Faculdade de Tecnologia

UNIVERSIDADE ESTADUAL DE CAMPINAS
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

PROJETO 1

LOCALIZA NA MATRIZ

SISTEMAS OPERACIONAIS

PARTICIPANTES

Alfredo Albélis Batista Filho - 193532

Cléofas Peres Santos - 195741

Vinícius Abrantes Pereira - 206681

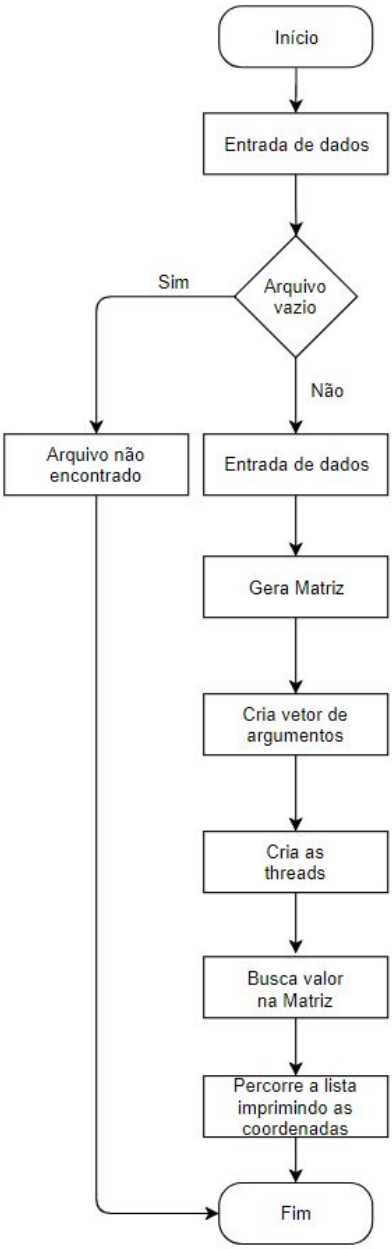
Limeira - SP
2018

DESCRIÇÃO DO PROBLEMA

Este projeto visa a criação de um programa que utilize múltiplas threads para procurar um determinado valor em uma matriz M x N (com M linhas e N colunas) e a análise do desempenho desse programa com 2, 4, 8 e 16 threads.

O programa deverá ser escrito para o sistema operacional Linux e obrigatoriamente utilizar a biblioteca POSIX Threads.

DESCRIÇÃO DA SOLUÇÃO DO PROBLEMA



Entrada de dados

Número de linhas da matriz

Número de colunas da matriz

Nome do arquivo matriz

Entrada de dados

Número de threads

Valor a ser buscado

Vetor de argumentos

Necessário para passar argumentos para a criação das threads

Busca valor na matiz

Percorre a matriz buscando o valor informado e substitui por 0 ou 1.

Substitui-se por:

0 - Caso valor a ser buscado != valor na matriz

1 - Caso Valor a ser buscado = valor na matriz

O fluxograma acima descreve a solução encontrada pelo grupo para o problema proposto. As três principais funções são:

void geraMatriz

* Função responsável por carregar os dados de um arquivo na matriz.

* @param *pfile Ponteiro para arquivo contendo os dados que serão carregados.

* @param num_linhas Número de linhas da matriz.

* @param num_colunas Número de colunas da matriz.

* @param matriz Nome da variável que receberá o arquivo.

Essa função gera uma matriz com as dimensões num_linhas e num_colunas fornecidas pelo usuário e a preenche com os dados contidos no arquivo também informado pelo usuário.

void procuraValor*

* Função para procurar um determinado valor na matriz.

* @param arg contendo as seguintes variáveis:

*num_linhas = num_linhas;

*num_colunas = num_colunas;

*num_thread = i;

*qtd_threads = qtd_threads;

* valor = valor;

*matriz = matriz[0]

Esta é a função que será executada pelas threads. Cada thread percorre uma certa quantidade de linhas da matriz, onde em cada linha a thread substitui o conteúdo do elemento por 0 quando seu valor for diferente do procurado e por 1 quando for igual.

void printCoordenadas

* Função para imprimir coordenadas do número encontrado.

