Multiplicação de Matrizes utilizando *threads*

Thiago I. Yasunaka RA:103069

***Resumo*—Programação concorrente envolve conceitos que tem como principal objetivo otimizar o processamento de forma que, para um problema qualquer, seja possível solucionar suas partes e ao final dela juntá-las, construindo assim, o resultado final. Com o problema da multiplicação de matrizes quadradas, é possível obter resultados que validam essa melhora na performance, visto que esse problema possui complexidade de ordem cúbica e torna mais simples a implementação e obtenção de resultados por meio de testes.**

I. Introdução

A Capacidade de processamento dos computadores evoluem todos os anos, isto é visivelmente perceptível conforme as companhias especialistas na área desenvolvem ferramentas que melhoram a performance dos dispositivos eletrônicos a todo momento. Existe na área da computação e engenharia um termo chamado de *Lei de Moore*. Segundo *Moore*(1965), a capacidade de processamento de um computador dobra a cada dois anos. A afirmação de *Moore* vem ocorrendo desde então, sendo que isto estimula desafios cada vez maiores por parte das pessoas que o desenvolvem.

O conceito de programação concorrente surge a partir da missão de otimizar a execução de algoritmos, tornando-os mais rápidos e performáticos. O clássico algoritmo de multiplicação de matrizes é uma forma de introduzir os principais assuntos que fazem da programação concorrente ser o que ela é.

22 de Agosto de 2021

II. Programação Concorrente

O principal objetivo da programação concorrente é otimizar algoritmos para que seja possível executá-lo de forma mais rápida, economizando dessa forma, o tempo. A maioria das linguagens de programação atuais são executadas de forma sequencial, isto significa, supondo um algoritmo qualquer, que a próxima instrução do algoritmo só será executada depois que a instrução atual for finalizada. Por exemplo, imagine que um algoritmo esteja procurando itens repetidos em uma lista, dessa forma, o modelo mais comum de fazê-lo é percorrer todos os itens da lista, indo do primeiro item até o último, a fim de encontrar todos estes itens. Perceba que dessa forma ocorre um algoritmo sequencial, pois você estaria observando item a item da lista continuamente, executando o próximo passo somente depois de finalizar o atual, tudo isso na mesma função, por exemplo.

Agora imagine um outro exemplo, a multiplicação de matrizes. Para construir um algoritmo que faz multiplicação de matrizes, seguindo o modelo B * A = C, sendo que, B e A são matrizes quadradas (logo, C será uma matriz quadrada), o custo para um algoritmo sequencial que resolva este problema será de pelo menos de ordem cúbica, ou seja, conforme o tamanho das matrizes aumentam, o tempo de execução delas sobem a uma potência de três. Este tempo de execução pode ser otimizado utilizando o que é conhecido como *threads*.

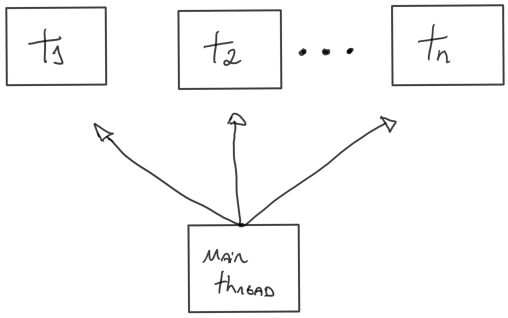

Figura 1: Conceito básico de *Threads*

Olhando a figura 1, é observável que *threads* pode ser visto como um modo de distribuir responsabilidades para outras estruturas além da sua aplicação principal. No exemplo da matriz quadrada, essa distribuição pode ser feita para que cada *thread* calcule somente uma parte da matriz resultante C, por exemplo, suponha que o cálculo esteja acontecendo com duas matrizes quadradas de ordem 4 e que será utilizado duas *threads*, diante disso, seria possível distribuir o cálculo da metade das matrizes para a *thread 1* e a outra metade para a *thread 2*.

Em teoria, para este exemplo, o tempo de execução seria reduzido a metade, se comparado com o algoritmo sendo executado de forma somente sequencial. Na prática, essa "perfeição"não ocorre, pois há diversos fatores que necessitam ser considerados durante uma execução real. Contudo, a performance utilizando os benefícios da programação paralela é consideravelmente melhor do que apenas utilizando a programação sequencial.

III. Metodologia

O cálculo da matriz quadrada requeriu um computador, então para os experimentos realizados, foram utilizados algumas ferramentas que serão descritas na tabela e tópicos abaixo:

- Versão GCC: gcc (Ubuntu 10.3.0-1ubuntu1) 10.3.0

- Biblioteca para threads: *pthread*

Tabela I
CONFIGURAÇÕES DA MÁQUINA

S.O	Ubuntu 21.04
Processador	11th Gen Intel® Core™ i7-1165G7 @ 2.80GHz × 4
Memória RAM	16 GB's
Placa de Vídeo	GeForce MX330/PCIe/SSE2

IV. ANÁLISE

Para apresentar a performance resultante utilizando *threads* e não utilizando *threads*, foi preciso construir um algoritmo sequencial e um algoritmo paralelo. A partir do tempo de execução destes dois algoritmos, é possível obter uma relação que consiste no tempo de execução com a quantidade de *threads* que foi utilizada durante os testes. Observe o gráfico abaixo:

