UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
IDCA3703 - PROGRAMAÇÃO PARALELA

TAREFA 6 - ESCOPO DE VARIÁVEIS E REGIÕES CRÍTICAS
RELATÓRIO DE EXECUÇÃO

ERNANE FERREIRA ROCHA JUNIOR

NATAL/RN, 17 DE ABRIL DE 2025

SUMÁRIO

1. INTRODUÇÃO

A constante π (pi) é amplamente conhecida na matemática por representar a razão entre a circunferência de um círculo e seu diâmetro. Embora seu valor exato seja irracional e infinito, é possível obter aproximações numéricas bastante precisas por meio de métodos computacionais. Um desses métodos é a técnica de Monte Carlo, que estima o valor de π por meio de simulações estocásticas baseadas em probabilidade e geometria. Trata-se de uma abordagem que se beneficia significativamente da paralelização, pois realiza um grande número de iterações independentes, o que a torna ideal para execução concorrente em múltiplos núcleos. Neste trabalho, implementamos em linguagem *C* a estimativa de π utilizando esse método e exploramos sua paralelização com a biblioteca *OpenMP*. Inicialmente, mostramos uma versão paralela simples com *#pragma omp parallel for*, que, embora promissora, gera resultados incorretos devido a uma condição de corrida na variável compartilhada de contagem. Em seguida, corrigimos esse problema utilizando a diretiva *#pragma omp critical* e refinamos a estrutura do programa aplicando diferentes cláusulas como *private*, *firstprivate*, *lastprivate* e *shared*. Por fim, discutimos como a cláusula *default(none)* contribui para tornar o escopo das variáveis mais explícito e seguro em contextos de paralelismo. Ao longo do relatório, analisamos o impacto dessas estratégias tanto na correção quanto no desempenho da estimativa.

2. METODOLOGIA

A implementação da estimativa estocástica de π foi realizada em linguagem *C*, utilizando o compilador *gcc-14* com suporte à biblioteca *OpenMP* para exploração de paralelismo por meio de diretivas. O código segue o método de Monte Carlo, em que pares de coordenadas aleatórias são gerados dentro de um quadrado de lado unitário, e verifica-se se esses pontos caem dentro de um círculo inscrito. A razão entre a quantidade de pontos dentro do círculo e o total de amostras é usada para estimar o valor de π. O programa foi compilado com a flag *-fopenmp* `e executado em um MacBook Air com processador Apple M2, que possui 8 núcleos (4 de desempenho e 4 de eficiência), e 8 GB de memória RAM.

Como não foi especificado manualmente o número de threads, o *OpenMP* utiliza automaticamente todos os núcleos disponíveis, distribuindo as iterações das regiões paralelas entre as threads de forma implícita. O processo de compilação e execução foi realizado da seguinte forma

```
gcc-14 -fopenmp ./task-6.variable-scope-and-critical-
regions/main.c -o ./task-6.variable-scope-and-critical-
regions/out/main.o && ./task-6.variable-scope-and-critical-
regions/out/main.o
```

Foram desenvolvidas e testadas sete versões do programa: uma versão sequencial de referência, uma versão paralela com condição de corrida causada pelo acesso concorrente à variável de contagem, uma versão corrigida com uso de região crítica, e outras versões explorando diferentes cláusulas de escopo de variáveis (*private, firstprivate*, *lastprivate*, *default(none)*), todas utilizando as diretivas *#pragma omp parallel* combinadas com *#pragma omp for*. O tempo de execução, em adição ao que foi solicitado, foi medido com a função *clock()* da biblioteca <*time.h*>, permitindo uma análise comparativa de desempenho entre as abordagens.

3. RESULTADOS

A execução das diferentes versões do programa forneceu uma base sólida para comparar a precisão das estimativas e o impacto das estratégias de paralelização no desempenho. A versão sequencial serviu como referência, entregando uma estimativa de π próxima ao valor real (≈ 3.1416) em cerca de 0.193 segundos. Em contraste, a primeira tentativa de paralelização com *#pragma omp parallel for* resultou em um valor completamente incorreto (≈ 0.2102), evidenciando uma condição de corrida provocada pelo acesso simultâneo de múltiplas threads à variável compartilhada de contagem.

