Primeiro Trabalho - Relatório Final

Comutação Concorrente (MAB-117) — 2021/1 REMOTO

Aplicação concorrente no processo de Eliminação Gaussiana

Carlos A. Cozzolino R.J. e Thiago de Oliveira Silva

[1]*115086800 e 111466197*

1. Projeto final e implementação da solução concorrente

O código aqui apresentado visa calcular a eliminação gaussiana de uma matriz quadrada [n][n] gerada por números aleatórios. O usuário digita a dimensão [n] e a quantidade de threads que vamos utilizar, o programa gera uma matriz, onde as threads realizam o escalonamento, faz substituição reversa para encontrar os valores do vetor x e MMQ para verificar se Ax - b = 0, ou seja, se está tudo certo com a resposta.

2. Casos de teste

O programa foi testado sendo executado com matrizes de tamanhos diferentes preenchidas com números aleatórios. Os valores utilizados para as dimensões das matrizes nos testes foram de ***500, 1500 e 3500*** *e o número de threads variante também em* ***2, 8 e 16*** *para cada dimensão. Para avaliar a corretude da aplicação verificamos se Ax - b = 0, uma vez que sabe-se que*

$$Ax = b$$

3. *Avaliação de desempenho*

Aqui temos algumas medições que foram feitas na máquina gamer do aluno Carlos Cozzolino (processador Xeon 2630L com 8 núcleos e 16 threads). O mesmo caso de teste foi executado 3 vezes para cada número de threads selecionado. O tempo foi calculado em segundos e foi utilizada a média das medições como parâmetro para cálculo do ganho de desempenho. O ganho de desempenho foi calculado da mesma forma que foi apresentado no lab 2.

$$(T_{sequencial} / T_{concorrente})$$

Para matriz de dimensão 500:

1ª medição sequencial: 0.270550 segundos

2ª medição sequencial: 0.271380 segundos

3ª medição sequencial: 0.272781 segundos

1ª medição concorrente com 2 threads: 0.475016 segundos

2ª medição concorrente com 2 threads: 0.483147 segundos

3ª medição concorrente com 2 threads: 0.453408 segundos

1ª medição concorrente com 8 threads: 0.504999 segundos

2ª medição concorrente com 8 threads: 0.509809 segundos

3ª medição concorrente com 8 threads: 0.506152 segundos

1ª medição concorrente com 16 threads: 0.867739 segundos

2ª medição concorrente com 16 threads: 0.874010 segundos

3ª medição concorrente com 16 threads: 0.871277 segundos

Desempenho para 2 threads: 0.577166

Desempenho para 8 threads: 0.535655

Desempenho para 16 threads: 0.311784

Para matriz de dimensão 1500:

1ª medição sequencial: 4.881975 segundos

2ª medição sequencial: 5.578578 segundos

3ª medição sequencial: 5.459147 segundos

1ª medição concorrente com 2 threads: 6.625512 segundos

2ª medição concorrente com 2 threads: 6.554548 segundos

3ª medição concorrente com 2 threads: 6.658968 segundos

1ª medição concorrente com 8 threads: 2.164055 segundos

2ª medição concorrente com 8 threads: 2.119707 segundos

3ª medição concorrente com 8 threads: 2.144198 segundos

1ª medição concorrente com 16 threads: 2.770309 segundos

2ª medição concorrente com 16 threads: 2.788512 segundos

3ª medição concorrente com 16 threads: 2.790594 segundos

Desempenho para 2 threads: 0.802444

Desempenho para 8 threads: 2.476633

Desempenho para 16 threads: 1.906684

Para matriz de dimensão 3500:

1ª medição sequencial: 65.809304 segundos

2ª medição sequencial: 63.131286 segundos

3ª medição sequencial: 61.699948 segundos

1ª medição concorrente com 2 threads: 64.821022 segundos

2ª medição concorrente com 2 threads: 64.273954 segundos

3ª medição concorrente com 2 threads: 63.529732 segundos

1ª medição concorrente com 8 threads: 15.746623 segundos

2ª medição concorrente com 8 threads: 15.742121 segundos

3ª medição concorrente com 8 threads: 14.619333 segundos

1ª medição concorrente com 16 threads: 14.718992 segundos

2ª medição concorrente com 16 threads: 15.627297 segundos

3ª medição concorrente com 16 threads: 14.769650 segundos

Desempenho para 2 threads: 0.989699

Desempenho para 8 threads: 4.134645

Desempenho para 16 threads: 4.225570

4. *Discussão*

Os valores obtidos estão de acordo com o esperado, já que a forma concorrente tende a ser mais ágil conforme a quantidade de dados aumenta e no código aqui presente podemos notar isso. Se continuarmos aumentando os valores, haverá uma diferença cada vez maior entre a concorrente com 8 threads, a com 16 threads e a sequencial. Por exemplo: para uma dimensão de 5000 temos que a sequencial leva algo em torno de 196 segundos, a concorrente com 8 threads leva 42 e a com 16 leva 37 segundos. Uma mudança que poderíamos fazer no programa concorrente para ele ficar ainda mais rápido seria a forma como é feita a varredura pela matriz.

5. *Referências bibliográficas*

Ruggiero, Márcia A. ***Cálculo Numérico: Aspectos Teóricos e Computacionais.*** *2ª Ed. Pearson Universidades. São Paulo. 2000.*

6. Link para o repositório:

https://github.com/Thiolivs/Concurrent-computing/tree/main/Trabalho%201