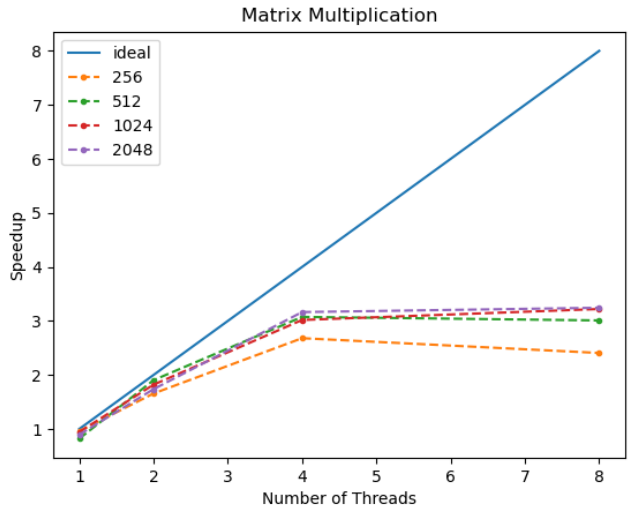

Figura 2: *Speedup* x *Threads*

Para compreender o gráfico, antes é necessário detalhar o que vem a ser o *speedup*. *Speedup* é um cálculo feito para representar a performance do algoritmo sequencial com o algoritmo paralelo, ou seja, é a relação t_sequencial / t_paralelo, em que t_sequencial é o tempo de execução do algoritmo sequencial e t_paralelo é o tempo de execução do algoritmo paralelo.
Dito isso, para que o algoritmo tenha resultados consideráveis é preciso que as retas para cada caso de teste se aproxime ao máximo da reta ideal.
Agora, analisando o gráfico acima, o caso ideal é uma reta linear. Os casos testados foram para os tamanhos de matrizes 256, 512, 1024 e 2048. Para cada tamanho, foram testados as execuções para 1, 2, 4 e 8 *threads*.
De acordo com o gráfico (figura 2), os tamanhos das matrizes não tem grande interferência para o resultado obtido no *speedup*. A maior variância foi para a matriz de tamanho 2048, que ficou um pouco abaixo do resultado se comparado com a média. Analisando as execuções observando-se o número de *threads*, os resultados obtidos para 1 e 2 *threads* foram muito próximos do resultado ideal, confirmando dessa forma a eficiência utilizando-se programação paralela. Conforme o número de *threads* vai aumentando, o resultado obtido vai chegando a níveis um pouco abaixo do ideal, visto que para 4 *threads*, o melhor resultado foi um *speedup* próximo a 3.5. Alguns motivos que podem causar estes valores estão representados abaixo:

- Interferências externas que estão sendo executadas pelo processador
- *Delay* ocasionado pelo hardware do processador
- Lógica do algoritmo

Os resultados obtidos para 8 *threads* foi muito abaixo do ideal, isto ocorre pois a quantidade de núcleos físicos que compõem o processador é menor do que a quantidade de *threads* criadas na execução do algoritmo. Portanto, esta variação já é esperada pois faltam recursos físicos para que o *speedup* seja próximo do valor ideal.

Tabela II
VALORES OBTIDOS PARA OS TESTES REALIZADOS

	Sequencial	1	2	4	8
256	0,04598	0,04792	0,02775	0.01715	0,0191
512	0,3588	0,43714	0,18846	0.11668	0,11922
1024	3,35	3,55	1,84	1,11	1,04
2048	39,68	44,35	22,84	12,54	12,23

threads x tamanho da matriz

A tabela II apresenta os valores obtidos durante os testes, o tempo foi medido em segundos. Ela mostra os valores que foram utilizados para construir a figura 2, e observando a tabela é possível concluir um melhor desempenho utilizando *threads* comparando-se com o algoritmo sequencial.

V. LIMITAÇÕES

Para obter os motivos que ocasionaram determinadas variações nos resultados obtidos, é necessário instalar bibliotecas de terceiros (*PAPI, Pin ou Tau*) para realizar análises no *hardware* do processador. Como o processador que foi realizado os testes não é suportado pela ferramenta, não foi possível fazer coletar estas análises.

VI. CONCLUSÃO

A implementação da multiplicação de matrizes possui um custo alto do ponto de vista computacional, uma forma de otimizar esse processamento é incluir um modelo de programação paralela fazendo com que a multiplicação de matrizes ocorra de forma separada, isto é, o resultado final da matriz é calculado em partes por diferentes *threads* do algoritmo. É possível afirmar que esta solução é uma forma não muito complexa de solucionar, pois este problema não possui a necessidade de trabalhar com problemas assíncrononos, isto significa que as partes do resultado final são independentes de outras partes.

REFERÊNCIAS

[1] D'EMIDIO, Marcelo. AVALIAÇÃO DA LEI DE MOORE E PROPOSTA DE UM MODELO DE PREVISÃO ALTERNATIVO BASEADO EM TÉCNICAS DE EXTRAPOLAÇÃO DE TENDÊNCIAS. São Paulo: Future Studies Research Journal, 2009. Disponível em: https://future.emnuvens.com.br/FSRJ/article/view/11/55. Acesso em: 22 ago. 2021.