UNIVERSIDADE ESTADUAL DE CAMPINAS

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

SISTEMAS OPERACIONAIS

JOÃO VITOR D'ALKMIN BASSO

KAUAN DA SILVA

VINÍCIUS AUGUSTO PRATES LOURENÇO

GitHub: https://github.com/kauan1/forasteiros

LIMEIRA

2019

- **Solução do Problema**

Começando com a criação da struct ‘ma’ global, ela servirá para armazenar os dados necessário para a rotação da matriz(matriz principal, número total de linhas, colunas e threads) e as variáveis também globais ‘aux1’ para contar a quantidade de threads e quantas vez a função ‘inverter’ é chamada.

Na função ‘main’ iniciamos com a criação das variáveis a serem usadas sendo elas, um ponteiro para a ‘struct’, depois 2 ponteiros para arquivos, um para o arquivo de entrada e outro para o de saída, 2 variáveis de clocks e uma variavel do tipo double.

As variáveis recebem os valores necessários no momento da execução, exemplo: ‘./s.o 100[número de linhas] 100[número de colunas] 8[número de threads] m.txt[matriz principal(arquivo de entrada)] inv8.txt[matriz invertida(arquivo de saída)]’.

Decidimos trabalhar com um vetor que se porta com matriz por ser mais rápida a alocação, alocamos o vetor principal ‘m->m’ e o que receberá a rotação ‘inv’ como mostra no site <https://gradvohl.github.io/alocaMatrizes/?fbclid=IwAR253i2rKATmMn7tgsQP3u_8IjI0zbCl5INkXekz-ogLBOhAStYn3V1A4rA >.

Criamos um vetor ‘vetT’ que receberá cada posição o identificador da thread quando for criada uma thread.

É chamada a função ‘matriz’, que recebe uma struct variável da struct ‘ma’ e o nome do arquivo de entrada. A função colocará os valores do arquivo de entrada no vetor ‘m->m’.

Iniciamos o contador de tempo e começamos a criação das threads pelo ‘for’ de acordo com o número de threads informada pelo o usuário, já na função ‘pthread_create’ o primeiro parâmetro informado é o ‘vetT’, o segundo ‘NULL’, o terceiro a função que a thread funcionará ‘inverter’ e a quarta é variável da struct ‘m’.

Na função ‘inverter’ que é onde as threads trabalharam, começa com a criação das variáveis necessárias para a inversão, criamo uma variável ‘m’ local da struct ‘*ma’, que recebe ‘*info’. Usamos 2 ‘for’ para percorremos as posições desejadas dos vetor onde ‘k’ é igual a linha e ‘a’ igual a coluna.

Exemplo de como a função ‘inverter’ deve funcionar, uma matriz 2x4 e 2 threads, a thread 0 começa na ‘m->m’[0][0] e a thread 1 ‘m->m’[0][1], depois thread

0 ‘m->m’[0][2] e thread 1 ‘m->m’[0][3], a coluna vai sendo acrescentado o número de threads escolhidas no início, e a linha vai de um em um.

thread 0 ‘m->m’[0][0] thread 1 ‘m->m’[0][1]

thread 0 ‘m->m’[0][2] thread 1 ‘m->m’[0][3]

thread 0 ‘m->m’[1][0] thread 1 ‘m->m’[1][1]

thread 0 ‘m->m’[1][2] thread 1 ‘m->m’[1][3]

Após cada thread fazer seu papel passa pela última função em ‘inverter’, ‘pthread_exit’ que sai da thread.

Voltando a função ‘main’ depois que todas as threads foram executadas a função ‘pthread_join’ finaliza as threads recebendo o identificador de cada thread criada, e logo após a última thread eliminada, o contador de tempo é finalizado e fazemos um conta que retorna o tempo em milissegundos e imprime na tela.

Depois da matriz rotacionada e as threads finalizadas chamamos a função ‘imprime’ que salva a rotação no arquivo de saída. E para finalizar a função ‘apagar’ desaloca e libera memória que está sendo usada para alocar dinamicamente as matrizes.

- ## Instruções de Compilação

Para a compilação do código no ambiente Linux é bem simples, basta aplicar no terminal a linha *gcc -pthread programa.c -o programa.o* em seguida a implementação do código *./programa.o 1000 1000 4 matrizentrada.txt matrizsaida.txt* para que sua execução seja por fim completada.

- ## Tempo de Execução

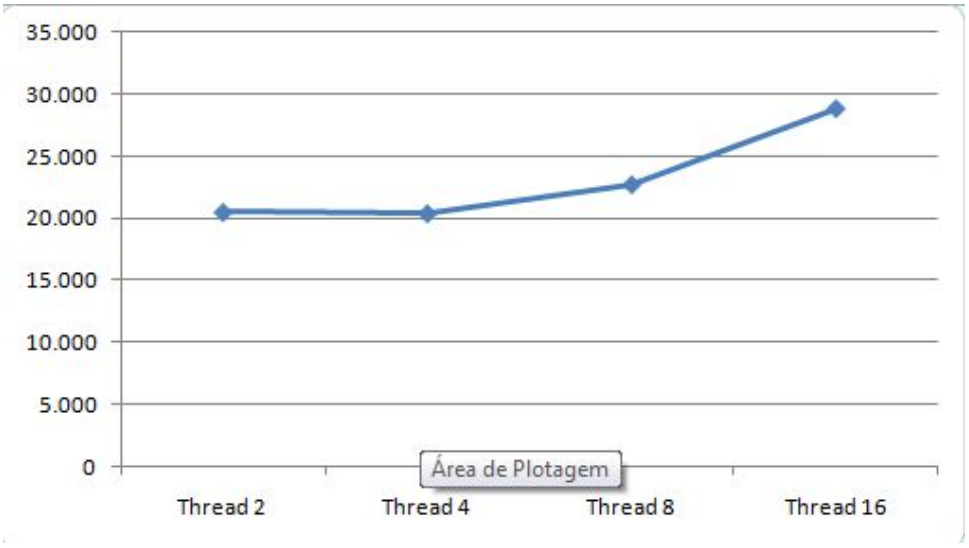

Figura 1 – Gráfico do tempo de execução do programa com 2, 4, 8 e 16 threads.

Com base na execução do programa com as diferentes quantidades, foi gerado o gráfico acima, onde na vertical à esquerda está localizado o tempo em milissegundos (ms) e na parte inferior estão os números de threads, seguindo respectivamente os pontos plotados.

Os valores finais de cada thread são:

2 threads - 20.508

4 threads - 20.405

8 threads - 22.721

16 threads - 28.816

- **Conclusão**

Por fim, conceituamos o grau de necessidade da implementação das threads no problema apresentado onde um de seus desafios era a observação de variações no sistema com essas aplicações lógicas em sua estrutura.

Concluímos por meio de teste, que por sua vez gerou o gráfico apresentado anteriormente, que o incremento das threads para este tipo de programa não é compensatório, pois seu tempo de criação ultrapassa o tempo de execução do programa, tornando-o de certa forma ineficiente, portanto prejudicando a otimização das execuções feitas pelo sistema.

Link do vídeo Projeto “Rotaciona Matriz” - Sistemas Operacionais

https://www.youtube.com/watch?v=FCYS6PGBBI8
