Universidade Federal do Estado do Rio de Janeiro - UFRJ
Bacharelado em Ciência da Computação - IC
Programação Concorrente 2023.1

VITÓRIA DE SOUZA SERAFIM - 120035369

1 Introdução

Este relatório tem como objetivo demonstrar como dois programas distintos, um concorrente e outro sequencial, com o mesmo objetivo, funcionam e quais as diferenças de desempenho entre cada um deles. A ideia é analisar como os programas funcionam para diferentes parâmetros de entrada e como isso afeta a execução dos códigos construídos.

2 Códigos

Os códigos mencionados neste relatório estão disponíveis no repositório do GitHub de Programação Concorrente, cujo link foi incluído no formulário de entrega do trabalho e pode ser encontrado no final deste relatório.

O objetivo desse relatório é avaliar e comparar o desempenho do programa em diferentes cenários de execução. Ao medir a aceleração e a eficiência do programa em relação a diferentes configurações de hardware, tamanhos de entrada ou implementações algorítmicas, buscamos identificar quão eficazmente o programa utiliza os recursos disponíveis e como ele se comporta em relação a variações nas condições de execução. Isso ajuda os desenvolvedores e pesquisadores a entenderem o impacto de otimizações ou paralelizações, além de fornecer "visões"sobre o desempenho geral do sistema em diferentes contextos de uso.

Portanto, para compreender o funcionamento dos nossos programas, dividiremos nossa análise em duas partes: primeiro, discutiremos como executar cada programa e, em seguida, examinaremos as diferenaças nos resultados obtidos.

2.1 Modelo Sequencial:

Para a correta execução do programa devemos inserir alguns parâmetros necessários. Os parâmetros necessários são duas matrizes (salvas em arquivos binários) e um arquivo do tipo '.bin' para guardamos o resultado da multiplicação. Além disso, é interessante comentar que no código existem algumas verificações caso as matrizes não respeitem o tamanho para serem multiplicadas. Um exemplo é dado a seguir:

```
#!/bin/bash
./multi_seq <Matriz1.bin> <Matriz2.bin> <Resultado.bin>
```

Onde:

- "Matriz1.bin"e a "Matriz2.bin"são as matrizes que serão multiplicada pelo programa.

- "Resultado.bin"é o arquivo binário resultado após a operação.

2.2 Modelo Concorrente:

De maneira análoga ao programa anterior, os parâmetros necessários para a versão concorrente são duas matrizes (salvas em arquivos binários) e um arquivo do tipo '.bin' para guardamos o resultado da multiplicação, porém agora devemos informar a quantidade de threads que irão executar a tarefa. Um exemplo é dado a seguir:

```
#!/bin/bash
./multi_conc <Matriz1.bin> <Matriz2.bin> <N_threads> <Resultado.bin>
```

Onde:

- "Matriz1.bin"e a "Matriz2.bin"são as matrizes que serão multiplicada pelo programa.
- "N Threads"é a quantidade de threads que o usuário escolherá para executar a tarefa.
- "Resultado.bin"é o arquivo binário resultado após a operação.

Além disso, os resultados estão formatados para imprimir no formato de planilha ".csv". Portanto, ao final da linha de comando mencionada anteriormente, é necessário adicionar " >> PlanilhaResultado.csv ".

3 Avaliação

A diferença fundamental entre códigos concorrentes e sequenciais reside na forma como as tarefas são executadas e coordenadas. Em um código sequencial, as instruções são executadas uma após a outra, em uma única linha de execução, enquanto em um código concorrente, múltiplas tarefas são executadas simultaneamente, cada uma em sua própria linha de execução. Essa capacidade de executar várias tarefas ao mesmo tempo, pode resultar em uma melhor utilização de recursos e melhorar potencialmente o desempenho da nossa aplicação.

E é isso que será observado em nossos testes: como os códigos desenvolvidos para realizar a multiplicação de matrizes se comportam com diferentes conjuntos de parâmetros. Nosso foco estará em analisar se há uma melhora ou piora no desempenho. Para os testes, utilizaremos matrizes de entrada com dimensões de 500x500, 1000x1000 e 2000x2000. Além disso, executaremos os testes com uma sequência de 1, 2, 4 e 8 threads nos casos concorrentes.

3.1 Testes e coleta de dados

Para os testes concorrentes, usaremos como referência os resultados obtidos da execução do código sequencial. É importante ressaltar que a corretude do código sequencial foi previamente testada. Para validar o funcionamento do programa concorrente, farei uso do comando **diff** no terminal, comparando os resultados produzidos pelo código concorrente com os resultados esperados.

Além disso, na execução do programa concorrente, foi empregado um comando de loop em ***bash***. Essa abordagem visa evitar a repetição na inserção dos parâmetros

necessários para a multiplicação das matrizes, simplificando a coleta de dados para calcularmos a aceleração e eficiência dos programas.