* @param num_linhas Número de linhas da matriz.

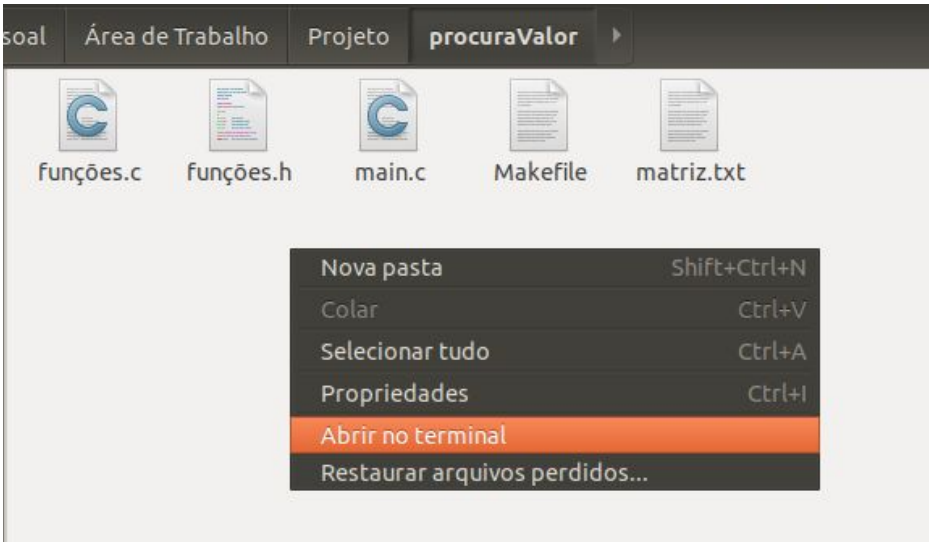
* @param num_colunas Número de colunas da matriz.

* @param matriz Matriz binária.

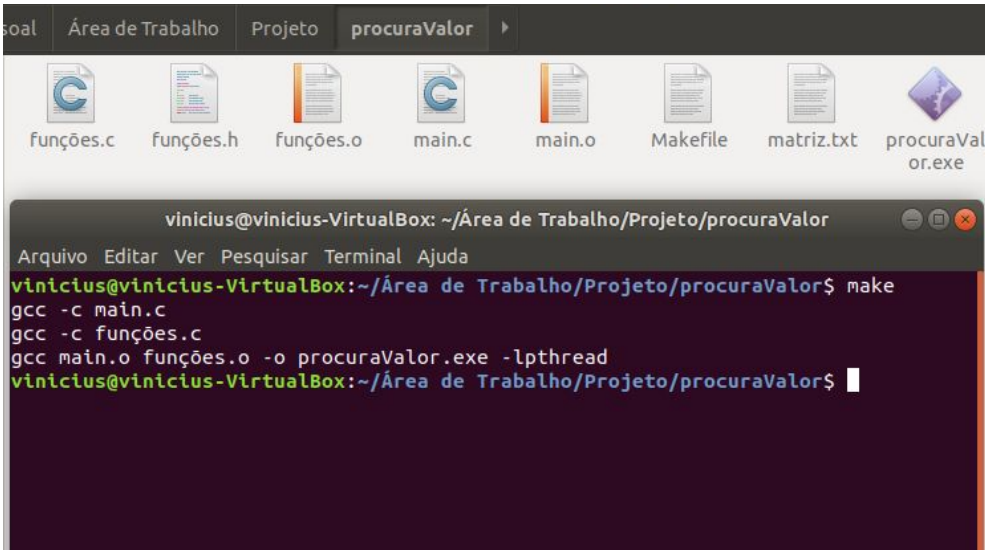
Esta função percorre a matriz esparsa criada pela função procuraValor e imprime as coordenadas do elemento cujo o conteúdo é igual a 1.

