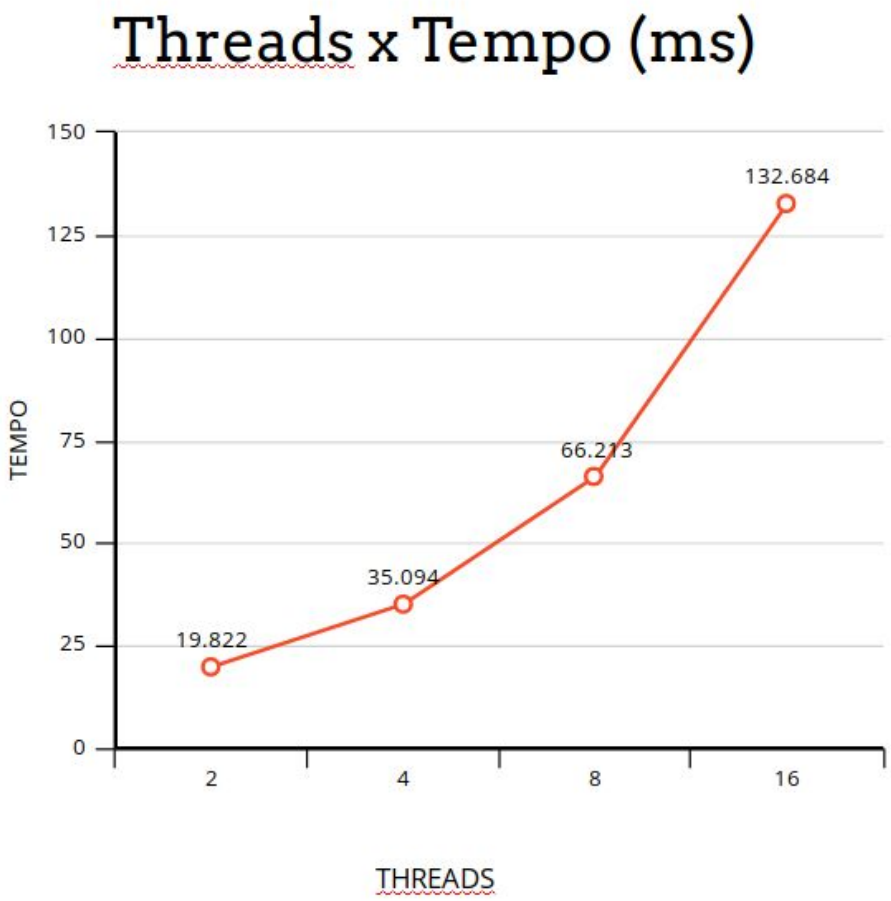
- gcc compilador para o programa;
- -o sinaliza a utilização da biblioteca de entrada e saída;
- Main.c é o nome do programa;
- rotacionaMat é o nome do executável.
- -pthread é o indicador do compilador para multi-processamento.

Segue um exemplo da linha de comando para a execução do programa:

```
./rotacionaMat 1000 500 16 matriz.dat matriz.rot
```

- ./rotacionaMat é o nome do programa;
- 1000 é o número de linhas da matriz;
- 500 é o número de colunas da matriz;
- 16 é o número de threads; e
- matriz.dat é o arquivo que contém os dados de entrada da matriz.
- matriz.rot é o arquivo que contém os dados da matriz rotacionada.

Gráfico Tempo x Threads:



Vídeo:

O vídeo do programa em execução pode ser encontrado no YouTube no link a seguir:

https://youtu.be/2FWhlasaeuQ

A execução no vídeo foi realizada em um PC no sistema operacional Ubuntu com as seguintes especificações:

Placa-mãe: Positivo

Processador: i5 - 6400 - 2.7GHz

Memória RAM: 8GB DDR4

Conclusão:

Com base na observação do gráfico, podemos ver que o uso de múltiplas threads sem a divisão da operação de rotação da matriz entre elas, só acarreta em um tempo de processamento maior, fazendo assim com que nesse caso seja melhor que o próprio computador decida quais threads vão realizar os processos.

Repositório completo no GitHub:

https://github.com/ArthurGini/SO