**Universidade Estadual de Campinas**

**Faculdade de Tecnologia da Unicamp**

# Sistemas Operacionais

Projeto 1 - “Divide Matriz” | Maligrupo

**Membros do grupo:**


Matheus Alves da Silva


Miguel Amaral


Roberta Gomes da Silva

# SUMÁRIO

Projeto “Divide Matriz” | Maligrupo

## DESCRIÇÃO DO PROJETO DESENVOLVIDO

Este grupo criou um programa que utiliza 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 foi escrito para o sistema operacional Linux e utilizou a biblioteca POSIX Threads para implementar as threads.

Os dados da matriz original vêm de um arquivo com valores flutuantes e aleatórios que é gerado e gravado em tempo de execução, enquanto que as matrizes resultantes são gravadas em arquivos de mesmo nome, mas com as extensões diag1 para os dados da primeira matriz e diag2 para os dados da segunda matriz.

Dentro do repositório do projeto é possível encontrar: o programa principal, chamado “*divideMatriz*.c”, contendo as funções que realizam a criação e execução das threads que realizam a divisão da matriz, junto com a contagem de tempo, definido pela biblioteca <sys/time.h>, o programa chamado “*arquivo*.c”, para manipular os arquivos e o cabeçalho <*arquivo*.h>, incluído por “*arquivo*.c”.

O programa foi testado para 2, 4, 8 e 16 threads, com matrizes 1000 x 1000. Para executar o código, foi criado um arquivo makefile, que facilita a compilação no terminal. As entradas do programa são informados diretamente da linha de comandos, com os valores N, T e Arquivo, onde:

- **N** = as dimensões da matriz (N x N) que resultará em outras duas;
- **T** = o número de threads que fará o processo de dividir a matriz;
- **Arquivo** = nome do arquivo que gera as extensões e armazena os dados.

O projeto está disponível no repositório **GitHub** do grupo .
Para acessá-lo, basta clicar no link abaixo:

GitHub

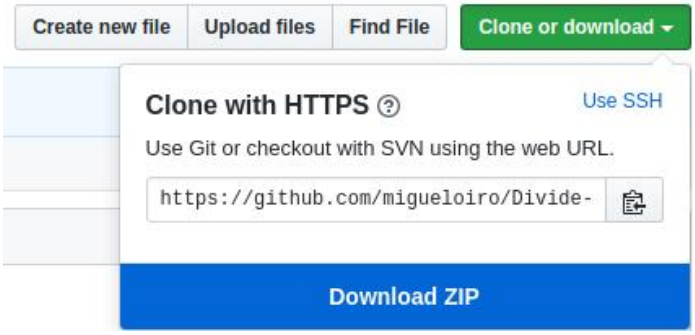
https://github.com/migueloiro/Divide-Matriz

