Relatório de avaliação de desempenho

Autor: Riquelme Freitas Gomes

DRE: 120032785

Neste laboratório, implementei dois programas para o problema de multiplicação de matrizes, um feito de forma sequencial (”lab3_seq.c”) e outro de forma concorrente (”lab3_conc.c”). Ambos foram divididos em três partes (inicialização, processamento e finalização) e os tempos de execução de suas partes foram registrados, assim como o tempo total.

Cada programa recebe como entrada dois arquivos binários contendo as matrizes de entrada (A e B) para a multiplicação, uma matriz em cada arquivo. Na versão concorrente, é recebido também o número de threads a ser utilizado no processamento dos cálculos. Ao fim, a matriz resultante (C) é escrita em outro arquivo binário e os tempos de execução de cada parte do código são exibidos.

Como rodar o programa concorrente:

```
./lab3_conc <nome arquivo 1> <nome arquivo 2> <nome arquivo 3> <número de threads>
```

onde os arquivos 1 e 2 são os arquivos binários de entrada e o arquivo 3 é o arquivo de saída.

Operação principal realizada no programa:

$$A_{lin,\,n} \times B_{n,\,col} = C_{lin,\,col}$$

onde **A** e **B** são matrizes de entrada e **C** é a matriz de saída.

Para comparar os resultados obtidos em cada configuração, utilizei arquivos contendo matrizes de entrada com três tipos de dimensões diferentes: 500x500, 1000x1000 e 2000x2000. Além disso, no caso concorrente, cada dimensão foi executada utilizando quatro número de threads distintos (1, 2, 4 e 8). Cada configuração foi executada três vezes e os valores médios desses tempos foram registrados nas tabelas deste relatório.

A partir das medidas dos tempos foi possível calcular a **aceleração (A)** e a **eficiência (E)** alcançadas por meio da concorrência. Tais valores estão sendo calculados da seguinte forma:

$$A_p = \frac{\text{tempo de processamento sequencial}}{\text{tempo de processamento concorrente com p threads}}$$

Para o cálculo da aceleração, optei por utilizar apenas o tempo de processamento e não o tempo total da execução do programa. Acredito que dessa forma é possível observar mais precisamente o impacto da concorrência na resolução do problema, uma vez que as partes de “inicialização” e “finalização” do código ocorrem da mesma maneira nos dois programas.

$$E = \frac{\text{Aceleração}}{\text{Número de threads}}$$

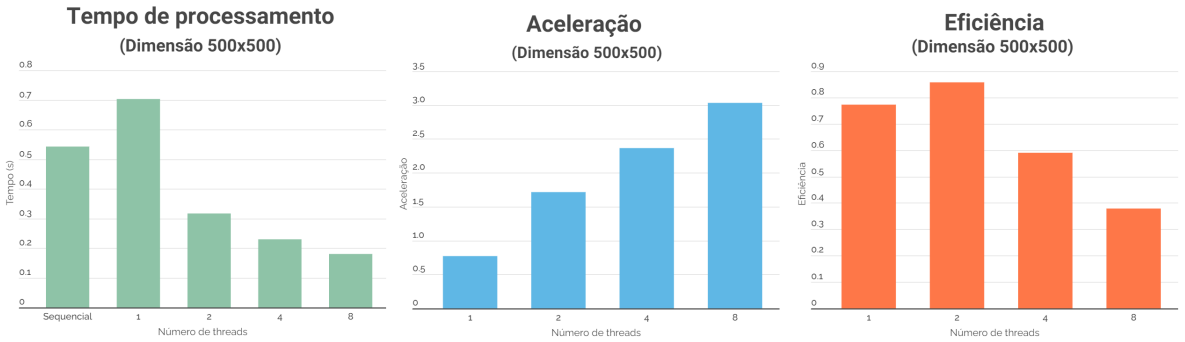
Tempo médio de execução (em segundos) do programa usando matrizes de entrada de dimensões: 500x500, 1000x1000 e 2000x2000.

Dimensões: 500x500

# Número de threads	⧗ Inicialização	⧗ Processamento	⧗ Finalização	∑ Tempo total
Sequencial	0.001329	0.543204	0.001649	0.546182
1	0.001724	0.703498	0.001679	0.706901
2	0.001657	0.316877	0.001335	0.319869
4	0.001766	0.229927	0.000993	0.232686
8	0.001876	0.179345	0.001727	0.182948

Aceleração e eficiência (500x500)

# Número de threads	Processamento	Aceleração	Eficiência
Sequencial	0.543204	1	
1	0.703498	0.772147184498	0.772147184498
2	0.316877	1.714242434762	0.857121217381
4	0.229927	2.362506360714	0.590626590179
8	0.179345	3.028821545067	0.378602693133



Pelos gráficos acima, podemos observar que o programa concorrente com apenas **uma** thread levou mais tempo no processamento do que o programa sequencial. Além disso, vemos que conforme é acrescido o número de threads utilizadas menor fica o tempo gasto no processamento e consequentemente maior é o valor da aceleração. A eficiência apresentou seu maior valor para duas threads.

