**SISTEMAS DE INFORMAÇÃO**

*Jéferson Alves Santos*

**Objetivo**  
  
Executar, analisar e modificar um programa MPI que estima o valor de π por meio do método de Monte Carlo (MMC) distribuído entre múltiplos processos.  
  
**Parte 1 — Execução básica**

1.1. Saída esperada

Execute o programa com 4 processos. Copie aqui a saída do terminal:

Valor estimado de Pi: 3.141130

Valor estimado de Pi: 3.141130

Valor estimado de Pi: 3.141130

Valor estimado de Pi: 3.141130

**Parte 2 — Análise de funcionamento**

2.1. O que cada processo faz neste código?

Cada processo é encarregado de produzir uma quantidade igual do total de pontos aleatórios (dividido pelo número de processos). Ele gera coordenadas x e y aleatórias entre 0 e 1, verifica se o ponto está dentro do quarto de círculo (se x² + y² ≤ 1) e contabiliza quantos pontos atendem a essa condição localmente. Posteriormente, todos os processos enviam suas contagens locais ao processo raiz (rank 0) por meio do MPI\_Reduce, que agrega os valores para determinar a estimativa global de π.

2.2. Por que o valor estimado de π varia a cada execução? Resposta:

 O valor oscila porque o método utiliza a geração de números aleatórios para simular os pontos. Cada execução gera uma sequência única de pontos aleatórios, o que provoca variações nas contagens de pontos dentro do círculo e, consequentemente, estimativas diferentes de π. A variação é menor (a aproximação é melhor) quanto maior o número de pontos.

2.3. Qual o papel da função MPI Reduce nesse código?

A função MPI\_Reduce é empregada para somar as contagens locais de pontos dentro do círculo de todos os processos, resultando em um valor global que é reunido no processo raiz (rank 0). Isso possibilita a estimativa de π a partir da relação entre o número total de pontos dentro do círculo e o total de pontos gerados, com o trabalho sendo realizado de forma paralela.

**Parte 3 — Modificação**

3.1. Modifique o código para que cada processo imprima a quantidade de pontos gerados e quantos caíram dentro do círculo.

O processo 0 ainda deve calcular e exibir o valor estimado de π com base na soma global dos acertos.

Faça um commit com sua modificação e anexe abaixo o código completo.

#include <mpi.h>  
#include <stdio.h>  
#include <stdlib.h>

int main(int argc, char\*\* argv) {

int num\_points = 10000000; // Total de pontos (ajuste se necessário para maior precisão)

int rank, size;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

srand(time(NULL) \* rank); // Semente diferente para cada processo para evitar repetições

int local\_points = num\_points / size;

int local\_inside = 0;

for (int i = 0; i < local\_points; i++) {

double x = (double) rand() / RAND\_MAX;

double y = (double) rand() / RAND\_MAX;

if (x \* x + y \* y <= 1.0) {

local\_inside++;

}

}

// Modificação: Cada processo imprime seus valores locais

printf("Processo %d gerou %d pontos, %d dentro do círculo\n", rank, local\_points, local\_inside);

int global\_inside;

MPI\_Reduce(&local\_inside, &global\_inside, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

double pi = 4.0 \* global\_inside / num\_points;

printf("Valor estimado de Pi: %f\n", pi);

}

MPI\_Finalize();

return 0;

}  
  
3.2. Copie aqui a saída do seu programa modificado:

Processo 0 gerou 10000000 pontos, 7852825 dentro do círculo  
Valor estimado de Pi: 3.141130

Processo 0 gerou 10000000 pontos, 7852825 dentro do círculo  
Valor estimado de Pi: 3.141130

Processo 0 gerou 10000000 pontos, 7852825 dentro do círculo  
Valor estimado de Pi: 3.141130

Processo 0 gerou 10000000 pontos, 7852825 dentro do círculo  
Valor estimado de Pi: 3.141130

**Parte 4 — Análise com utilitários Linux**

4.1. Use o comando time para medir o tempo de execução do programa com 2, 4 e 8 processos.

Processos | Tempo (real)

2 | 0m11.865s

4 | 0m0.421s

8 | 0m0.603s

4.2. O uso de CPU foi distribuído de forma equilibrada entre os Processos?

Sim, a utilização da CPU é balanceada, pois o trabalho é repartido de maneira justa: cada processo produz a mesma quantidade de pontos (num\_points / size) e executa cálculos equivalentes.