Projeto "Divide Matriz" | Maligrupo

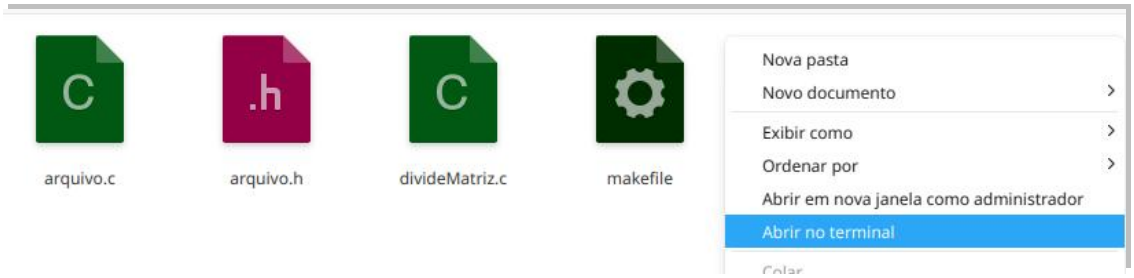
## INSTRUÇÕES DE COMPILAÇÃO

**Primeiro passo**: acesse o repositório do grupo no GitHub, no link acima.

**Segundo passo**: faça o download do repositório:




**Terceira passo**: descompacte o arquivo .zip e abra a pasta no terminal:



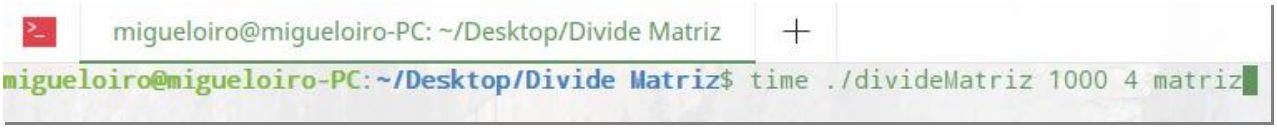

**Quarto passo**: no terminal, insira o comando "make" (sem aspas) e aperte enter:

**Quinto passo**: após a compilação, insira os dados para executar o programa:


Onde:

- ./divideMatriz é o nome do programa;
- 1000 é a dimensão da matriz (1000 x 1000);
- matriz é o nome do arquivo que trabalhará com os dados (até 30 caracteres)

**Obs**: para este exemplo, a saída da compilação gerará as matrizes resultantes com os arquivos matriz.diag1 para a matriz superior e matriz.diag2 para a matriz inferior. Também é exibido o tempo de execução das threads em milissegundos e em segundos. Para saber o tempo de execução total do programa, basta inserir o comando time, da seguinte forma:



Projeto “Divide Matriz” | Maligrupo

## DESCRIÇÃO DA SOLUÇÃO DO PROBLEMA

Após compilar, executar e fornecer os parâmetros de entrada do programa, o mesmo verifica se o número de parâmetros informado está correto. Se não estiver, o programa encerra ali mesmo, exibindo uma mensagem de erro. Essa verificação é importante para garantir que tudo se execute da maneira correta.

Após, a função main faz a chamada da função que aloca, dinamicamente, as matrizes que serão utilizadas. Para este projeto, são alocadas três matrizes no total. Caso tudo ocorra bem com a alocação, a função gravaArquivo gera números aleatórios (uma matriz de dimensão 100 gerará 1000 arquivos), em ponto flutuante e grava estes valores em um arquivo, com o nome que foi informado como parâmetro, na linha de comandos. Com o arquivo gerado e os valores gravados nele, a função gravaMatriz é chamada para fazer a gravação destes dados na matriz original, a primeira que foi alocada dinamicamente.

Após as operações acima, é executada a função thread, que cria o número de threads de acordo com o informado na linha de comandos e indica qual função essas threads devem processar. A contagem de tempo entre duas marcações começa antes da criação das threads e termina após a execução de todas elas. A função indicada para as threads executarem é a divideMatriz, que realiza a divisão da matriz original em outras duas: a superior e a inferior.

Posterior a isto, a função main faz a chamada das funções gravaArquivo_diag1 e gravaArquivo_diag2, que gravam as matrizes resultantes em arquivos com o mesmo nome do original, mas com as extensões correspondentes. As manipulações com arquivos são feitas em outro programa, chamado “arquivo.c”. Se tudo ocorrer conforme o esperado, as matrizes são liberadas da memória e o programa encerra.

O grupo gravou um vídeo que mostra a descrição do código fonte, a compilação e a execução do programa. O link para o vídeo está disponível abaixo:

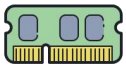
https://bit.ly/2K01QAw

## RESULTADOS

Para os testes, um notebook com as seguintes configurações foi utilizado:

Asus Vivobook X510UR | Sistema Operacional: Deepin Linux 15.9


Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
Nº de núcleos: 2 | Nº de threads: 4



4 GB DDR4 2400MHz

Foram realizados testes com 2, 4, 8 e 16 Threads, com matrizes 1000 x 1000.


