MONSTROS S.O.

GitHub

https://github.com/HeavyBR/Monstros-S.O

YouTube

https://www.youtube.com/watch?v=KSxqcRow4FY

Professor da Disciplina

Dr. André Leon Gradvohl

Integrantes

Matheus Cumpian 222182
Gabriel Velasco 216507
Matheus Rosisca 222360

Sumário

Descrição do Problema

Esse programa tem como objetivo rotacionar uma matriz NxM em 90º no sentido horário utilizando a linguagem C. Como requisito principal o programa deve obrigatoriamente utilizar múltiplas threads para fazer a rotação, com a biblioteca *POSIX threads*.

Exemplo:

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

O PROGRAMA SÓ FUNCIONA EM SISTEMAS LINUX!

Descrição da Solução do Problema – Algoritmo em Alto Nível

O programa, que executa apenas em Linux, utiliza threads para realizar a rotação de 90° da matriz informada pelo usuário.

Ao iniciar o programa verificado se o número de parâmetros informados na execução é válido, se for inválido uma mensagem aparecerá informando o erro ao usuário e será mostrado as opções válidas para uma próxima execução, o mesmo acontece com o número de threads, o programa irá verificar se o número de threads é 2, 4, 8 ou 16 para continuar a execução.

Após isso uma função que percorre todo o arquivo informado e salva os dados lidos em uma matriz alocada dinamicamente é executada.

É criado um vetor de threads e uma struct de argumentos com o número de threads a serem executadas. A próxima etapa é a divisão das colunas que cada thread irá girar e o armazenamento desses valores em sua struct, portanto thread[i] receberá arguments[i].

Cada thread então é criada com a função de rotacionar e os argumentos que ela utilizará. A função de rotacionar consiste em percorrer as colunas indicadas como argumentos para cada thread, ou seja cada thread tem uma coluna inicial e uma coluna final, ler o valor escrito em cada posição e realizar um cálculo de qual será a posição deste valor na matriz rotacionada e salvá-lo no lugar indicado.

Após todas as threads executarem a parte do rotacionamento da matriz existe um join que aguarda todas encerrarem e voltarem para a execução linear da main() do programa e passar para a próxima parte.

Depois que todas as threads são encerradas a função que imprime os dados no arquivo de saída é executada.

Instruções de Compilação

1. Clone o repositório para seu computador
2. Navegue até a pasta ProjetoGrupo
3. Dê permissão para execução do script com o comando

```
chmod a+x run.sh
```

4. Rode o script de compilação com o comando

```
./run.sh
```

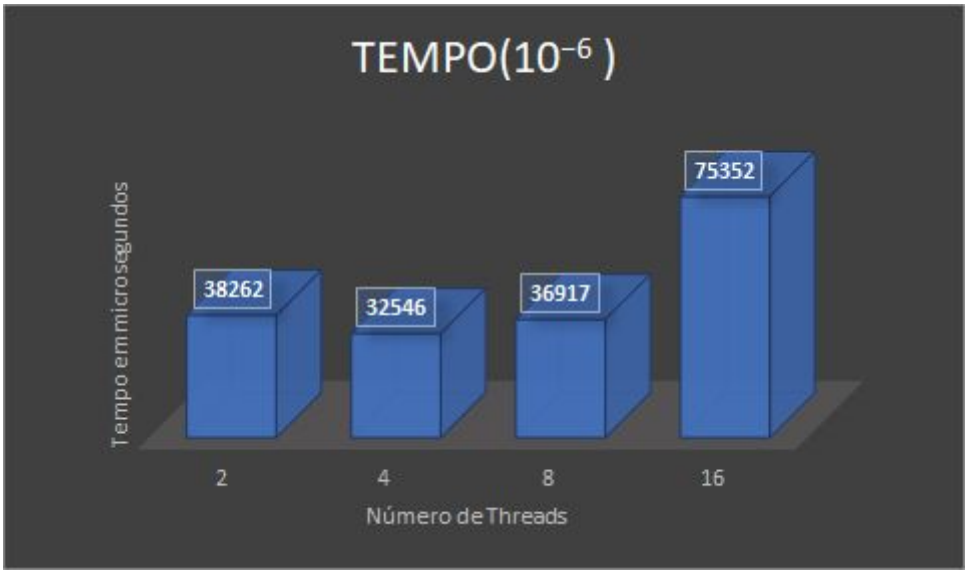
5. Observe as intruções de execução com o comando

```
./giramatriz
```

Importante ressaltar que dependendo da sua distribuição Linux você deverá trocar o -pthread por -lpthread dentro do arquivo run.sh.

Tempos de Execução (Matriz 1000x1000)

Matriz 1000x1000	
Threads	Tempo (µs)
2	38262
4	32546
8	36917
16	75352



Conclusões

Percebe-se que para matrizes 1000x1000, a maior velocidade ocorreu quando rotacionada com 4 threads, porém é importante observar que 4 threads e 8 threads saíram na frente de 2 threads, provando o ponto que o paralelismo das threads impactou positivamente na execução do programa. Entretanto, quando observado o resultado de 16 threads percebe-se que o tempo de execução aumentou demasiadamente comparado a qualquer outra quantidade de threads, nossos testes foram feitos utilizando um processador AMD FX 6300 de 3 núcleos reais e 6 virtuais, porém em nossa configuração da máquina virtual Ubuntu foi setado um número de 4 núcleos para a VM, dito isso uma das hipóteses que temos é que múltiplas threads tomam

vantagens de processadores multi-núcleos, pois ocorre o cada núcleo pode ficar com “n-threads” e isso diminui ainda mais a troca de contexto da CPU, acreditamos que para 4 threads utilizamos o máximo de vantagem dos 4 núcleos do processador, já com 8 e 16 a complexidade aumentou exponencialmente, ficando mais difícil para o SO organizar o paralelismo. Ademais um número muito grandes de threads podem fazer com que as mesmas interfiram negativamente na divisão de recursos, diminuindo o desempenho.