SuperComp - Atividade 14

Giovana Cassoni Andrade.

Nessa atividade, é trabalhado um algoritmo de sorteios aleatórios para calcular o pi chamado Monte Carlo. Primeiramente é implementada e executada a versão sequencial desse algoritmo, sequencial.cpp, mostrando o valor de pi estimado e o tempo de execução na Tabela 1.

	Estimativa de PI	**Tempo**
sequencial	3.1492	0.0272498 segundos

Tabela 1 - Valor estimado de pi e tempo de execução do código de implementação sequencial.

Analisando o código, a sequência de números aleatórios foi gerada corretamente usando o gerador std::mt19937 com uma seed baseada no tempo atual através da função chrono::system_clock::now().time_since_epoch().count(), garantindo uma sequência única a cada execução. Isso assegura que os números aleatórios sigam uma distribuição uniforme no intervalo [0, 1], como esperado. Não houve dificuldades na implementação, que está bem estruturada e funcional, com o código cumprindo seu objetivo de estimar o valor de Pi utilizando o método de Monte Carlo e medir o tempo de execução.

Em seguida, é feita a paralelização do algoritmo, paralelizacao.cpp, usando a técnica de for paralelo tratando a variável sum com uma operação de redução, novamente obtendo o valor de pi estimado e o tempo de execução na Tabela 2.

	Estimativa de PI	**Tempo**
paralelização	3.13816	0.0190663 segundos

Tabela 2 - Valor estimado de pi e tempo de execução do código de implementação paralela.

A geração de números aleatórios em ambientes paralelos pode ser um obstáculo devido ao uso de geradores de números aleatórios que compartilham um estado global. Esse compartilhamento pode gerar números repetidos ou padrões indesejados, comprometendo a aleatoriedade. Cada thread que acessa o gerador precisa sincronizar para evitar conflitos, o que reduz o paralelismo e aumenta o overhead de gerenciamento das threads.

Para resolver esse problema, foi usada a classe std::mt19937 junto com o uniform_real_distribution, garantindo que cada thread tenha seu próprio gerador de números aleatórios independente. Porém, a forma inicial com que o gerador foi configurado

gera o mesmo gerador de números aleatórios para todas as threads, o que pode acabar causando o problema de repetição dos números aleatórios.

Quanto ao impacto no desempenho, o uso de geradores de números aleatórios independentes com a paralelização melhorou a qualidade dos números gerados, bem como o tempo.

Por fim, são realizadas mudanças no algoritmo paralelizado feito anteriormente para uma outra abordagem de paralelização, melhoria.cpp, dando a cada thread o seu próprio gerador de números aleatórios, apresentando na Tabela 3 o valor de pi estimado com a melhoria e o tempo dessa nova versão paralela.

	Estimativa de PI	**Tempo**
melhoria	3.14156	0.0192305 segundos

Tabela 3 - Valor estimado de pi e tempo de execução do código de implementação paralela melhorada.

As mudanças realizadas no algoritmo visam resolver o problema da geração de números aleatórios em paralelo, garantindo que cada thread tenha um gerador de números aleatórios independente com seeds diferenciadas, adicionando o identificador da thread (omp_get_thread_num()) à seed inicial do gerador mt19937.

Em comparação ao código anterior, a geração de números aleatórios foi paralelizada de maneira eficaz, pois cada thread opera de forma independente, sem a necessidade de sincronizações frequentes.

Quanto aos valores da tabela, o valor de pi estimado não sofreu mudanças significativas, no entanto, essa nova abordagem pode oferecer uma leve melhoria na precisão ao evitar padrões repetidos ou colapsos na aleatoriedade, e o tempo de execução, não houve uma melhoria significativa.

Ao observar a Tabela 4, uma compilação das outras 3 tabelas anteriores, pode-se afirmar algumas coisas, que são explicadas em seguida.

	Estimativa de PI	**Tempo**
sequencial	3.1492	0.0272498 segundos
paralelização	3.13816	0.0190663 segundos
melhoria	3.14156	0.0192305 segundos

Tabela 4 - Valor estimado de pi e tempo de execução dos 3 códigos implementados.

Conclui-se que sobre o tempo de execução entre a versão sequencial e as versões paralelas houve uma melhoria razoável, apesar de não ser extrema, e que a estimativa de pi permaneceu bem precisa em todas as versões, com a versão paralelizada melhorada um pouco mais precisa.

Ao paralelizar o algoritmo de Monte Carlo para estimar Pi, o principal desafio foi garantir a geração correta de números aleatórios em um ambiente multithread, sendo necessário atribuir um gerador de números aleatórios independente para cada thread, como o mt19937, com seeds diferentes.

A paralelização permitiu que múltiplas threads trabalhassem ao mesmo tempo no sorteio de pontos, mas os benefícios de desempenho foram limitados pela sobrecarga

associada à criação e manutenção dos geradores de números aleatórios para cada thread. Apesar disso, o valor estimado de Pi não mudou de forma significativa, mas a precisão foi melhorada com a abordagem paralela.