**Obs**: o tempo das Threads pode variar a cada execução, pois os dados são gerados de maneira aleatória e nunca são os mesmos a cada compilação, podendo ser mais rápida ou mais demorada a leitura das matrizes, na hora da divisão das mesmas. No entanto, para este projeto, as diferenças entre execuções com o mesmo número de Threads é quase imperceptível, mesmo em milissegundos.

## CONCLUSÃO

Nos resultados obtidos no tópico acima, o menor tempo de execução foi com 4 threads, enquanto que o maior foi com 16 threads. Muitas pessoas podem pensar que, quanto maior o número de threads, mais rápida será a execução do programa. Como ilustrado no gráfico, isto é verdade até certo ponto, no entanto, como visto em aula, uma CPU possui um número limitado de threads (a CPU utilizada para esses testes possuí 4 threads). Um processo que utiliza múltiplas threads gera “disputas”, pois tais threads requisitam os recursos alocados a outras threads no mesmo processo, podendo ocasionar em deadlock, aumentando o tempo de execução do programa. O responsável por essa “disputa”, neste projeto, é o método “pthread_join”, que garante que todas as threads serão executadas antes de o programa acabar.

Ou seja, a programação com múltiplas threads nem sempre é uma vantagem, pois, em alguns casos, pode não ser recomendado utilizá-las. Com isso, é possível concluir que não se deve inserir mais threads que CPU’s disponíveis.

# REFERÊNCIAS

GitHub

Alocação dinâmica de matrizes em apenas uma etapa:
https://github.com/gradvohl/alocaMatrizes/blob/master/matrizesSimplesDinamicas.c

---

UFMG

Tutorial POSIX Threads:
https://homepages.dcc.ufmg.br/~coutinho/pthreads/ProgramandoComThreads.pdf

---

Makefile Generator:
http://solver.assistedcoding.eu/makefilegen

---

stackoverflow

Calculando o tempo passado entre duas marcações:

