Universidade Federal do Rio Grande do Norte - UFRN
Departamento de Engenharia da Computação e Automação - DCA
DCA3703 - Programação Paralela

Tarefa 06: Escopo de variáveis e regiões críticas

Discente: Quelita Míriam
Docente: Samuel Xavier de Souza

I. INTRODUÇÃO

A programação paralela é uma ferramenta poderosa para otimizar o desempenho de algoritmos que demandam alto custo computacional. Neste relatório, é investigado o comportamento de variáveis em regiões paralelas utilizando OpenMP, através da implementação da estimativa estocástica de π (pi). O foco principal é compreender o impacto do escopo de variáveis (*private, firstprivate, lastprivate, shared*) e das diretivas *critical* e *default(none)* em programas paralelos.

A tarefa propõe a versão sequencial e paralelizada do cálculo de π e a análise de erros provocados por condições de corrida, seguidos da correção utilizando estratégias adequadas. O objetivo é observar como diferentes cláusulas afetam o funcionamento e o resultado do programa, além de reforçar boas práticas na definição de escopos em programação paralela.

II. TEORIA

A estimativa estocástica de π utiliza o método de Monte Carlo. Gera-se aleatoriamente pontos no plano (x, y) dentro do quadrado [0,1] x [0,1], como mostrada na imagem 1. A razão entre os pontos que caem dentro do círculo de raio 1 e os pontos totais gera uma aproximação de π, pois:

$$\pi \approx 4 * \frac{pontos\ dentro\ do\ círculo}{total\ de\ pontos}$$

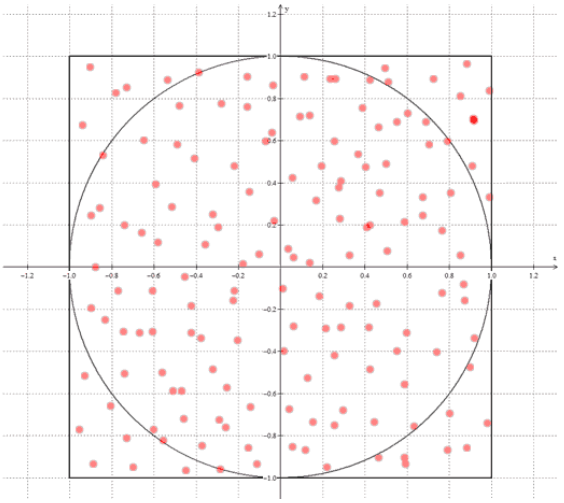

Imagem 1: Ideia geométrica simples para estimar o valor de π com o método de integração de Monte Carlo

Abaixo, um pseudocódigo da implementação do cálculo feito.

```
01. monteCarloPi(n)
02. |   acertos ← 0
03. |   para i ← 0 até n
04. |   |   x ← sorteie um número real entre 0 e 1
05. |   |   y ← sorteie um número real entre 0 e 1
06. |   |   se(x * x + y * y < 1)
07. |   |   |   acertos ← acertos + 1
08. |   |   fim_se
09. |   fim_para
10. |   retorne 4 * acertos / n
11. fim_monteCarloPi
```

Imagem 2: Pseudocódigo da estimativa estocástica de π

III. METODOLOGIA

Compilando o código[1] a partir de "*gcc main.c -o main -fopenmp -lm && ./main*" testam-se diferentes versões para a tarefa:

- Versão sequencial: sem paralelismo, usada como referência.
- Versão paralela com erro: paraleliza o loop, mas sofre de condição de corrida por atualizar uma variável compartilhada (*count*) sem proteção.
- Versões corrigidas: utilizando estratégias como *critical, private, firstprivate, lastprivate e default(none)* para corrigir erros e explorar o comportamento das variáveis.

IV. RESULTADOS E DISCUSSÕES

Perante os resultados abaixo do código compilado, a versão com erro apresenta resultado incorreto devido a uma condição de corrida, onde múltiplas threads acessam e modificam a variável compartilhada *count* ao mesmo tempo. O resultado para esse erro depende tanto do compilador quanto da máquina utilizada, pois diferentes arquiteturas de hardware, modelos de memória e estratégias de escalonamento de threads podem influenciar como as operações concorrentes ocorrem. Assim, em algumas execuções ou máquinas, o erro pode ser mais ou menos evidente, como neste caso que parece ser quase correto. As demais versões corrigem isso, cada uma aplicando uma estratégia de controle de escopo e sincronização.

[1] Link do código desenvolvido para a tarefa proposta:
https://github.com/quelita2/programacao-paralela/blob/main/topico01/tarefa06/src/main.c

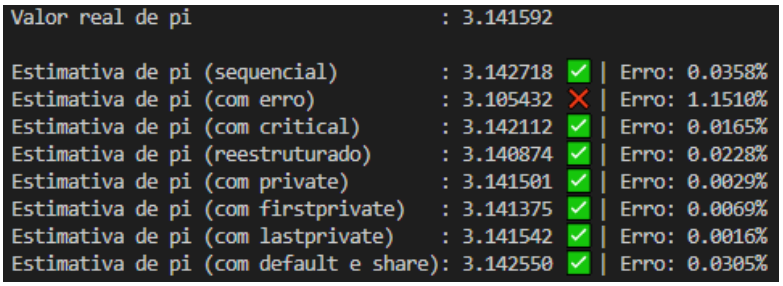

Imagem 1: Resultados de compilação da atividade

Cláusulas OpenMP utilizadas:

- ***Private(var)***: Cria uma cópia privada da variável para cada thread. É utilizada quando a variável não deve ser compartilhada. Evita conflitos ao usar contadores locais.
- ***FirstPrivate(var)***: Cópia privada, mas com o valor inicializado com o valor original. É útil para seeds ou variáveis que precisam de um valor inicial. Usado para gerar números aleatórios independentes com base no tempo.
- ***LastPrivate(var)***: Cópia privada, mas ao final da execução, o valor da última iteração é atribuído à variável original. É utilizada quando se quer manter o valor final da última iteração do loop.
- ***Shared(var)***: A variável é visível e compartilhada entre todas as threads. Deve ser usada com cuidado, pois necessita de sincronização (*critical*) para evitar condições de corrida.
- ***Default(none)***: Obrigatoriedade de declarar o escopo de todas as variáveis. Seu uso aumenta a segurança e a clareza do código. Muito útil em programas grandes e complexos.
- ***Critical***: Garante que apenas uma thread por vez execute o bloco de código. É necessária ao atualizar variáveis shared como o count.

V. CONCLUSÃO

A tarefa proposta permitiu compreender, de forma prática, os impactos do escopo de variáveis em regiões paralelas usando OpenMP. A estimativa estocástica de π, além de ser uma aplicação simples, foi eficaz para observar erros clássicos como condições de corrida e testar soluções com as cláusulas de escopo.

A correta definição do escopo (*private, firstprivate, lastprivate, shared*) é essencial para a correção e desempenho do programa. Além disso, o uso de *default(none)* se mostrou uma prática recomendada para garantir maior controle sobre as variáveis em programas paralelos mais complexos.

O experimento mostrou que, com o uso adequado das cláusulas, é possível obter resultados próximos ao valor real de π com desempenho otimizado e sem erros de concorrência.

VI. REFERÊNCIAS

CYBERINI, Victor. *Calculando o valor de pi via método de Monte Carlo*. Blog Cyberini, 2018. Disponível em: <https://www.blogcyberini.com/2018/09/calculando-o-valor-de-pi-via-metodo-de-monte-carlo.html>.