Dimensões: 1000x1000

# Número de threads	Inicialização	Processamento	Finalização	∑ Tempo total
Sequencial	0.006044	4.695144	0.003135	4.704323
1	0.008387	6.278306	0.006355	6.293048
2	0.006409	3.035315	0.003727	3.045451

# Número de threads	⧖ Inicialização	⧖ Processamento	⧖ Finalização	∑ Tempo total
4	0.006046	2.037159	0.004421	2.047626
8	0.006212	1.704971	0.003469	1.714652

Aceleração e eficiência (1000x1000)

# Número de threads	⧖ Processamento	Aceleração	Eficiência
Sequencial	4.695144	1	
1	6.278306	0.747836120125	0.747836120125
2	3.035315	1.546839125428	0.773419562714
4	2.037159	2.304750881006	0.576187720252
8	1.704971	2.753796985403	0.344224623175

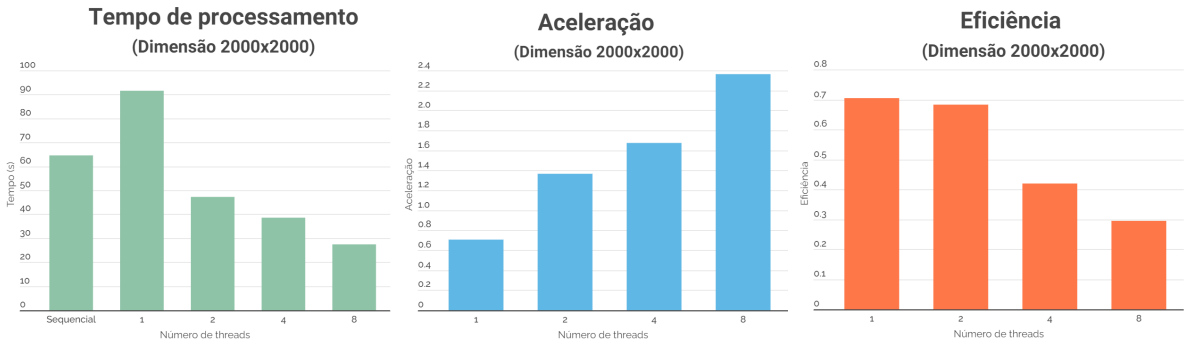
Nesse caso, a mudança para matrizes de entrada com dimensões 1000x1000 não alterou significativamente o comportamento dos gráficos. O tempo de processamento continua tendo uma redução quase pela metade na variação de 1 para 2 threads e a eficiência com 8 threads continua sendo a menor de todas. No entanto, agora as eficiências com 1 e 2 threads apresentam resultados mais próximos que anteriormente.

Dimensões: 2000x2000

# Número de threads	⧗ Inicialização	⧗ Processamento	⧗ Finalização	∑ Tempo total
Sequencial	0.028138	64.654314	0.01501	64.697462
1 thread	0.068456	91.594524	0.010267	91.673247
2 threads	0.045954	47.37556	0.011382	47.432896
4 threads	0.042459	38.571686	0.016663	38.630808
8 threads	0.024382	27.359167	0.012245	27.395794

Aceleração e eficiência (2000x2000)

# Número de threads	⧗ Processamento	Aceleração	Eficiência
Sequencial	64.654314	1	
1	91.594524	0.705875320669	0.705875320669
2	47.37556	1.364718728391	0.682359364196
4	38.571686	1.676211768394	0.419052942099
8	27.359167	2.363168220728	0.295396027591



Por fim, para matrizes de dimensões 2000x2000, vemos que a eficiência com 1 thread se torna um pouco maior que a com 2 threads, embora ambas ainda sejam maiores que com 4 ou 8 threads. Além disso, com o aumento da quantidade de dados, todos os tempos de processamento cresceram significativamente. Porém, em todos os casos os piores tempos são sempre com 1 thread e depois com a versão sequencial.

Configuração da máquina

Processador: Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz, 1190 Mhz, 4 Núcleo(s), 8 Processador(es) Lógico(s)

RAM instalada: 8,00 GB