https://stackoverflow.com/a/27448980

Projeto “Divide Matriz” | Maligrupo

# CÓDIGO FONTE DO PROJETO

## divideMatriz.c

```c
/* Inclusoes Padrao. */
#include <stdio.h> //printf(), fscanf(), fprinft() e perror().
#include <stdlib.h> //malloc(), free(), exit(), atoi(), rand() e srand().
#include <pthread.h> //Biblioteca POSIX Threads.
#include <string.h> //strcpy.
#include <sys/time.h> //gettimeofday() e struct timeval.

/* Inclusao criada. */
#include "arquivo.h"

/* Prototipo das Funcoes. */
void *divideMatriz(void *tArg);
void thread (int t, int n);
void alocaMatriz (int n);
double diferencaTempo_miliseg(struct timeval inicio, struct timeval fim);

/* Estrutura com o Argumento para as Threads executarem. */
struct new
{
   int n; //Dimensao da Matriz (N x N).
}flag;

/* Declaracao das Matrizes que serao utilizadas. */
double *matriz; //Matriz original.
double *matrizSuperior;
double *matrizInferior;

/**
 * Funcao principal do programa.
 * @Parametro argc -> Quantidade de argumentos.
 * @Parametro argv -> Vetor que contem os argumentos.
 * @return zero.
 */
int main(int argc, char *argv[])
{
   /* Declarando as variaves: */
   /* Armazenam que foi digitado pelo usuario na linha de comandos. */
   int n = atoi(argv[1]); //Armazena a dimensao da matriz (N x N).
   int t = atoi(argv[2]); // Armazena o numero de Threads.
   char nomeArquivo[31]; //Armazena o nome do Arquivo com os dados para a Matriz.
   strcpy(nomeArquivo,argv[3]);

   flag.n = n; //Preenche a Struct com a dimensao da Matriz.

   /* Tratamento de erro para caso a qtd de parametros fornecidos for invalida: */
   if (argc != 4)
   {
      perror("Numero de parametros invalido, tente novamente!");
      exit(EXIT_FAILURE);
   }

   /* Alocando as Matrizes que serao utilizadas. */
   alocaMatriz(n);

   /* Abre o Arquivo e grava (n*n) dados nele. */
   gravaArquivo(nomeArquivo, n);
```

```
    /* Grava os dados do Arquivo acima na Matriz Original. */
    gravaMatriz(nomeArquivo, matriz, n);

    /* Invoca a funcao responsavel pelas threads. */
    thread(t, n);

    /* Gravando os resultados nos arquivos correspondentes. */
    gravaArquivo_Diag1(nomeArquivo, matrizSuperior, n);
    gravaArquivo_Diag2(nomeArquivo, matrizInferior, n);

    /* Liberando a memoria alocada para as Matrizes. */
    free (matriz);
    free (matrizSuperior);
    free (matrizInferior);

    /* Fim da funcao principal. */
    return 0;
}

/**
 * Funcao que as Threads executam. Dividem a Matriz em Superior e Inferior.
 * @Parametro tArg -> Dimensao da Matriz.
 * @return NULL.
 */
void *divideMatriz(void *tArg)
{
    /* Carregando a Struct nesta Funcao. */
    struct new *args = tArg;

    /* Carregando dado da Struct. */
    int n = (args->n);

    /* Divide a Matriz Original em duas: Superior e Inferior. */
    int i, j;
    //Superior
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            if(i > j)
                matrizSuperior[i*n+j] = 0;

            else if (i <= j)
                matrizSuperior[i*n+j] = matriz[i*n+j];
        }
    }
    //Inferior
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            if(i > j)
                matrizInferior[i*n+j] = matriz[i*n+j];

            else if (i <= j)
                matrizInferior[i*n+j] = 0;
        }
    }
    /* Retorno valido para funcao do tipo (void*). */
```

```
        return (NULL);
    }

    /**
     * Funcao para criar e programar o que as Threads devem executar.
     * @Parametro t -> Quantidade de Threads.
     * @Parametro n -> Vetor que contem os argumentos.
     */
    void thread (int t, int n)
    {
        /* Declarando as estruturas e variaveis de contagem de tempo: */
        struct timeval inicio;
        struct timeval fim;
        double tempo;

        /* Inicio da contagem de tempo: */
        gettimeofday(&inicio, 0);

        /* Armazenando a quantidade de Threads. */
        pthread_t threads[t];

        /* Cria e executa as Threads a partir da quantidade informada */
        int i;
        for (i=0; i<t; i++)
            pthread_create(&threads[i], NULL, divideMatriz, (void*)&flag);

        /* Garante que todas as Threads ser達o executadas antes do programar acabar */
        for (i=0; i<t; i++)
            pthread_join(threads[i], NULL);

        /* Fim da contagem de tempo: */
        gettimeofday(&fim,0);

        /* Imprime o resultado: */
        tempo = diferencaTempo_miliseg(inicio, fim);
        printf("Tempo de execucao das Threads para t = %i em:\n", t);
        printf("- Milisegundos: %.2f ms.\n", tempo);
        printf("- Segundos: %f s.\n", tempo/1000);
    }

    /**
     * Funcao para alocar as Matrizes do programa.
     * @Parametro n -> Dimensao da Matriz.
     */
    void alocaMatriz (int n)
    {
        /* Alocando as Matrizes: */
        matriz = (double*) malloc(sizeof(double)*n*n);
        matrizSuperior = (double*) malloc(sizeof(double)*n*n);
        matrizInferior = (double*) malloc(sizeof(double)*n*n);
        //Caso o processo de alocacao falhe:
        if (matriz == NULL || matrizSuperior == NULL || matrizInferior == NULL)
        {
            perror("Falha ao alocar a Matriz");
            exit(EXIT_FAILURE);

    }
}

/**
 * Funcao para calcular a diferenca de tempo entre duas marcacoes, em milisegundos.
 * @Parametro inicio -> Membro da estrutura que marca o inicio da contagem.
 * @Parametro fim -> Membro da estrutura que marca o fim da contagem.
 * @return Resultado em ms.
 */
double diferencaTempo_miliseg(struct timeval inicio, struct timeval fim)
{
    return (fim.tv_sec - inicio.tv_sec) * 1000.0f + (fim.tv_usec - inicio.tv_usec) / 1000.0f;
}
```

