Relatório 14- Bruna Meinberg

1. Pi sequencial

Valor estimado de Pi: 3.14152
Tempo de execução: 0.00302902 segundos

Utilizei o `rand` para geração de números aleatórios, contudo, sabe-se que eles são pseudo-aleatórios devido a sua semente. Por isso, não é possível considerar que são completamente aleatórios, mas mesmo assim, houve um bom desempenho.

2. Pi paralelo - versão 1

Valor estimado de Pi (paralelizado): 3.13936
Tempo de execução (paralelizado): 0.0161417 segundos

Utilizar o `rand` dentro de um ambiente pode ser problemático, porque a geração de sementes pode tornar-se conflituosa, afinal sua ordem não é completamente aleatória e como as threads não “globalizam” essas sementes, isso pode dar problemas.

3. Pi paralelo - versão 2

Valor estimado de Pi (com melhoria de paralelização): 3.1472
Tempo de execução (com melhoria de paralelização): 0.00498059 segundos

A segunda versão de paralelização melhorou o valor do pi, e tomou menos tempo para ser executada. Nesse caso, a paralelização funciona de forma aleatória, por isso tem um melhor

desempenho. Ao invés de utilizar a biblioteca rand, utilizei o seguinte gerador de números aleatórios:

```
std::random_device rd;
std::mt19937 generator(rd() ^ omp_get_thread_num());
std::uniform_real_distribution<double> distribution(0.0, 1.0);
```

Essa segunda versão de paralelização trabalhou melhor.

Conclusão

Comparando os “pis” e tempos de execução

	Pi sequencial	V1 de paralelização	V2 de paralelização
Pi	3.14152	3.13936	3.1472
Tempo de execução	0.00302902	0.0161417	0.00498059

1. Houve uma melhoria significativa no tempo de execução entre a versão sequencial e as versões paralelas?

Em questão de eficiência e questão de tempo, o pi sequencial teve melhor desempenho.

2. A estimativa de pi permaneceu precisa em todas as versões?

Não, na primeira versão paralelizada teve um erro considerável

3. Quais foram os maiores desafios ao paralelizar o algoritmo, especialmente em relação aos números aleatórios?

Lidar com a semente e os instantes a serem aleatorizados

4. O uso de threads trouxe benefícios claros para este problema específico?

Não