UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ENGENHARIA DE SOFTWARE

FELIPE YUJI SHIRAE, R.A.:1997408

GABRIEL HENRIQUE DE ALMEIDA SOUZA, R.A.:2504162

GUSTAVO PREZOTO BOCA, R.A.: 2250373

JOÃO PEDRO SANTOS DE ARAÚJO, R.A.: 2525852

VICTOR MATHEUS TAVARES RAFAEL, R.A.: 2525917

Cálculo de Pi por Método Monte Carlo

CORNÉLIO PROCÓPIO

2024

Relatório do Trabalho Prático - Disciplina de Computação Paralela

1. Introdução

Este relatório apresenta o desenvolvimento e a análise de três soluções (sequencial, paralela e distribuída) para o cálculo do valor de Pi utilizando o Método de Monte Carlo. O objetivo principal é comparar os tempos de execução dessas soluções e avaliar sua escalabilidade e eficiência.

O Método de Monte Carlo utiliza técnicas de amostragem aleatória para resolver problemas numéricos. Neste caso, pontos são gerados aleatoriamente dentro de um quadrado unitário, e a proporção de pontos que caem dentro de um círculo unitário permite estimar o valor de Pi.

2. Metodologia

2.1. Descrição do Problema

Método de Monte Carlo é um termo utilizado para se referir a qualquer método que resolve um problema gerando números aleatórios e observando se uma dada fração desses números satisfaz uma propriedade previamente estabelecida.

Para calcular a área da circunferência unitária, utilizaremos o método de integração de Monte Carlo. A ideia é colocar a circunferência dentro de uma figura, cuja área seja fácil de calcular, e sortear pontos aleatórios dentro da figura. Utilizaremos um quadrado.

Se o ponto sorteado estiver dentro da circunferência, então marcamos um acerto. Ao final dos sorteios, espera-se que a área da circunferência seja proporcional à taxa de acertos e à área do quadrado. Esse valor será uma aproximação para π.

"Espera-se" porque o método é estocástico, isto é, os resultados obtidos são aleatórios. Porém, para uma grande amostra de pontos, ele deve convergir para o resultado esperado, ou seja, π.

A taxa de acertos é a razão entre o número de acertos e o total de sorteios realizados. Este valor pode ser visto como uma aproximação da probabilidade de sortear um ponto e ele estar dentro da circunferência.

A probabilidade exata é igual à razão entre a área da circunferência e a área do quadrado:

$$P = \frac{A_{\text{circ}}}{A_{\text{quad}}}$$

Onde:

- P: probabilidade do ponto estar dentro da circunferência.
- A_{circ}: área da circunferência.
- A_{quad}: área do quadrado.

Na circunferência unitária, temos Acirc=π, o que implica em:

$$P = \frac{\pi}{A_{\text{quad}}}$$
$$\pi = P.A_{\text{quad}}$$

Suponha que sorteamos N pontos e que $N_{acertos}$ é o número de acertos, onde $N_{acertos} \leq N$. Então, P será aproximadamente igual a $N_{acertos}/N$, portanto o valor aproximado de π é dado por:

$$\pi \approx \frac{N_{\text{acertos}}}{N}.A_{\text{quad}}$$

Quanto maior o número de sorteios, melhor será a aproximação.

Verificação se o Ponto está dentro da Circunferência

Por simplicidade, vamos utilizar a circunferência unitária com centro na origem, definida pela equação:

$$x^2 + y^2 = 1$$

Dado um ponto (x,y), se o ponto está dentro da circunferência, então ele deve satisfazer a seguinte condição:

$$x^2 + y^2 < 1$$

2.2. Soluções Implementadas

- Solução Sequencial:
  - Na solução sequencial, o cálculo de π é feito em uma única thread, que gera todos os pontos aleatórios e verifica se eles estão dentro do círculo.
  - Características:
    - Código fácil de entender e implementar.
    - Não muito eficiente, pois executa apenas em uma thread, aproveitando apenas um núcleo do processador.
    - O tempo de execução cresce linearmente com o número de pontos.

- Solução Paralela:
  - Na solução paralela, o trabalho é dividido entre múltiplas threads (total de threads do seu processador recuperados dinamicamente), que calculam partes do resultado simultaneamente. Cada thread gera um subconjunto dos pontos e conta os que estão dentro do círculo.
  - Características:
    - Aproveita múltiplos núcleos do processador, reduzindo drasticamente o tempo de execução do algoritmo.
    - Usamos ExecutorService e Future para controle e gerenciamento das threads.

- Solução Distribuída:
  - Na solução distribuída, o cálculo é dividido entre vários clientes (escravos) que executam partes da tarefa. Um servidor (mestre) coordena o processo, distribui o trabalho e calcula o resultado final.
  - Características:
    - Pode processar grandes volumes de dados distribuindo o trabalho entre vários clientes.
    - Comunicação feita via sockets usando ServerSocket e Socket.

- Possui uma complexidade maior que nas soluções sequenciais e paralelas, devido à necessidade de comunicação em rede.
- A principal desvantagem é que existe um pequeno gargalo durante a comunicação entre os servidores/clientes.
- Um ponto de melhoria do algoritmo distribuído poderia ser implementar paralelismo no código do cliente.

2.3. Configuração do Ambiente

- Hardware:
  - Processador: Ryzen 7 5700X.
  - Número de núcleos: 8 (16 threads).
  - Memória RAM: 16 GB.
- Software:
  - Sistema operacional: Windows 11.
  - Linguagem de programação: Java 18.

3. Resultados

3.1. Tempos de Execução

Os tempos foram medidos para diferentes tamanhos de N. A tabela abaixo apresenta os resultados (em segundos):

Tamanho N	Sequencial	Paralela	Distribuída
100.000.000	1263ms	155ms	365ms
500.000.00	6280ms	576ms	1678ms
1.000.000.000	12503ms	1220ms	3330ms
5.000.000.000	62517ms	5549ms	17542ms

3.2. Análise de Escalabilidade

A análise de escalabilidade mostra que a solução sequencial, por utilizar apenas um núcleo do processador, cresce linearmente em tempo com o aumento do problema, tornando-se ineficiente para grandes volumes. A solução paralela, ao explorar múltiplas threads, reduz drasticamente o tempo de execução, sendo a mais eficiente no ambiente testado, com ganhos significativos ao aproveitar os 16 threads do processador. Já a solução distribuída, apesar de introduzir overhead de

comunicação, se destaca em ambientes com múltiplas máquinas, permitindo escalabilidade horizontal e processamento de volumes massivos que excedem os limites de um único sistema.

Sugestão de Melhoria:

1. Distribuido: implementar paralelismo no código dos clientes;

4. Desafios e Soluções

- Desafio: Geração eficiente de números aleatórios em paralelo.
    - Solução: Utilização de geradores independentes para cada thread.
- Desafio: Sincronização entre os resultados das threads.
    - Solução: Uso de coleções thread-safe para agregar resultados.
- Desafio: Configuração da comunicação entre hosts.
    - Solução: Implementação de sockets.

5. Conclusão

O trabalho demonstrou a viabilidade e os benefícios das soluções paralela e distribuída em relação à sequencial, especialmente em volumes elevados de dados. A solução paralela apresentou os melhores resultados em termos de eficiência, enquanto a solução distribuída foi limitada por gargalos de comunicação.

A experiência permitiu compreender as vantagens e limitações de cada abordagem, destacando a importância de otimizações para soluções distribuídas em ambientes reais.

6. Link para o Repositório

Link para o GitHub