UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

LEONARDO GONÇALVES - 111337097
KATHLEEN SANTANA - 113163232

COMPUTAÇÃO CONCORRENTE

APROXIMAÇÃO DE PI UTILIZANDO O MÉTODO DE MONTE CARLO

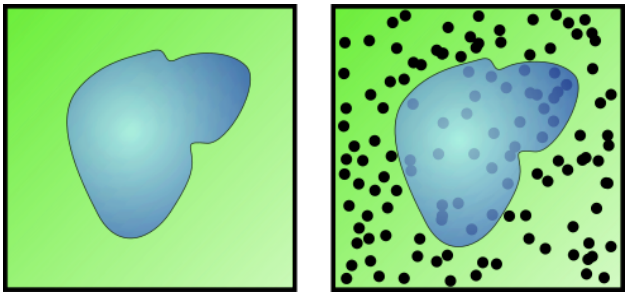
2021

Introdução

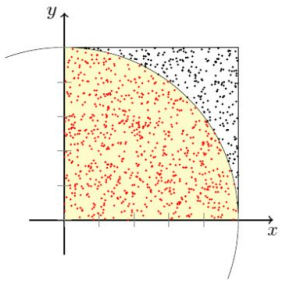
Neste trabalho iremos apresentar uma forma para calcular o valor de pi através do método de Monte Carlo. Para complementar os resultados obtidos, também calculamos uma aproximação utilizando a aproximação através de uma série infinita, conhecida como série de Gregory-Leibniz (também conhecida como série de Madhava-Leibniz). Conseguimos alcançar precisão de até oito casas decimais com o método de Monte Carlo.

Método de Monte Carlo

O método de Monte Carlo pode ser entendido como qualquer método de uma classe de métodos estatísticos que se baseiam em amostragens aleatórias consideravelmente grandes para solucionar problemas de modo a obter resultados numéricos. Com ele, podemos resolver alguns problemas interessantes, como encontrar a área aproximada de uma curva ou estimar o número de peixes em um lago, por exemplo. Esse método pode ser aplicado para solucionar problemas de Física, Medicina, Engenharia, Biologia Computacional, Computação gráfica, Estatística aplicada e Inteligência artificial para jogos, apenas para citar algumas das áreas que podem se aproveitar do seu comportamento.



O método consiste no lançamento de pontos aleatórios em um espaço limitado. Para o nosso caso, onde queremos estimar pi, modelamos um círculo de raio unitário inscrito em um quadrado e geramos pontos aleatórios dentro desse quadrado, que podem estar dentro ou fora do círculo. Com uma grande quantidade de pontos, a razão dos pontos dentro do círculo pelo total de pontos gerados se aproxima de um quarto da razão do círculo pela área do quadrado. Dessa maneira nós podemos estimar o valor de pi numericamente, utilizando a geração de números pseudo-aleatórios em uma distribuição uniforme entre 0 e 1. Como todos os números são positivos, a área analisada é apenas a do primeiro quadrante. Devemos notar que isso não interfere na razão entre os números de pontos dentro do círculo e o número de pontos dentro do quadrado já que ambos (círculo e quadrado) tem sua área reduzida a um quarto, mantendo a razão constante.



Saber se um ponto está dentro do círculo é relativamente simples, bastando calcular a distância do ponto até a borda do círculo. Ou seja, para cada ponto, pegamos sua coordenada (x,y) no plano cartesiano e verificamos se a soma do quadrado dos termos é menor ou igual ao raio ao quadrado. Assim, basta que se calcule $x^2 + y^2 <= 1^2$ para verificar se o ponto está dentro do círculo ou não.

Método de aproximação por Série de Madhava-Leibniz

Podemos utilizar a série abaixo para computar uma quantidade significativa de termos de pi.

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{\pi}{4}$$

Essa série converge lentamente para pi. Para obter 10 dígitos decimais, é necessário calcular cerca de 5 x 10^9 termos. Ou seja, apesar dela convergir lentamente, sabemos que a série permite que encontremos um número razoável de termos para a aproximação.

Paralelizando os algoritmos

Para paralelizar os métodos utilizados neste trabalho, utilizamos aproximações bastante semelhantes a serem discutidas aqui.
A começar pela paralelização do método de Monte Carlo, vamos começar deixando aqui seu pseudo-código:

1. Inicializar pontos_círculo, pontos_quadrado e número de pontos a serem gerados N.

2. Para i <= N:

3. Gerar ponto x aleatório.

4. Gerar ponto y aleatório.

5. Calcular d = x*x + y*y.

6. Se d <= 1, incrementar pontos_círculo.

7. Calcular pi = 4*(pontos_círculo/pontos_quadrado).

8. Encerrar

Para paralelizar esse algoritmo, um ponto deve ser levado em consideração é a geração de número aleatórios. É importante utilizar algoritmos de geração de números pseudo-aleatórios que sejam paralelizáveis, ou sua solução pode sofrer problemas (de sincronização e tempo de execução).
Passando para a divisão de tarefas entre as threads, optamos por evitar condições de corrida na aplicação, fazendo com que cada uma das threads utilizadas calculasse uma parcela da quantidade de pontos que a serem lançados e incrementa um contador local que represente a quantidade de pontos que estão dentro do círculo. Ao fim da computação, retornamos esse valor da thread e incrementamos uma variável global.

