Relatório da Tarefa 06

Escopo de variáveis e regiões críticas

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

Introdução

O objetivo desta atividade foi implementar, em linguagem C, a estimativa estocástica de π utilizando o método de Monte Carlo, além de explorar técnicas de paralelização com OpenMP. A proposta visou investigar o desempenho do programa ao empregar diferentes diretivas de paralelismo, focando na identificação e correção de condições de corrida, e analisando o impacto das cláusulas de escopo na obtenção de resultados corretos e consistentes.

A **estimativa estocástica** de π é baseada na ideia de gerar pontos aleatórios dentro de um quadrado e contar quantos desses pontos caem dentro de um círculo inscrito, o que, por meio de probabilidade, permite estimar o valor de π. À medida que o número de pontos gerados aumenta, as estimativas de π se tornam mais precisas, mas isso também resulta em maior tempo de processamento. As condições de corrida podem comprometer a precisão dos cálculos, pois múltiplas threads ou processos podem acessar e modificar os dados compartilhados simultaneamente, levando a resultados inconsistentes.

Para solucionar esse problema, foi essencial utilizar#pragma omp critical para evitar condições de corrida, garantindo que as threads acessem e modifiquem os dados compartilhados de maneira controlada e sem conflitos, assegurando a precisão dos resultados. Também, foram testadas diversas cláusulas para entender o comportamento e como isso pode ajudar a tornar o escopo mais claro em programas complexos.

Metodologia

A primeira versão do código foi paralelizada com #pragma omp parallel for, porém, apresentava resultados inconsistentes devido à condição de corrida causada pelo acesso simultâneo à variável **pontos_dentro_circulo** pelas múltiplas threads.

Para corrigir, foi utilizado #pragma omp critical, protegendo a atualização da variável pontos_dentro_circulo, garantindo que apenas uma thread a modificasse por vez.

Em seguida, a paralelização foi reestruturada com separando a região paralelizada com o paralelismo do for, essa forma permite controle mais granular e facilita o uso das cláusulas de escopo.

Foram testadas individualmente as seguintes cláusulas:

- **private;**
- **firstprivate;**
- **lastprivate;**
- **shared ;**
- **default(none);**

Resultados

Na implementação inicial, ao paralelizar o código utilizando `#pragma omp parallel for` sem qualquer controle de acesso à variável `pontos_dentro_circulo`, foram observados valores inconsistentes para a estimativa de π.

Estimativa de π: 0.471198

Isso aconteceu devido à presença de uma condição de corrida, em que múltiplas threads tentavam atualizar a variável `pontos_dentro_circulo` simultaneamente, gerando resultados imprevisíveis.

Para corrigir esse problema, foi aplicada a diretiva `#pragma omp critical`, que impede que mais de uma thread modifique a variável **pontos_dentro_circulo** ao mesmo tempo. Com isso, a condição de corrida foi eliminada e a estimativa de π se estabilizou, aproximando-se do valor esperado, como mostrado no seguinte resultado:

Estimativa de π: 3.141538

Além disso, diferentes cláusulas do OpenMP foram testadas para observar como cada uma afetaria o comportamento do programa.

A cláusula `private` foi usada para garantir que cada thread tivesse sua própria cópia de variáveis de loop, o que corrigiu problemas causados pela concorrência de acesso às variáveis compartilhadas.

π com private: 0.00000

E mesmo criando uma variável local para ser o contador e somando a local ao fim da com o critical, a versão global não acumula os valores somados dentro da região paralela, o resultado foi incorreto.

A cláusula `firstprivate` assegurou que cada thread iniciasse com um valor de variável distinto, no caso, o valor inicial de semente, o que ajudou a garantir que cada thread gerasse uma sequência independente de números aleatórios.

A cláusula `lastprivate`, que preserva o valor final de uma variável após o término do loop, não teve impacto significativo no resultado, pois o valor final da variável `pontos_dentro_circulo` não era utilizado após o loop. Por outro lado, a cláusula `shared` permitiu que variáveis como `pontos_dentro_circulo` fossem compartilhadas entre as threads, mas exigiu que o acesso a essas variáveis fosse protegido com `#pragma omp critical` para evitar que múltiplas threads as modificassem simultaneamente, o que poderia causar inconsistências.

Por fim, a cláusula `default(none)` foi aplicada para forçar uma declaração explícita de todas as variáveis como `private`, `shared` ou outras, o que aumentou a clareza do código e ajudou a evitar erros sutis. Esses resultados demonstram a importância de aplicar as diretivas corretamente ao trabalhar com OpenMP para garantir que o programa funcione adequadamente em um ambiente paralelo, evitando condições de corrida e melhorando a precisão dos cálculos.

Conclusão

A implementação da estimativa estocástica de π utilizando o método de Monte Carlo em paralelo demonstrou como as diretivas do OpenMP podem afetar diretamente o comportamento de um programa e a precisão dos resultados. Cada cláusula utilizada no OpenMP teve um impacto distinto na execução e no controle das variáveis.

A cláusula `private` foi essencial para garantir que cada thread tivesse sua própria cópia das variáveis de loop, evitando problemas de concorrência e interferência no valor das variáveis compartilhadas. Isso assegurou que cada thread fosse independente no cálculo.

Por outro lado, a cláusula `firstprivate` garantiu que cada thread iniciasse com um valor inicial de variável igual ao valor de `semente`, proporcionando aleatoriedade

distinta entre as threads e evitando que múltiplas threads gerassem os mesmos números aleatórios. Isso foi crucial para a variabilidade nos pontos gerados e, consequentemente, para uma melhor estimativa de π.

A cláusula `lastprivate` foi aplicada, mas não teve impacto relevante neste caso, pois o valor da variável não era necessário após o término do loop. No entanto, ela poderia ser útil em outros cenários, onde o valor final de uma variável precisa ser preservado após o término de uma iteração paralela.

A cláusula `shared` permitiu que a variável `count` fosse compartilhada entre todas as threads, mas foi necessário protegê-la com a diretiva `#pragma omp critical` para evitar condições de corrida, garantindo que apenas uma thread pudesse modificar a variável por vez. Isso é particularmente importante em programas onde múltiplas threads precisam acessar e modificar variáveis compartilhadas de forma segura.

Finalmente, a cláusula `default(none)` mostrou-se extremamente útil para tornar o escopo das variáveis mais claro e explícito. Ao forçar a declaração de todas as variáveis com um escopo definido (seja `private`, `shared`, etc.), ela ajudou a evitar ambiguidade, tornando o código mais seguro e fácil de entender, especialmente em programas mais complexos. Em sistemas grandes e com várias threads, a clareza na definição do escopo das variáveis pode evitar erros difíceis de detectar, como o uso incorreto de variáveis compartilhadas ou a manipulação de variáveis de loop concorrentes.