

Faculdade de Tecnologia da Unicamp

Projeto da Disciplina de Sistemas Operacionais

| | |
|--------------------------|--------|
| Leonardo Polizel Martins | 220153 |
| Mariana Ramos dos Santos | 221887 |

Sumário

1. Introdução
2. Descrição da Solução do Problema
3. Instruções para a compilação
4. Vídeo
5. Gráfico
6. Conclusão
7. Link para repositório no GitHub

Introdução

Por meio deste relatório, iremos explicar o código que foi criado para a resolução do problema apresentado (o link para o código fonte está presente no final do relatório), dar as instruções de como o compilar e também apresentar uma conclusão com os resultados obtidos por meio de um gráfico comparando o tempo de processamento do código para diferentes números de threads usados.

Descrição da solução do problema

Para a divisão das matrizes, notamos que, para a divisão da diagonal principal para cima, a condição para vetor não ser 0 é que o número da coluna seja maior ou igual que o número da linha, enquanto para a divisão da diagonal abaixo o número da coluna deve ser abaixo do número da linha.

Tendo isso em vista, criamos um laço for para percorrer a matriz, em que “i” representa o número de linhas e “j” representa o número de colunas, assim para cada divisão, uma função if foi criada para verificar se o vetor cumpria as condições ditas acima, se não, será igualado a 0, nós preferimos igualar a zero ao invés de retirar essa parte dos arquivos finais por motivo de se ter uma melhor visualização do resultado da execução do programa.

```

void *Div1(void *arg){ // Divide a matriz A e coloca a diagonal principal e acima na matriz B;

    for(int i = 0; i<N;i++){
        for(int j=0;j<N;j++){

            if(j>i || j==i){
                B[i][j] = A[i][j];
            }
            else{
                B[i][j] = 0;
            }

        }
    }
}

void *Div2(void *arg){ // Divide a matriz a e coloca a diagonal principal e abaixo na matriz C;

    for(int i = 0; i<N;i++){
        for(int j=0;j<N;j++){

            if(i>j && i!=j){
                C[i][j] = A[i][j];
            }
            else{
                C[i][j] = 0;
            }

        }
    }
}

```

Para a criação das Threads, foi usada a função pthread, utilizando assim um laço for para criar T threads, usando o pthread_create para executar as threads para as 2 divisões (Div1 e Div2). Depois, um laço for com o pthread_join para esperar que a execução das thread fosse finalizada.

Para realizar o cálculo do tempo de execução, utilizamos a função clock da biblioteca <time.h>, como o pedido fosse que calculássemos da criação das threads até o fim da divisão, começamos o cálculo imediatamente antes do laço for com o pthread_create, e terminamos o cálculo imediatamente depois do laço for com o pthread_join.

```

clock_t ini = clock();// Para calcular o tempo em segundos

for (int i=0; i<T;i++){// Laço para criar as threads e chamar as funções de divisão
    arg[i]=i+1;
    pthread_create(&threads[i], NULL, Div1, (void*)&arg[i]);
    pthread_create(&threads[i], NULL, Div2, (void*)&arg[i]);
}

for (int i=0; i<T;i++){// Laço para criar as threads e chamar as funções de divisão
    pthread_join(threads[i],NULL);
}
printf("\nFuncao executou em %f segundos\n", ((double)clock() - ini) / CLOCKS_PER_SEC);

```

Finalmente, para que as matrizes fossem gravadas em arquivos conforme a saída pedida no enunciado do projeto, fizemos uma função void chamada GravaMatriznoArquivo e nela declaramos dois arquivos e duas strings. Usando a função strcpy da biblioteca <string.h> igualamos as strings NomeFinal1 e NomeFinal2 a NomeArq, variável global que nessa fase do código guarda o nome do arquivo de entrada do usuário.