INSTRUÇÕES PARA COMPILAR

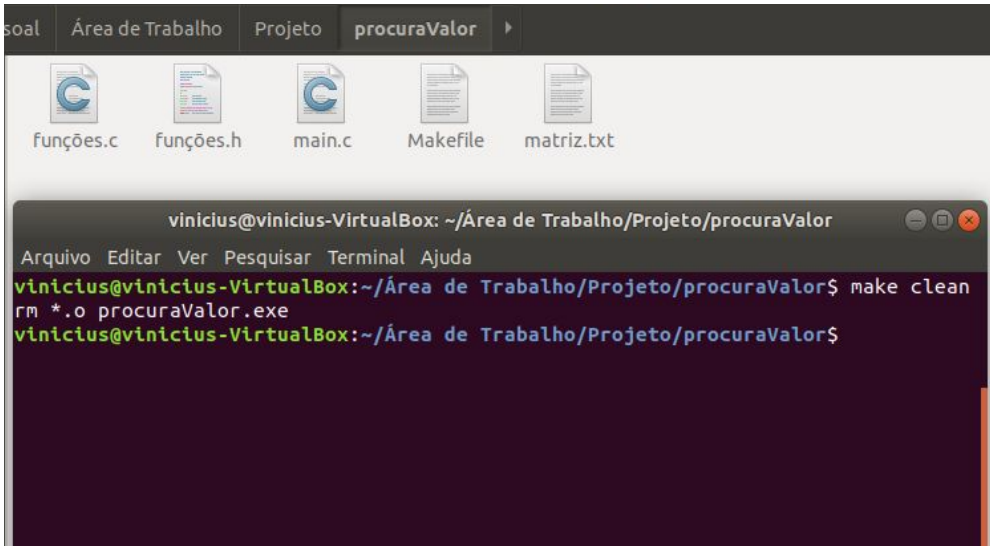
Para simplificar a compilação do código fonte e criação do executável, foi desenvolvido um arquivo makefile. Para utilizá-lo, basta abrir o terminal na pasta que contém os arquivos fonte e digitar o comando ***make***.



Feito isso, será criado o executável chamado *procuraValor.exe*, bem como os arquivos objeto necessários para sua criação.

