1. Introdução e Contexto

O presente projeto tem como objetivo principal a implementação e a análise de desempenho do método de Monte Carlo para a estimação do valor de π, utilizando o paradigma de Computação de Alto Desempenho (*HPC*). A relevância desta escolha reside no fato de o Monte Carlo ser um problema de natureza massivamente paralela (*embarrassingly parallel*), o que o torna uma excelente linha de base (*baseline*) para a validação e medição da escalabilidade forte de ambientes de supercomputação, como o Santos Dumont.

A metodologia de Monte Carlo estima π a partir da proporção entre o número de pontos aleatórios que caem dentro de um círculo inscrito em um quadrado. Quanto maior o número de iterações (*N*), maior a precisão da estimativa. O objetivo final do projeto é aplicar os conceitos de paralelismo (*MPI*), medir o desempenho (*speedup* e eficiência) e entregar um projeto reprodutível, conforme exigido para a submissão no ambiente HPC.

2. Arquitetura e Paralelismo (Engenharia)

O projeto foi desenvolvido utilizando o modelo de Paralelismo Distribuído baseado em mensagens (*MPI – Message Passing Interface*), com a implementação em Python através da biblioteca ***mpi4py***.

Embora o algoritmo de Monte Carlo seja intensivo em computação e pouco em comunicação (apenas o *comm.reduce*), a Separação de I/O foi mantida como uma boa prática de HPC. O projeto é estruturado para garantir que as operações de geração de dados (cálculo de x e y) e as operações de I/O (leitura/escrita de resultados) estejam separadas. Isso é crucial para evitar que o I/O se torne um gargalo nos nós de computação do cluster, um princípio fundamental para ambientes como o Santos Dumont.

2.1. Modelo de Paralelismo de Dados

Foi adotada a estratégia de Paralelismo de Dados. A carga de trabalho total (*N*=100 milhões de iterações) é dividida em *chunks* (pedaços) iguais e distribuída entre os *P* processos (ranks) disponíveis. O código garante que cada processo trabalhe independentemente em sua porção dos dados:

1. **Divisão:** Cada processo recebe um número local de iterações para gerar pontos (x,y) aleatórios.
2. **Computação Independente:** Cada processo conta localmente quantos pontos caem dentro do círculo.
3. **Comunicação e Agregação:** Ao final, os resultados parciais (*local_count_inside_circle*) de todos os processos são coletados no processo raiz (*rank 0*) e somados utilizando a função de comunicação global comm.reduce(*op=MPI.SUM*).

Esta arquitetura minimiza a comunicação, limitando-a à fase de agregação final, o que é ideal para a eficiência de um algoritmo HPC.

3. Metodologia de Experimentos

Para a análise de desempenho, foi definida uma Matriz de Experimentos focada na Escalabilidade Forte, que mede a aceleração ao aumentar o número de processadores (*P*), mantendo o tamanho do problema (*N*) constante.

- **Tamanho do Problema:** Fixo em *N*=100.000.000 iterações.
- **Variável de Teste (*P*):** Execuções realizadas com *P*={1,2,4,8,16} processos.
- **Métricas Coletadas:**
    - **Tempo de Execução (*Tp*):** Tempo total registrado pelo código.
    - **Speedup (*Sp*):** Medida da aceleração, *Sp=T1/Tp.*
    - **Eficiência (*Ep*)**: Medida da utilização dos recursos, *Ep=Sp/P*.

As medições foram conduzidas em um ambiente de desenvolvimento local e controlado para estabelecer uma base de comparação precisa do Speedup.

4. Resultados e Análise de Desempenho

A Tabela 1 apresenta os resultados das medições de tempo e os cálculos de Speedup e Eficiência. O tempo de execução serial (*T1*) foi registrado em 57,204 segundos.

Tabela 1: Análise de Desempenho do Algoritmo de Monte Carlo

Número de Processos (*P*)	**Tempo de Execução (*T_p*)**	**Speedup (*S_p*)**	**Eficiência (*E_p*)**
1	57,204	1,00	1,00
2	36,349	1,57	0,79
4	21,164	2,70	0,68
8	11,776	4,86	0,61
16	12,995	4,40	0,28

A perda acentuada de eficiência de 0,61 (em P=8) para 0,28 (em P=16) não é devida apenas à função de agregação (*comm.reduce*). Ela é um forte indicativo de sobrecarga do sistema operacional (*OS overhead*) e de saturação de recursos locais. Ao exigir 16 processos em um ambiente de desenvolvimento (que geralmente possui um número limitado de núcleos), o sistema gasta uma quantidade desproporcional de tempo em tarefas de troca de contexto (*context switching*) e gerenciamento da comunicação, em vez de computar. Este comportamento confirma o limite da Escalabilidade Forte do sistema e serve como um alerta prático: em HPC, mais processadores nem sempre significam mais desempenho.