A função strcat, da mesma biblioteca, une essas variáveis às extensões. Após isso criamos arquivos com esses nomes e gravamos neles as novas matrizes.

```

void GravaMatriznoArquivo(){
    FILE *saida1;
    FILE *saida2;
    char NomeFinal1 [100];
    char NomeFinal2 [100];

    strcpy(NomeFinal1, NomeArq);

    strcat(NomeFinal1, ".diag1");

    saida1 = fopen(NomeFinal1, "w");

    for(int i = 0; i<N; i++){
        for (int j = 0; j<N; j++){
            fprintf(saida1, "%lf", B[i][j]);
        }

        strcpy(NomeFinal2, NomeArq);

        strcat(NomeFinal2, ".diag2");

        saida2 = fopen(NomeFinal2, "w");

        for(int i = 0; i<N; i++){
            for (int j = 0; j<N; j++){
                fprintf(saida2, "%lf", C[i][j]);
            }
        }
    }
}

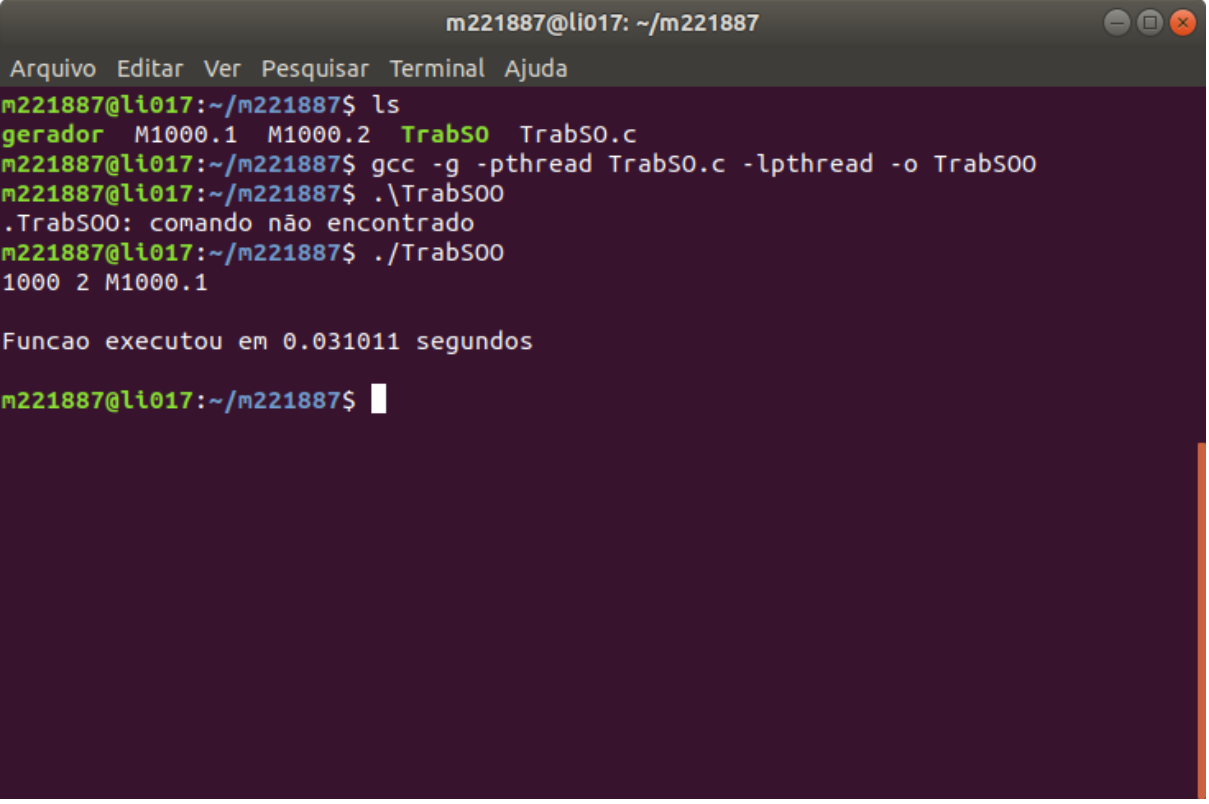
```

Instruções para a compilação

Para a compilação, o usuário deve fazer o download do programa que cria a matriz original e do programa do código (disponível no github do projeto, o qual link está no final do relatório).