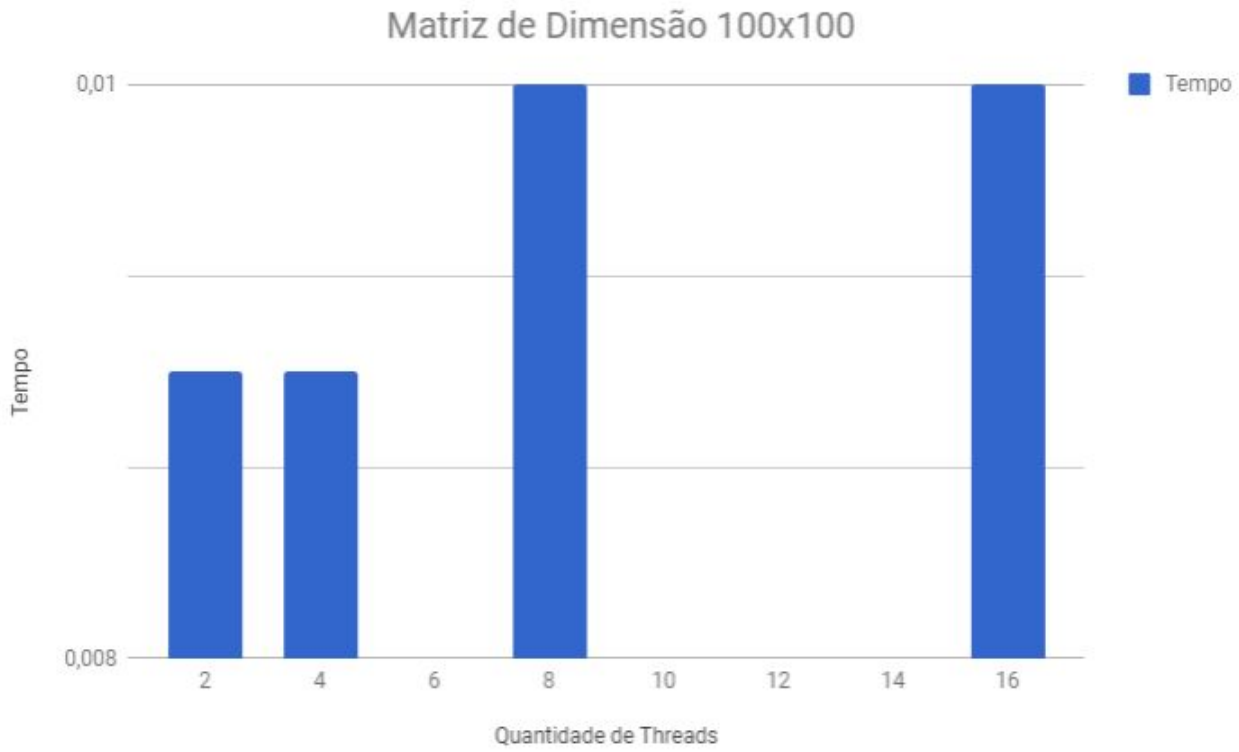


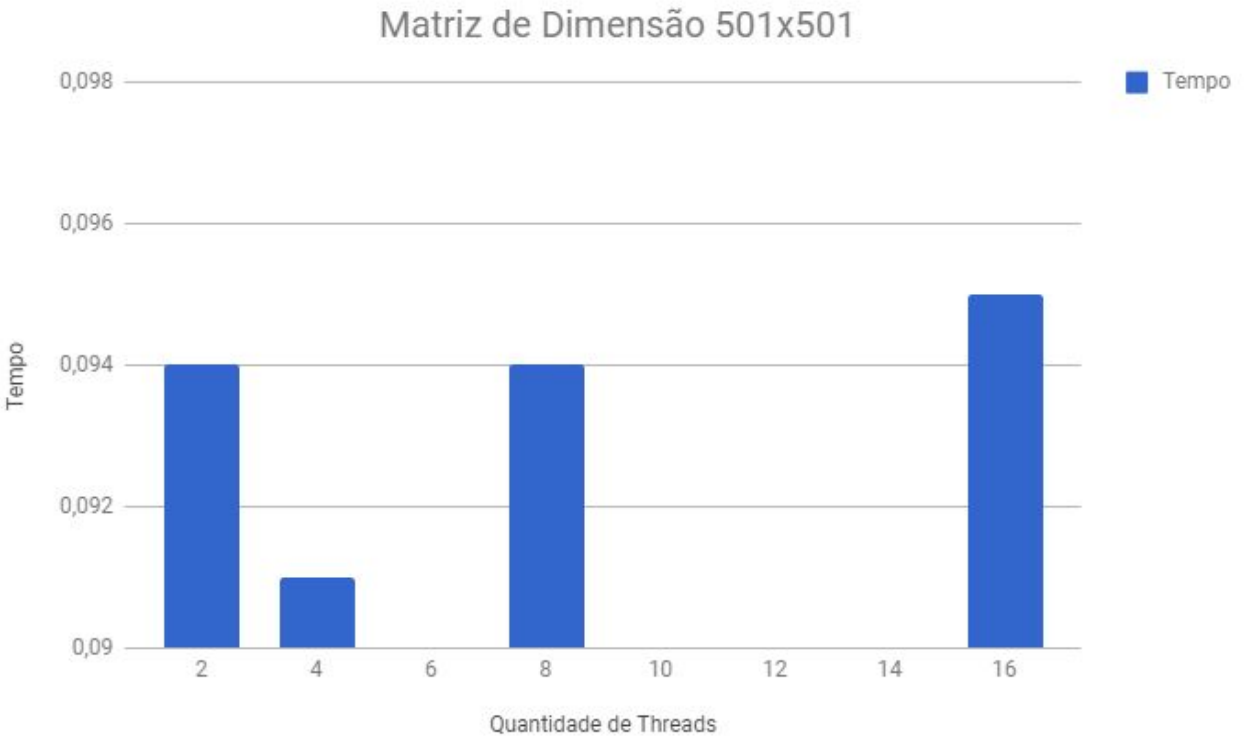
Além do comando make, é possível utilizar o comando ***make clean***, que serve para apagar todos os arquivos gerados anteriormente. Ou seja, este comando apaga todos os arquivos objeto gerados, juntamente com o executável.