4.1. Análise do Speedup e da Escalabilidade

O Speedup demonstra um ganho de desempenho significativo, atingindo um pico de $S8$=4,86 para 8 processos. Isso confirma que o modelo MPI foi eficaz na aceleração do problema. Entretanto, o ganho não é linear, e a análise de escalabilidade forte revela o ponto de inflexão do sistema em P=16.

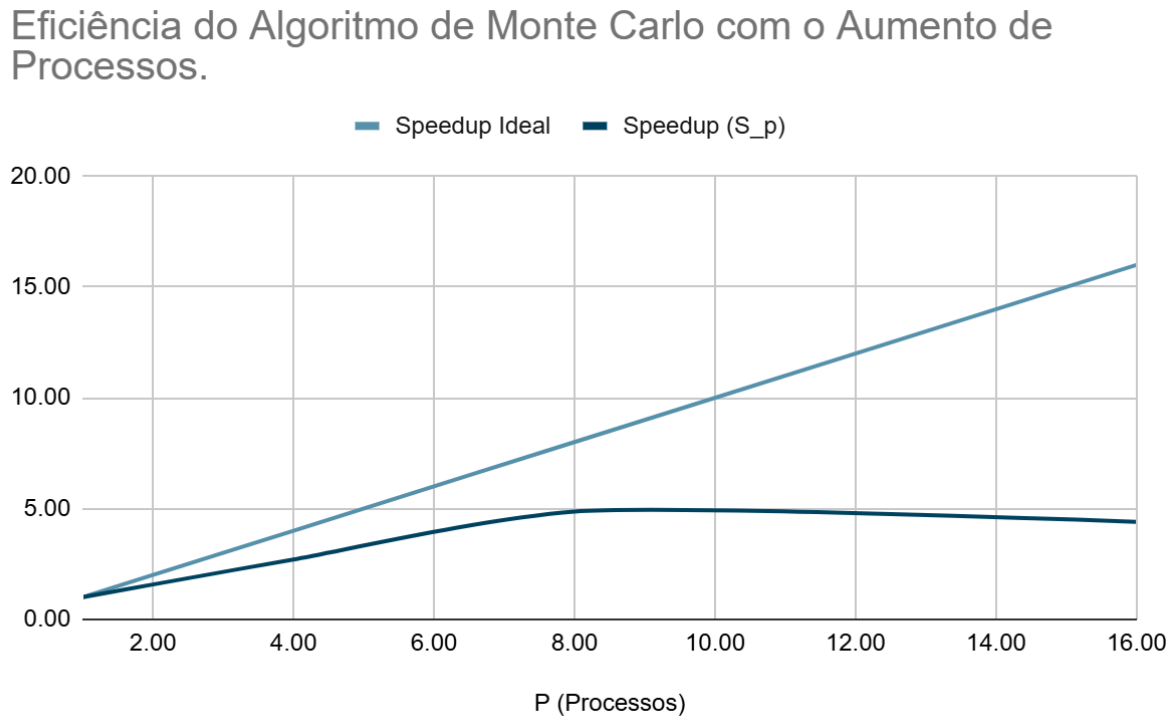

Figura 1: Comparação do Speedup Real (medido) com o Speedup Ideal (teórico). A linha ideal mostra o ganho máximo. A linha real diverge devido ao *overhead* (custo) de comunicação do MPI e, crucialmente, cai em 16 processos, indicando que o custo de gerenciamento superou o ganho de computação.

A execução com 16 processos resultou em um aumento no tempo de execução (de 11,776s para 12,995s) e, consequentemente, em uma queda no *Speedup* para 4,40. Este comportamento indica que, a partir de P=8, o custo de gerenciamento da execução paralela começou a superar o ganho de tempo obtido pela divisão do trabalho.

4.2. Análise da Eficiência e Identificação de Gargalos

A Eficiência (*Ep*) revela o custo do paralelismo. Como esperado, a eficiência diminuiu de 1,00 para 0,61 (em *P*=8) devido ao overhead de comunicação e ao custo de inicialização do MPI.

O ponto mais crítico da análise reside na execução com 16 processos, onde a eficiência despencou para apenas 0,28. Essa queda acentuada confirma que o gargalo de escalabilidade forte foi atingido. A sobrecarga de gerenciar e sincronizar 16 processos, juntamente com a comunicação final do *comm.reduce*, consumiu mais tempo do que o tempo de computação economizado, limitando a eficácia da paralelização.