# arquivo.h

```
/* Cabecalho de Bibliotecas e Funcoes */
#include <stdio.h> //fscanf(), fprinft(), fclose() e perror()
#include <stdlib.h> //exit(), rand() e srand()
#include <time.h> //Para a função srand()
#include <string.h> //strcat() e strcpy()

/* Gera dados aleatórios. */
double geraDados();

/* Função para gravar dados de um Arquivo na Matriz. */
double gravaMatriz(char nomeArquivo[], double *matriz, int n);

/* Funções para gravar dados em Arquivo. */
void gravaArquivo(char nomeArquivo[], int n);
void gravaArquivo_Diag1(char nomeArquivo[], double *matriz, int n);
void gravaArquivo_Diag2(char nomeArquivo[], double *matriz, int n);
```

# arquivo.c

```c
/* Inclusao do cabecalho de Bibliotecas e Funcoes. */
#include "arquivo.h"

/* Gera valores entre 0 e 1, em ponto flutuante. */
double geraDados()
{
    return 0.00 + ( rand() / ( RAND_MAX / ( 1.00 - 0.00 ) ) );
}

/**
 * Funcao para gravar dados aleatorios em um Arquivo.
 * @Parametro nomeArquivo -> Nome do Arquivo a ser aberto.
 * @Parametro n -> Quantidade de elementos a serem gerados (n * n).
 */
void gravaArquivo(char nomeArquivo[], int n)
{
    /* Adiciona a extensao ".dat" ao nome do Arquivo. */
    char extensao[31];
    strcpy(extensao,nomeArquivo);
    strcat(extensao,".dat");

    /* Abrindo o Arquivo para escrita: */
    FILE *arquivo = fopen(extensao,"wb");
    //Em caso de falha:
    if(arquivo == NULL)
    {
        perror("Falha ao abrir arquivo para gravar dados");
        exit(EXIT_FAILURE);
    }

    /* Utilizando registradores para um acesso mais veloz: */
    register int i;

    /* Garante a geração de números aleatorios: */
    srand(time(NULL));

    /* Gravando dados no Arquivo: */
    for (i=0; i < (n*n); i++)
        fprintf (arquivo, "%lf ", geraDados());

    /* Fecha o arquivo apos gravar os dados. */
    fclose(arquivo);
}

/**
 * Funcao para gravar dados do Arquivo em uma Matriz
 * @Parametro nomeArquivo -> Nome do Arquivo a ser aberto.
 * @Parametro matriz -> Matriz que sera gravada.
 * @Parametro n -> Dimensao da Matriz.
 * @return Matriz preenchida.
 */
double gravaMatriz(char nomeArquivo[], double *matriz, int n)
{
    /* Adiciona a extensao ".dat" ao nome do Arquivo. */
    char extensao[31];
```

```
    strcpy(extensao,nomeArquivo);
    strcat(extensao,".dat");

    /* Abrindo o Arquivo para leitura: */
    FILE *arquivo = fopen(extensao,"rb");
    //Em caso de erro:
    if(arquivo == NULL)
    {
        perror("Falha ao abrir arquivo para leitura de dados");
        exit(EXIT_FAILURE);
    }

    /* Utilizando registradores para um acesso mais veloz: */
    register int i,j;

    /* Gravando dados na Matriz: */
    for (i=0; i<n; i++){  //Percorre linhas.
        for(j=0; j<n; j++){ //Percorre colunas.
            fscanf(arquivo, "%lf ", &matriz[i*n+j]);
        }
        fprintf(arquivo, "%s", "\n");
    }

