Calculando quantidade de múltiplos

Computação Concorrente (MAB-117) - 2020/2 REMOTO

Gabriela da Silva Mattos

119048406

Descrição do problema

- O programa tem como objetivo receber uma matriz quadrada preenchida com os números de 1 a (dimensão da matriz * dimensão da matriz) e calcular o número de múltiplos dos números 2, 3, 5, 7 e 11.
- A entrada do programa é a dimensão de uma matriz quadrada e o número de threads que o usuário queira que sejam usadas. A saída gerada será um vetor que armazenará a quantidade de múltiplos de 2, 3, 5, 7 e 11 nessa ordem.

Projeto da solução concorrente

- As estratégias que podem ser utilizadas para dividir a tarefa principal entre fluxos de execução independentes são:
    - Cada thread calcula uma linha, em sequência.
    - Cada thread calcula uma linha e pula o número de threads de linhas, dessa forma cada thread calcula as linhas que forem múltiplas do seu identificador (Essa foi a estratégia que eu usei).
    - Criar uma thread para cada elemento da matriz.

- A estratégia escolhida por mim, foi a segunda que eu mencionei, cada thread calcula uma linha múltipla de seu identificador. Eu usei essa estratégia porque acho que é mais fácil de evitar erros e a melhor em relação ao desempenho do programa. Uma thread por elemento acarreta perda de desempenho, pois as tarefas a serem executadas são muito pequenas, sendo assim não vale a pena criar tantas threads.

- Usei 3 estruturas de dados, uma matriz, que é a matriz de entrada, um vetor, para as threads armazenarem a quantidade dos múltiplos de cada linha da matriz, e um último vetor que armazena as quantidades totais de múltiplos de 2, 3, 5, 7 e 11, nessa ordem.

- Os argumentos passados para as threads foram o identificador da thread no sistema, null para propriedades default, a função a ser executada e seu identificador local. As threads percorriam a matriz e somavam + 1 a respectiva posição no vetor de saída.

Casos de teste

- Para um conjunto de casos teste eu testei o programa para as dimensões de matrizes **500**, **1000**, e **2000** e o número de threads **2**, **3**, e **4**.

- Para testar o funcionamento do programa, eu coloquei printf para verificar se a matriz e os vetores estavam sendo inicializados corretamente, preenchidos corretamente, se as threads estavam funcionando corretamente, cada uma percorrendo a sua parte da matriz e verifiquei se as somas estavam corretas. Em relação a corretude, eu executei 5 vezes cada caso de teste, e obtive sempre o mesmo resultado de acordo com o tamanho da matriz, apenas o tempo era variável.

Avaliação de desempenho

- < Dimensão >< Número de Threads > : 500 2; 500 3; 500 4; 1000 2; 1000 3; 1000 4; 1000 5; 2000 2; 2000 3; 2000 4;
- Sistema operacional: Ubuntu 20.04. 1 LTS; Processador: Intel Core i3-5005U CPU @ 2.00GHz × 4; Capacidade de disco: 500,1GB. A Máquital possui 2 sistemas operacionais, todos os testes foram realizados no Ubuntu, mas não sei se isso afeta diretamente a execução dos programas.
- Foram executadas 5 vezes cada caso de teste e foi escolhido o resultado com o menor tempo.
- O cálculo do **ganho esperado** foi:

  Para matrizes **500x500**, tempos:
  Inicialização: 0.003672
  Para encontrar e contar os múltiplos: 0.009726
  Finalização: 0.000108

  Tsequencial:0,013506
  ts: 0.000378
  tp(1):0.009726

  Tsequencial/ts + tp(P)

  Ganho para 2 processadores: 1.562651
  Ganho para 3 processadores: 1.923383
  Ganho para 4 processadores: 2.174529

  Para matrizes **1000x1000**, tempos:
  Inicialização: 0.009600
  Para encontrar e contar os múltiplos: 0.027703
  Finalização: 0.000360

  Tsequencial:0.0376630
  ts: 0.0132
  tp(1):0.027703

  Tsequencial/ts + tp(P)

  Ganho para 2 processadores: 1.3922702
  Ganho para 3 processadores: 1.6788132
  Ganho para 4 processadores: 1.8713836

Para matrizes **2000x2000**, tempos:
Inicialização: 0.032630
Para encontrar e contar os múltiplos: 0.093126
Finalização: 0.001207

Tsequencial:0.126963
ts: 0.033837
tp(1):0.93126

Tsequencial/ts + tp(P)

Ganho para 2 processadores: 1.579141791
Ganho para 3 processadores: 1.9569198046
Ganho para 4 processadores: 2.2227996

Dimensão da Matriz	Número de threads	Tempo
500	1	0.0135060
500	2	0.007632
500	3	0.007253
500	4	0.006663
1000	1	0.0376630
1000	2	0.024286
1000	3	0.022141
1000	4	0.020072
2000	1	0.1269630
2000	2	0.083131
2000	3	0.070043
2000	4	0,068854

Discussão

- O ganho de desempenho foi bem próximo do esperado, em alguns momentos foi até maior, principalmente com a matriz de dimensão 1000x1000.
- Acho que uma possível melhoria seria tornar a inicialização da matriz concorrente, pois talvez isso tornasse o programa mais rápido.

- A principal dificuldade foi pensar em uma maneira de armazenar a soma dos múltiplos de forma que não houvesse condição de corrida. Primeiro pensei em fazer variáveis globais que as threads pudessem apenas somar mais um toda vez que encontrasse algum múltiplo de 2, 3, 5, 7 e 11, mas isso não deu certo. Depois pensei em adicionar os valores em matrizes, de forma que o tamanho dessa matriz de saida fosse (dimensão digitada pelo usuário) x 5, e assim cada coluna armazenaria os múltiplos de 2, 3, 5, 7 e 11 de cada linha da matriz de entrada, mas isso também não deu certo pela forma de percorrer a matriz. Assim, minha última escolha foi armazenar essas quantidades de múltiplos em um vetor, e deu certo.

Referências bibliográficas

- Material disponibilizado pela professora Silvana Rossetto para a turma de Computação Concorrente 2020.2 remoto.