Computação Concorrente, 2021.2 - Laboratório 2 - Multiplicação de Matrizes Quadradas - Relatório

David Rodrigues Albuquerque - davidra@dcc.ufrj.br

Novembro, 2021

Sumário

1 Resumo

Este documento tem como por objetivo descrever os resultados obtidos e esperados do laboratório 2. Este laboratório consiste em projetar e executar um algoritmo capaz de realizar, dada uma entrada N = dimensão da matriz quadrada e M = número de threads, o cálculo da multiplicação de matrizes quadradas de tamanho N de forma sequencial e de forma concorrente utilizando as M threads. Ao final, será feita a liberação da memória utilizada pelo programa.

O programa também deve ser capaz de avaliar o desempenho de todas as operações.

2 Configurações da Máquina

Para a execução do algoritmo do cálculo de matrizes quadradas, foi utilizada as seguintes configurações da máquina:

- Arquitetura: x86-64
- CPU: Intel Core i7 10610U - 4 núcleos/8 threads - Frequência 2.3GHz até 4.9 GHz
- Armazenamento: SSD M.2 512 GB
- Memória: 16GB Ram DDR4 3200Mhz

3 Resultado Esperado

Serão utilizados diversos cenário de testes, sendo estes baseados no tamanho da matriz quadrada. Essa matriz quadrada será preenchida com valores de ponto flutuante aleatórios.

- Tamanho 500: Será realizado o cálculo de forma sequencial e de forma concorrente utilizando 1/2/4 threads
- Tamanho 1000: Será realizado o cálculo de forma sequencial e de forma concorrente utilizando 1/2/4 threads
- Tamanho 2000: Será realizado o cálculo de forma sequencial e de forma concorrente utilizando 1/2/4 threads

Cada um desses cenários de testes será executado 5 vezes, de forma a minimizar interferências geradas pelo sistema operacional na velocidade e tirar o tempo de execução mais baixo.

Espera-se que, em casos menores, o tempo de execução do cálculo sequencial seja mais eficiente ou igual, por ser um cálculo pequeno. Desta forma, o cálculo sequencial não desperdiça tempo escalonando as threads. Conforme cresce-se o tamanho da matriz, devemos perceber um desempenho superior na execução concorrente.

4 Código

O código utilizado na execução do programa pode ser obtido no seguinte repositório.

https://github.com/Albuquerque-David/concurrent-computing/tree/main/lab2

5 Resultado obtido

Mínimo encontrado em todos os casos.
Tabela gerada através do result.py.

Tipo	Tamanho: 500	Tamanho: 1000	Tamanho: 2000
Sequencial	0.461109	4.481114	65.480626
Concorrente: 1 Thread	0.457742	4.467366	65.491993
Desempenho: 1 Thread	1.0073556719724213	1.0030774286234885	0.9998264367981595
Concorrente: 2 Thread	0.232515	2.063246	30.618
Desempenho: 2 Threads	1.9831365718340752	2.1718757724478808	2.138631719903325
Concorrente: 4 Thread	0.12611	1.013899	14.639143
Desempenho: 4 Threads	3.656403140115772	4.419684800951574	4.472982195747387
Liberar memória	0.224948	1.78508	14.29819

6 Conclusão

Como esperado, podemos observar que, para valores pequenos da matriz, o cálculo sequencial é equivalente ao desempenho do cálculo realizado de forma concorrente ou levemente superior, como no caso do cálculo sequencial da matriz $N = 2000$. Mas, assim que escalamos o tamanho da matriz, podemos observar mudanças significativas e proporcionais ao tamanho das threads. Quando se utiliza 2 threads, o desempenho tende a dobrar, e quando se usa 4 threads, o desempenho tende a quadruplicar.

Utilizando a Lei de Amdahl nas linhas do desempenho para o cálculo, podemos observar que de fato o paralelismo está sendo realizado, nos fornecendo esse ganho de desempenho.