Após baixados, o usuário deve colocar ambos na mesma pasta, e então abrir seu compilador (a compilação deve ser realizada em ambiente Linux).

Abaixo, uma imagem passo-a-passo de como realizar a compilação:



```
m221887@li017: ~/m221887
Arquivo Editar Ver Pesquisar Terminal Ajuda
m221887@li017:~/m221887$ ls
gerador M1000.1 M1000.2 TrabSO TrabSO.c
m221887@li017:~/m221887$ gcc -g -pthread TrabSO.c -lpthread -o TrabSOO
m221887@li017:~/m221887$ ./TrabSOO
.TrabSOO: comando não encontrado
m221887@li017:~/m221887$ ./TrabSOO
1000 2 M1000.1

Funcao executou em 0.031011 segundos

m221887@li017:~/m221887$
```

1º comando = ls

Cujo o objetivo é permitir ao usuário a visualização do conteúdo da pasta.

2º comando = gcc -g -pthread TrabSO.c -lpthread -o TrabSOO

É feita a compilação por meio do padrão gcc com -lpthread ao final da linha de comando para incluir a biblioteca POSIX Threads ao resultado da compilação. Os outros comandos foram uma escolha pessoal nossa para nos ajudar no desenvolvimento, onde -g pede informações de depuração e -pthread pede suporte a POSIX Threads.

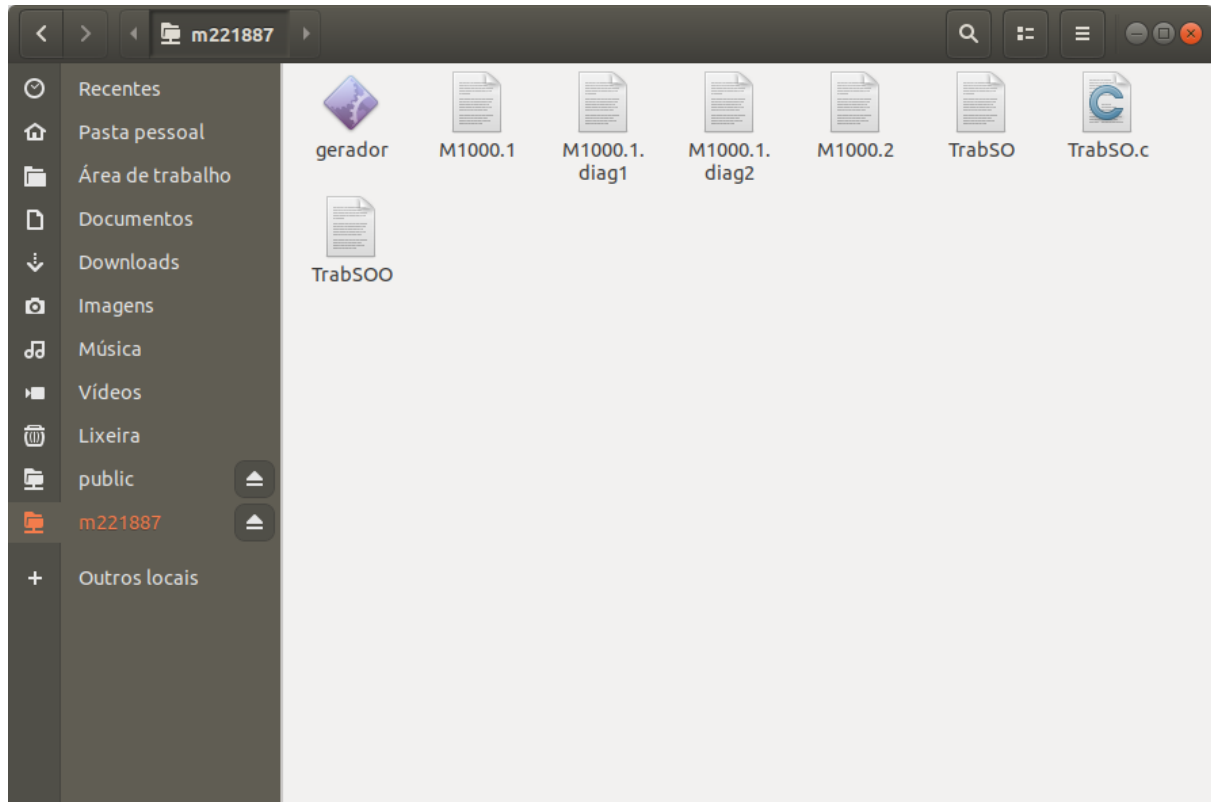
O programa também pode ser tranquilamente compilado com o comando gcc TrabSO.c -lpthread -o TrabSOO.