    /* Fecha o arquivo apos gravar os dados. */
    fclose(arquivo);

    /* Retorna a Matriz preenchida. */
    return *matriz;
}

/**
 * Funcao para gravar dados da Matriz Superior no Arquivo _diag1
 * @Parametro nomeArquivo -> Nome do Arquivo a ser aberto.
 * @Parametro matriz -> Matriz que sera gravada.
 * @Parametro n -> Dimensao da Matriz.
 */
void gravaArquivo_Diag1(char nomeArquivo[], double *matriz, int n)
{
    /* Adiciona a extensao "_diag1.dat" ao nome do Arquivo: */
    char extensao[31];
    strcpy(extensao,nomeArquivo);
    strcat(extensao,"_diag1.dat");

    /* Abrindo o Arquivo para escrita: */
    FILE *arquivo = fopen(extensao,"wb");
    //Em caso de erro:
    if(arquivo == NULL)
    {
        perror("Falha ao abrir arquivo para leitura de dados");
        exit(EXIT_FAILURE);
    }

    /* Utilizando registradores para um acesso mais veloz: */
    register int i,j;
```

```
    /* Gravando a Matriz no Arquivo: */
    for (i=0; i<n; i++){ //Percorre Linhas.
        for(j=0; j<n; j++){ //Percorre Colunas.
            fprintf(arquivo, "%lf ", matriz[i*n+j]);
        }
        fprintf(arquivo, "%s", "\n");
    }

    /* Fecha o arquivo apos gravar os dados. */
    fclose(arquivo);
}

/**
 * Funcao para gravar dados da Matriz Inferior no Arquivo _diag2
 * @Parametro nomeArquivo -> Nome do Arquivo a ser aberto.
 * @Parametro matriz -> Matriz que sera gravada.
 * @Parametro n -> Dimensao da Matriz.
 */
void gravaArquivo_Diag2(char nomeArquivo[], double *matriz, int n){

    /* Adiciona a extensão "_diag2.dat" ao nome do Arquivo: */
    char extensao[31];
    strcpy(extensao,nomeArquivo);
    strcat(extensao,"_diag2.dat");

    /* Abrindo o Arquivo para escrita: */
    FILE *arquivo = fopen(extensao,"wb");
    //Em caso de erro:
    if(arquivo == NULL)
    {
        perror("Falha ao abrir arquivo para leitura de dados");
        exit(EXIT_FAILURE);
    }

    /* Utilizando registradores para um acesso mais veloz: */
    register int i,j;

    /* Gravando a Matriz no Arquivo: */
    for (i=0; i<n; i++){ //Percorre Linhas.
        for(j=0; j<n; j++){ //Percorre Colunas.
            fprintf(arquivo, "%lf ", matriz[i*n+j]);
        }
        fprintf(arquivo, "%s", "\n");
    }

    /* Fecha o arquivo apos gravar os dados. */
    fclose(arquivo);
}
```

# makefile

```
# Arquivo Makefile para compilar o programa de divisao de Matriz em outras duas:
# Superior e Inferior
#
# Declaracoes:
OBJS    = divideMatriz.o arquivo.o
SOURCE  = divideMatriz.c arquivo.c
HEADER  = arquivo.h
OUT = divideMatriz
# Compilador padrao:
CC     = gcc
# Comandos de compilacao:
FLAGS   = -g -c -Wall
LFLAGS     = -lpthread

#Dependencias de compilacao:
all: $(OBJS)
    $(CC) -g $(OBJS) -o $(OUT) $(LFLAGS)

divideMatriz.o: divideMatriz.c
    $(CC) $(FLAGS) divideMatriz.c

arquivo.o: arquivo.c
    $(CC) $(FLAGS) arquivo.c

#Limpando:
clean:
    rm -f $(OBJS) $(OUT)
```