Ao concluir todas as execuções necessárias, teremos como resultado três planilhas .csv contendo os dados correspondentes conforme às dimensões das matrizes e aos números de threads mencionados anteriormente. Com essas planilhas montaremos gráficos para visualizar o comportamento dos programas.

3.2 Tempo de execução, Aceleração e Eficiência

Após a execução e coleta das informações de cada execução, foi analisado o comportamento de cada programa. Neste caso de teste, consideramos o tempo de execução, a aceleração e a eficiência como os principais pontos de análise. Notamos que os tempos de inicialização e finalização são significativamente menores em comparação com o tempo de execução. Portanto, para comparar os tempos dos programas, utilizamos o tempo total dessas partes, o que não afeta o resultado e funciona como um ruído nos dados. Como podemos ver a seguir:

Tamanho das matrizes	Tempo Sequencial	Tempo 1 Thread	Tempo 2 Threads	Tempo 4 Threads	Tempo 8 Threads
500x500	0,78575	0,7955	0,45375	0,38925	0,3295
1000x1000	7,9015	7,57925	4,3555	2,203	2,324333333
2000x2000	79,015	84,17975	38,08875	23,94875	22,37425

Figura 1: Tempos de execução para diferentes matrizes em segundos

Uma observação interessante é que esses programas foram testados em dois computadores distintos, resultando em tempos de execução diferentes para cada máquina. No entanto, ao analisar a aceleração e a eficiência, ambos os resultados apontaram na mesma direção. Assim, é importante observar o gráfico de aceleração obtido (Lembrando que foi definido como aceleração Tempo sequencial/Tempo concorrente):

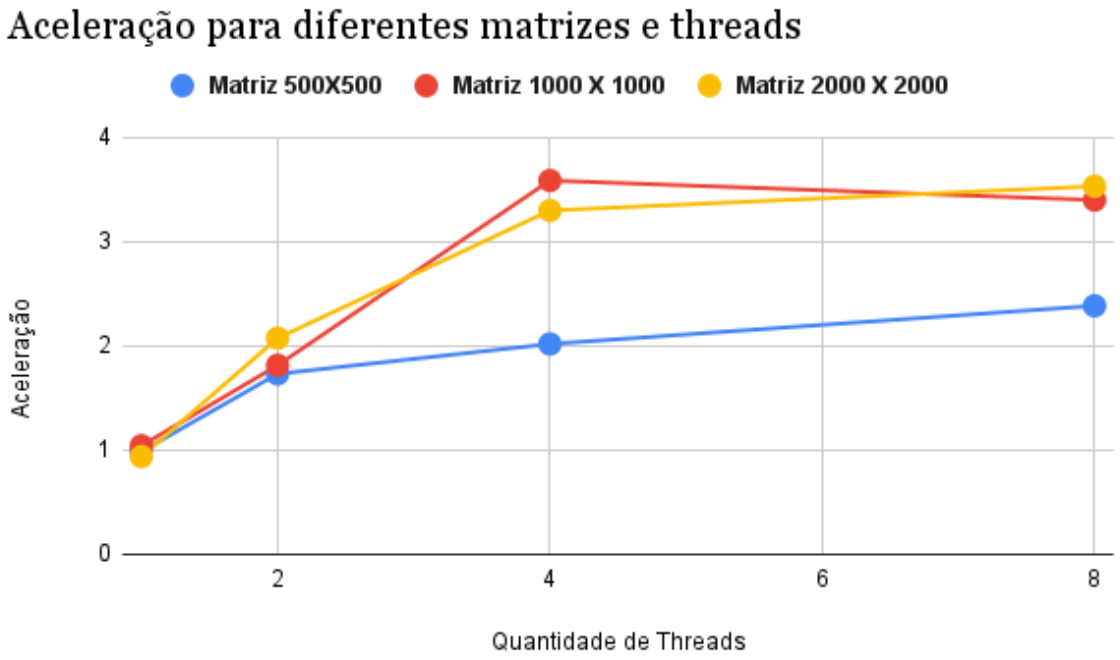

Figura 2: Gráfico das acelerações

Ao examinar o gráfico acima, é perceptível a diminuição da velocidade dos programas concorrentes em matrizes com dimensões menores que a de 2000x2000. Esse comportamento pode ser explicado com alguns problemas na concorrência do programa, um deles é o possível overhead da paralelização, onde o custo associado à divisão do trabalho entre várias threads pode superar os benefícios da paralelização para problemas menores. Outros fatores, como um possível desequilíbrio na carga de trabalho entre as threads, também podem contribuir para a perda de desempenho. Além disso, a sobrecarga do sistema operacional ao lidar com muitas threads para problemas menores pode afetar negativamente o desempenho do programa concorrente.

