Relatório da Tarefa 05

Comparação entre programação sequencial e paralela

20210093987 - Francisca Paula de Souza Braz
Data da entrega: 20/04/2025
Disciplina: DCA3703 - PROGRAMAÇÃO PARALELA - T01 (2025.1)

Introdução

A programação paralela é uma técnica utilizada para acelerar a execução de programas, aproveitando múltiplos núcleos de processadores modernos. Este relatório analisa a paralelização de um programa em C que conta a quantidade de números primos entre 2 e um valor máximo n.

O objetivo principal é comparar o desempenho entre a versão sequencial e a paralela do programa e discutir os desafios encontrados no processo.

Nessa atividade consigamos identificar com mais clareza quando duas ou mais threads/processos tentam modificar o mesmo dado ao mesmo tempo, isso pode causar erros imprevisíveis, chamados de condições de corrida e também como o equilíbrio de carga funciona, pois se uma thread estiver sobrecarregada enquanto outras estão ociosas, há desperdício de recursos.

Metodologia

Foram implementadas duas versões do mesmo algoritmo:

- Versão sequencial (sec.c)

- Versão paralela (par.c), utilizando #pragma omp parallel for com e sem a cláusula reduction.

Os testes foram realizados com diferentes valores de n: 1.000, 10.000 e 100.000.000. E os resultados foram avaliados com base na quantidade de números primos identificados e no tempo de execução, medido em milissegundos.

Para a medição precisa do tempo, foi utilizada a função gettimeofday(), da biblioteca <sys/time.h>, que fornece marcações com precisão de microssegundos. E fazendo uso do gettimeofday fizemos uma comparação detalhada entre as versões sequencial e paralela do algoritmo, executadas em um notebook com 16 GB de RAM e 12 núcleos de processamento.

Para avaliar o desempenho e a correção da paralelização, foram implementadas duas versões de um programa em C que conta quantos números primos existem entre 2 e um valor máximo n: uma versão sequencial, uma paralela.

A paralelização foi feita aplicando a diretiva #pragma omp parallel for sobre o laço principal que verifica se cada número é primo.

O que resultou em valores de tempos menores, mas, em contrapartida, os valores dos primos deram divergentes. Ao realizar o teste da versão paralelizada observou-se que o contador++ pode ser executado ao mesmo tempo, por várias threads e perder contagens. A solução encontrada foi usar uma cláusula de redução, para obter uma análise mais completa, o código sequencial foi executado individualmente e, em seguida, o código paralelo foi executado com e sem a cláusula de redução.

A versão correta utilizou a cláusula reduction(+:contador), como demonstrado abaixo:

```
#pragma omp parallel for reduction(+:contador)
for (int i = 2; i <= m; i++) {
    if (ehPrimo(i)) {
        contador++;
    }
}
```

Isso garante que cada thread conte separadamente e depois os valores sejam somados corretamente.

Resultados

Os resultados demonstram claramente os desafios e benefícios da programação paralela. A versão paralela sem a cláusula reduction apresentou resultados incorretos nas contagens de primos, indicando problemas de sincronização e condição de corrida no acesso à variável compartilhada. Com a inclusão da cláusula reduction, os resultados passaram a coincidir com os da versão sequencial, garantindo a correção do programa.

Quanto ao desempenho, observa-se que para valores pequenos de n, a versão sequencial ainda é mais rápida, devido à sobrecarga de criação e gerenciamento de threads. No entanto, à medida que o valor de n aumenta, a versão paralela com reduction apresenta ganhos expressivos de desempenho, chegando a ser quase cinco vezes mais rápida do que a versão sequencial para n = 100.000.000.

Versão	Valor de n	Qtd de primos	Tempo de execução (ms)
Sequencial	1.000	168	0.069
Paralela (sem `reduction`)	1.000	85	0.485
Paralela (com `reduction`)	1.000	168	0.411
Sequencial	10.000	1229	0.679
Paralela (sem `reduction`)	10.000	791	0.506
Paralela (com `reduction`)	10.000	1229	0.487
Sequencial	100.000.000	5.761.455	128.961,801
Paralela (sem `reduction`)	100.000.000	5.757.113	26.760,887
Paralela (com `reduction`)	100.000.000	5.761.455	26.424,433

Esses testes evidenciam dois dos principais desafios iniciais da programação paralela: garantir a correção dos resultados usando as cláusulas e alcançar um bom

balanceamento de carga, aproveitando efetivamente os recursos de hardware disponíveis.

Conclusão

A implementação paralela de algoritmos pode trazer ganhos significativos de desempenho, especialmente para tarefas computacionalmente intensivas, como a contagem de números primos em grandes intervalos. Contudo, os experimentos realizados evidenciam que a paralelização exige cuidados adicionais para garantir a correção dos resultados. O uso incorreto de variáveis compartilhadas, como observado nas versões sem reduction, pode comprometer a precisão da saída final, mesmo que o desempenho melhore.

Além disso, notou-se que para valores pequenos de entrada, a sobrecarga da paralelização pode tornar a execução mais lenta do que na versão sequencial. Isso reforça a importância de avaliar cuidadosamente o custo-benefício do uso de paralelismo em cada situação específica, considerando o tamanho da tarefa e a capacidade do hardware.

A reflexão mais importante deste trabalho é que programar em paralelo não significa apenas distribuir tarefas, mas também gerenciar sincronização, comunicação e equilíbrio de carga entre threads. Esses aspectos são essenciais para se obter não só eficiência, mas também correção e confiabilidade nos resultados. A aplicação de técnicas como reduction e a compreensão do comportamento do sistema tornam-se, assim, fundamentais para o sucesso na programação paralela.