3º comando = ./TrabSOO

Faz o programa executar.

Entrada do programa = Dimensão da Matriz(no exemplo acima, 1000) Número de Threads(no exemplo acima, 2) e o nome do Arquivo(no exemplo acima,M1000.1)

Após a execução, serão criados os 2 arquivos, o “.diag1” produto da Div1 (Divisão da diagonal principal para cima) e o “.diag2” produto da Div2 (Divisão da parte abaixo da diagonal principal)



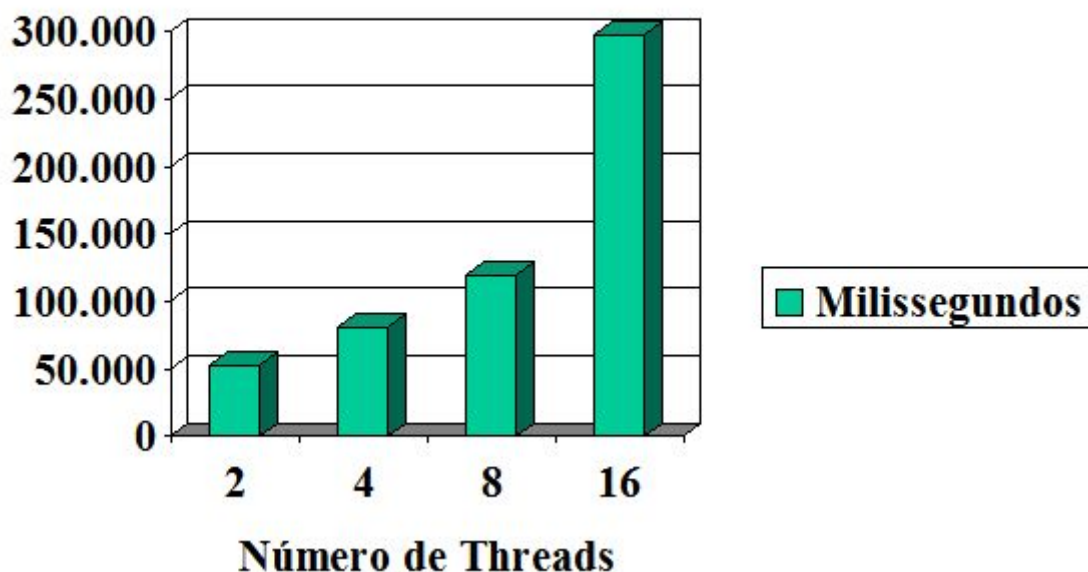
Vídeo

No link abaixo se encontra o vídeo que contém o código sendo compilado e executando alguns testes.

<https://www.youtube.com/watch?v=xDYrEfWj5F8&feature=youtu.be>

Gráfico do tempo de execução das divisões:

Tempo de execução das divisões



Conclusão

Depois de analisar a diferença entre os tempos de execução, é possível notar que ao passo em que mais threads foram usadas, também houve o aumento em milissegundos desse tempo, dessa forma, se conclui que o aumento no número de Threads utilizadas também leva ao aumento do tempo de execução das divisões.

Link do repositório no Github com o código-fonte:

<https://github.com/mariana-ramos/ProjetoDivideMatriz>