A correção inicial com *#pragma omp critical* eliminou o erro lógico e restaurou a precisão da estimativa, mas ao custo de um tempo de execução significativamente maior (≈ 3.758 *segundos*), refletindo o impacto da serialização imposta pela região crítica. Em seguida, a aplicação da cláusula *private* para criar uma variável de contagem local em cada thread permitiu a obtenção de um resultado correto com uma melhora expressiva no tempo (≈ 0.320 *segundos*), embora ainda inferior à versão sequencial em termos de precisão absoluta.

O uso de *firstprivate* aprimorou a geração de números aleatórios ao garantir *seeds* diferentes entre as *threads*, resultando em um valor de π mais preciso (≈ 3.1418) e em um tempo reduzido (≈ 0.227 *segundos*). A cláusula *lastprivate* demonstrou sua função ao preservar o valor da última iteração da variável de controle do laço. Finalmente, a versão com *default(none)* exigiu a declaração explícita dos escopos de todas as variáveis compartilhadas e privadas, promovendo maior clareza e segurança no código, e produziu um resultado preciso com tempo de execução comparável ao da versão com *firstprivate*.

```
✅ Case 1 - Sequential:      π ≈ 3.141877914187791 | Time: 0.193s
❌ Case 2 - Parallel for:    π ≈ 0.210229621022962 | Time: 0.372s (race condition)
✅ Case 3 - Critical:        π ≈ 3.142048314204831 | Time: 3.758s (idleness)
✅ Case 4 - Private:         π ≈ 3.129574312957431 | Time: 0.320s
✅ Case 5 - Firstprivate:    π ≈ 3.141803514180352 | Time: 0.227s
✅ Case 6 - Lastprivate:     π ≈ 3.130124713012471 | Time: 0.330s | Last i: 9999998
✅ Case 7 - Default(none):   π ≈ 3.141803114180311 | Time: 0.225s
```

Os tempos de execução e valores estimados foram consistentemente influenciados pela maneira como o acesso às variáveis compartilhadas foi tratado e pela forma como os dados foram distribuídos entre as *threads*. A ausência de controle adequado resultou em erros graves, enquanto o uso correto das cláusulas do *OpenMP* não apenas corrigiu o problema, mas também proporcionou um ganho considerável de desempenho em relação à abordagem sequencial.

4. CONCLUSÃO

A implementação da estimativa estocástica de π serviu como um exemplo eficaz para explorar os conceitos fundamentais de paralelização com *OpenMP*, destacando tanto os benefícios quanto os desafios dessa abordagem. A tentativa inicial de paralelizar o laço com *#pragma omp parallel for* evidenciou rapidamente a importância de gerenciar corretamente o acesso a variáveis compartilhadas, já que a ausência desse controle levou a uma condição de corrida e, consequentemente, a resultados incorretos. A partir dessa falha, foram testadas diferentes soluções com uso de regiões críticas e variáveis privadas, permitindo recuperar a precisão da estimativa e, em alguns casos, melhorar significativamente o desempenho.

Cada cláusula analisada — *private*, *firstprivate*, *lastprivate*, *shared* e *default(none)* — demonstrou seu papel específico na definição do escopo das variáveis e na coordenação entre as *threads*. Enquanto *private* e *firstprivate* foram fundamentais para evitar conflitos e garantir independência nas execuções paralelas, *lastprivate* mostrou utilidade em situações onde o valor da última iteração é relevante. A cláusula *default(none)*, por sua vez, reforçou boas práticas de programação ao exigir declarações explícitas, contribuindo para a legibilidade, manutenção e segurança de códigos mais complexos.

No geral, o exercício evidenciou que a paralelização bem-sucedida depende não apenas do uso de diretivas como *#pragma omp parallel*, mas também do domínio preciso sobre as cláusulas de escopo. Além disso, mostrou-se essencial realizar testes cuidadosos e interpretar criticamente os resultados para garantir que o ganho de desempenho não comprometa a correção lógica do programa.

5. ANEXOS

- Repositório no Github com o programa desenvolvido: https://github.com/ErnaneJ/parallel-programming-dca3703