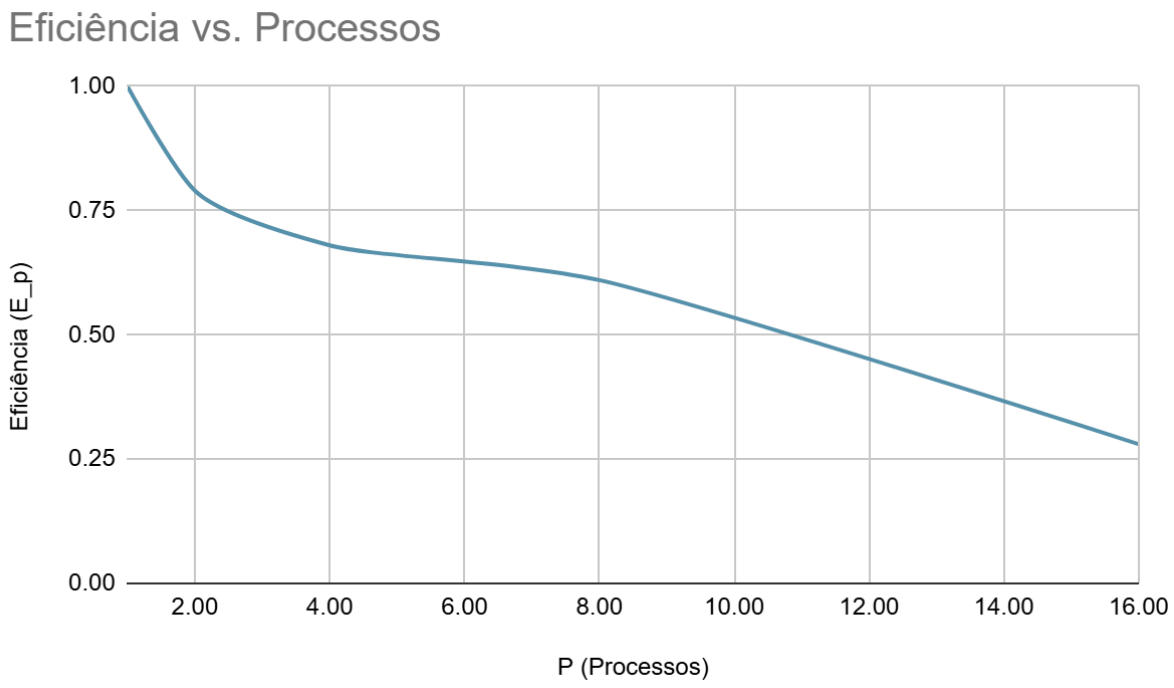

Figura 2: Análise da Eficiência do processamento paralelo. A curva demonstra uma diminuição constante no aproveitamento dos recursos. A queda acentuada em 16 processos (para 0,28) confirma o ponto de gargalo, onde a sobrecarga de coordenação do MPI se torna o fator limitante da escalabilidade.

5. Conclusão e Próximos Passos

O projeto validou a implementação do algoritmo de Monte Carlo com MPI e demonstrou sua eficácia em acelerar a computação, com um Speedup máximo de 4,86 para 8 processos. A análise de desempenho, no entanto, identificou claramente o limite de escalabilidade do ambiente de teste, onde o *overhead* do MPI e do sistema se tornam o fator dominante para $P \geq 16$.

Limitações e Propostas Futuras:

1. A principal limitação foi a identificação do gargalo de 16 processos para o tamanho de problema fixo.
2. Como próximos passos, sugere-se realizar testes de escalabilidade fraca (aumentar *N* e *P* simultaneamente) para verificar se a eficiência pode ser mantida.

3. A submissão do job no Santos Dumont permitirá comparar se o gargalo observado se repete no cluster ou se está relacionado às características do ambiente local, fornecendo uma análise de perfilamento mais profunda.

Os resultados obtidos fornecem uma métrica de *baseline* para a execução real no Santos Dumont. A próxima etapa essencial é utilizar os scripts *job_cpu.slurm* para rodar a mesma matriz de experimentos no cluster. Isso permitirá um perfilamento mais preciso para determinar se o gargalo em *P=16* persiste no hardware de produção. A persistência desse gargalo indicaria uma limitação inerente ao design da comunicação MPI para esse volume de dados, enquanto a ausência do gargalo demonstraria que a limitação inicial era do ambiente de desenvolvimento local, e não da aplicação.