Olhando para o gráfico das eficiências dos algoritmos, também podemos perceber que o desempenho da operação de multiplicação para as matrizes 2000x2000 é superior às matrizes menores. O tempo necessário para criar e destruir threads pode representar uma porção significativa do tempo total de execução em matrizes "pequenas", pois o trabalho a ser distribuído entre as threads é menor, isto é, em processos grandes existe uma utilização mais eficaz dos recursos do sistema e uma redução proporcional do overhead de paralelização.

Conforme aumentamos o número de threads, essa diferença se torna mais evidente no gráfico, isso deve-se à maior quantidade de trabalho disponível para ser distribuída entre elas. Consequentemente, o custo associado à criação e destruição das threads diminui à medida que a carga de trabalho aumenta. Podemos ver isso no gráfico a seguir:

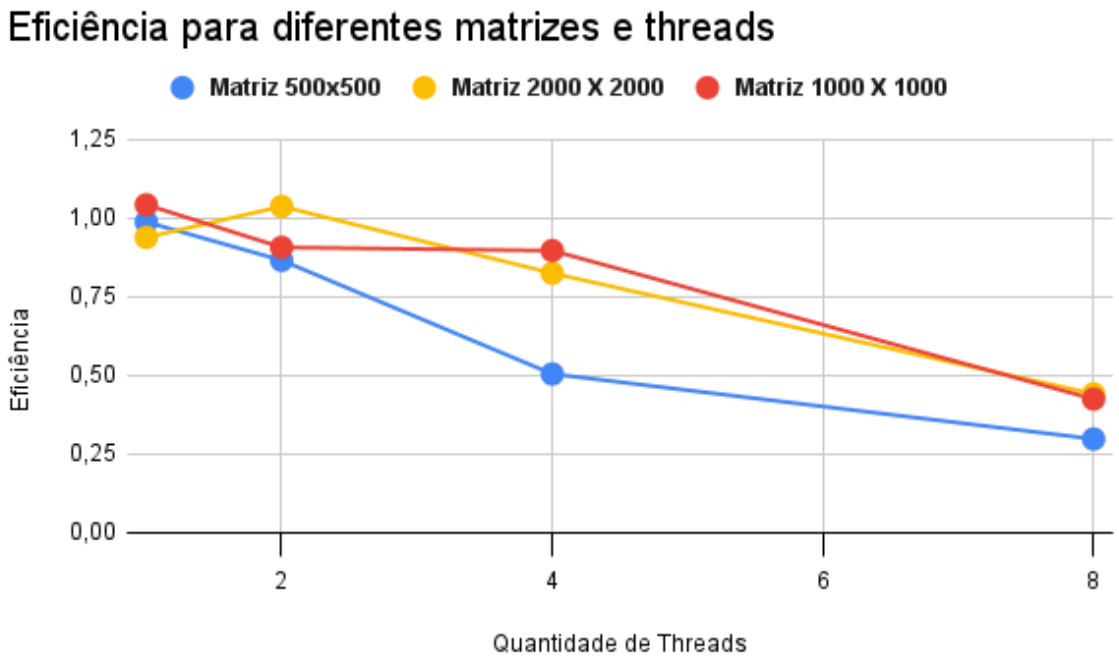

Figura 3: Gráfico das eficiências

4 Conclusão

Concluindo, conforme visto durante todo o relatório, é esperado que o programa concorrente apresente um tempo médio de execução mais rápido, especialmente em cenários com múltiplas tarefas, onde a divisão de trabalho entre as threads resulta em economia de tempo. No entanto, é importante destacar que o programa sequencial muitas vezes executa a tarefa mais rapidamente do que o programa concorrente, mesmo com apenas uma thread. Isso pode ocorrer pela execução desnecessária de blocos adicionais quando há apenas uma thread. Por outro lado, o programa sequencial calcula a multiplicação de forma mais direta e rápida, sem a sobrecarga adicional associada à concorrência. Essa observação ressalta a importância de considerar não apenas o tempo médio de execução, mas também o contexto específico de cada implementação ao avaliar o desempenho de programas sequenciais e concorrentes. Além disso, sabendo que o objetivo desse trabalho não é validar a corretude dos códigos, mas sim seu desempenho, não foram citados no relatório os detalhes dos códigos, mas eles foram devidamente testados - inclusive os resultados sequenciais serviu de "validador"para os resultados encontrados pelo código concorrente.

5 Configuração da máquina

- **CPU:** Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
- **GPU:** NVIDIA GeForce MX250
- **RAM:** 8GB

6 Links e Anexos

1. Repositório com o código feito no laboratório
2. Documento do Laboratório