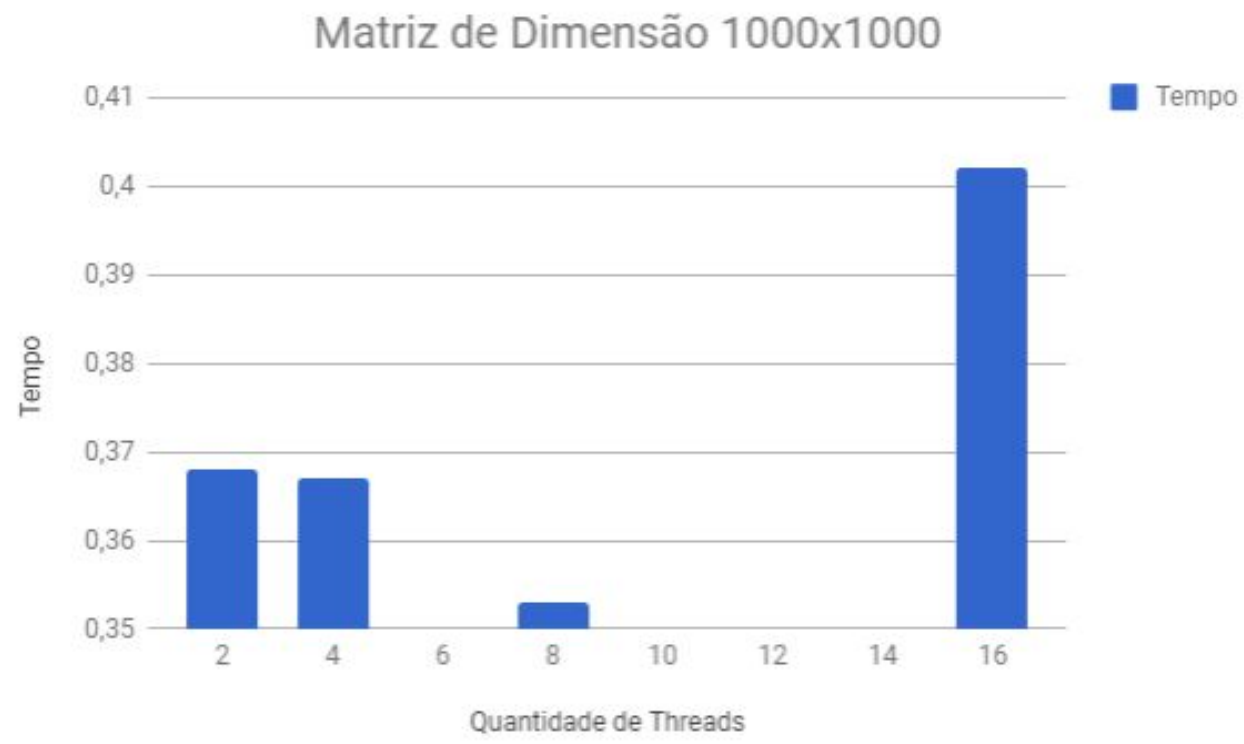


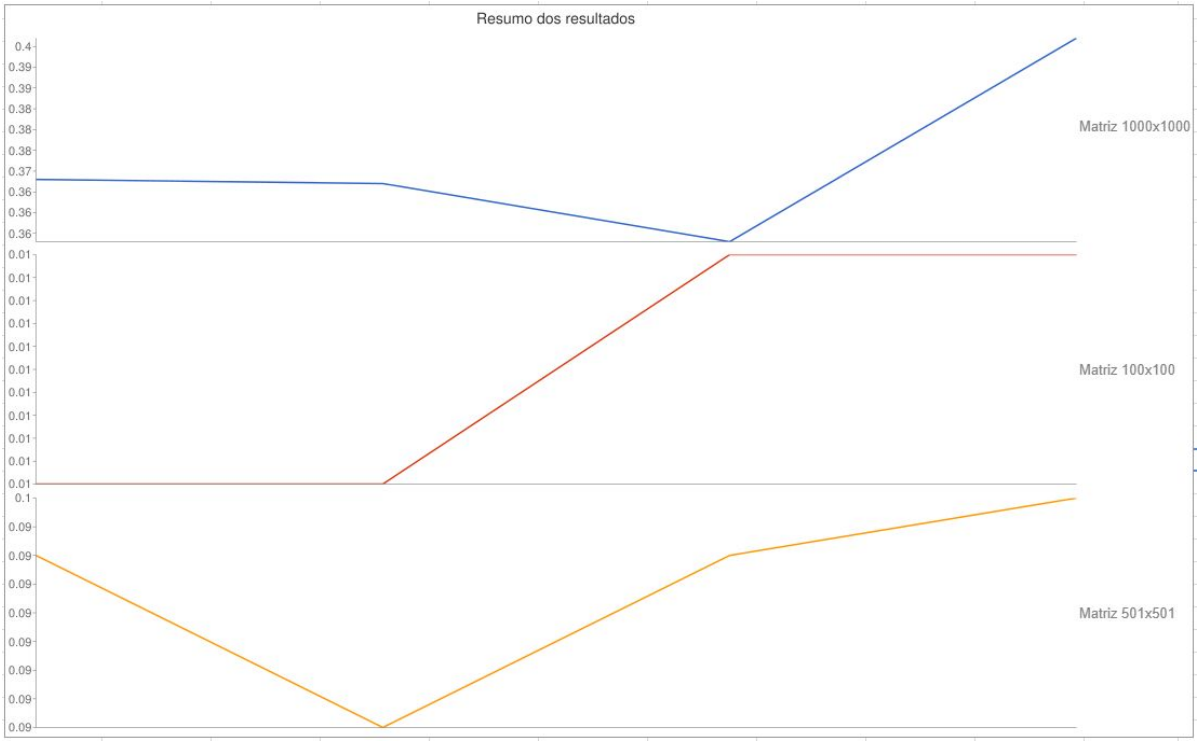
GRÁFICOS COM OS TEMPOS DE EXECUÇÃO DO PROGRAMA

Abaixo se encontram os gráficos dos tempos (em milissegundos) de execução dos testes realizados pelo grupo. Para os testes foram utilizadas matrizes de dimensões variadas, e cada gráfico contém o desempenho de uma certa dimensão utilizando 2, 4, 8 e 16 threads.




Matriz de Dimensão 501x501
Tempo
0,098
0,096
0,094
0,092
0,09
Tempo
2
4
6
8
10
12
14
16
Quantidade de Threads


Matriz de Dimensão 1000x1000
Tempo
0,41
0,4
0,39
0,38
0,37
0,36
0,35
Tempo
2
4
6
8
10
12
14
16
Quantidade de Threads



CONCLUSÃO

A vantagem de se trabalhar com threads é o ganho de desempenho, uma vez que as mesmas são criadas para diminuir o tempo de execução do programa. Nos testes realizados, podemos ver que houve uma diminuição do tempo de execução se compararmos as execuções de 2 e 4 threads. Contudo para uma quantidade de threads superior a 4, a maioria dos testes tiveram um tempo de execução superior.

Após refletir sobre os resultados obtidos, o grupo levantou algumas hipóteses que poderiam explicar o motivo pelo qual uma maior quantidade de threads não implicou em um maior desempenho:

- Quantidade de núcleos: A máquina utilizada para a realização dos testes tem somente dois núcleos físicos e 4 núcleos lógicos. Devido a isso, uma quantidade de threads superior a 4 implicaria em muitos processos concorrendo por processador, que neste contexto se torna um recurso escasso.
- Solução ineficiente: O algoritmo desenvolvido atende e soluciona o problema proposto, porém não é muito eficiente. O objetivo da criação das threads era dividir a matriz para que a procura fosse otimizada. Contudo, na solução desenvolvida, após a execução das threads, a thread principal percorre sozinha toda a matriz, tornando a solução atual ineficiente.

Link do repositório do projeto: https://github.com/viniciusAbrantes/Projeto

Link da planilha com os gráficos: https://goo.gl/NCuSqE