Paralelizar a soma dos elementos da série de Madhava-Leibniz foi feito de maneira análoga à utilizada no método de Monte Carlo.
Mas aqui, temos um ponto importante sobre o trabalho realizado por cada thread. Nesse caso, nós realizamos a soma dos termos da série de trás para frente (do menor valor para o maior valor), buscando diminuir a propagação de erro numérico na soma dos termos.

Testes Realizados

Após desenvolver a aplicação, os métodos foram executados com os mesmos parâmetros para base de comparação. Foram utilizados lançamentos de 10^3, 10^5, 10^7 e 10^9 termos, com 1, 2 e 4 threads. Para cada aplicação, foram tomadas cinco execuções de cada algoritmo, e vamos utilizar o que apresentou menor tempo de execução. Todos os testes foram realizados em um computador com processador Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz.

Resultados obtidos

Aqui podemos ver o ganho ao paralelizar o algoritmo de Monte Carlo para duas e quatro threads. Todos os valores estão em segundos.

Ganho de aceleração do Método de Monte Carlo

Termos/ Threads	1	2	Aceleração
10^3	0,000292	0,000106	2,75
10^5	0,006165	0,002947	2,09
10^7	0,619614	0,27143	2,28
10^9	61,858966	27,31274	2,26

Termos/ Threads	1	4	Aceleração
10^3	0,000292	0,000097	3,01
10^5	0,006165	0,001688	3,65
10^7	0,619614	0,138284	4,48
10^9	61,858966	13,773739	4,49

E aqui podemos ver o ganho de paralelizar o método para aproximação usando a soma dos termos de uma série de Madhava-Leibniz.

Ganho de aceleração da Série de Madhava-Leibniz

Termos/ Threads	1	2	Aceleração
10^3	0,000088	0,000075	1,17
10^5	0,002108	0,000807	2,61
10^7	0,152827	0,075008	2,04
10^9	15,002644	7,472333	2,01

Termos/ Threads	1	4	Aceleração
10^3	0,000088	0,000075	1,17
10^5	0,002108	0,000502	4,20
10^7	0,152827	0,038239	4,00
10^9	15,002644	3,765197	3,98

Como podemos verificar, ambas tiveram um ganho bastante semelhante de desempenho de velocidade.
Agora precisamos comparar os valores de pi que foram encontrados nessas aproximações.

Precisão dos termos calculados pelo Método de Monte Carlo

Termos/Threads	1	2	4
10^3	3,14000000000	3,12800000000	3,15200000000
10^5	3,14444000000	3,14159265359	3,14484000000
10^7	3,14037560000	3,14244160000	3,14074280000
10^9	3,14149798000	3,14159265359	3,14171516800

Precisão dos termos calculados pela Série de Madhava-Leibniz

Termos/Threads	1	2	4
10^3	3,1425916543395432	3,1465876583355400	3,1572365803449500
10^5	3,1416026534897900	3,1416426530897900	3,1417493179787200
10^7	3,1415927535897800	3,1415931535897400	3,1415942202562300
10^9	3,1415926545897900	3,1415926585897900	3,1415926692564600

Podemos notar que os valores de aproximação encontrados com a Série de Madhava-Leibniz se aproximam muito mais do valor de pi calculado para as quantidades de termos calculados. Isso se dá pela forma que como calculamos a aproximação em cada método. Sendo assim, para conseguir melhores aproximações para o método de Monte Carlo, seria preciso calcular um número muito maior de termos do que o usado pela nossa simulação. Uma forma de verificar a velocidade com que esse método converge com mais casas decimais seria utilizar o lançamento de uma quantidade muito maior de pontos e verificar a frequência com a qual conseguimos números cada vez mais precisos. Ao fazer

isso, notamos que a frequência de resultados obtidos seguem a tendência de uma curva normal.

Conclusão

De acordo com os dados coletados, é possível calcular o valor de pi com uma quantidade razoável de termos corretos de maneira relativamente rápida e eficiente. Em casos onde queremos provar a aproximação de forma rápida, podemos utilizar o método de Monte Carlo com uma quantidade não tão grande de termos, mas perderemos em precisão devido a natureza da solução. No caso em que precisamos encontrar um maior número de casas decimais, a aproximação pela soma de termos da série nos traz uma resposta consideravelmente rápida para a mesma quantidade de termos utilizados pelo método de Monte Carlo. Então, caso a necessidade seja encontrar muitas casas decimais de forma rápida, podemos utilizar a série. Caso queiramos demonstrar a aproximação, utilizamos Monte Carlo.

Referências

Wikipedia. drand48_r(3) — Linux manual page. man7.org. Disponível em: https://man7.org/linux/man-pages/man3/drand48_r.3.html. Acesso em: 7 jan. 2021.

Wikipedia. Monte Carlo method. Wikipedia. Disponível em: https://en.wikipedia.org/wiki/Monte_Carlo_method. Acesso em: 7 jan. 2021.

Wikipedia. Monte Carlo algorithm. Wikipedia. Disponível em: https://en.wikipedia.org/wiki/Monte_Carlo_algorithm. Acesso em: 7 jan. 2021.

Ingargiola, Giorgio . CIS 307: Pthreads: Thread-safe Versions of POSIX.1 and C-language Functions. Disponível em: https://cis.temple.edu. Acesso em: 7 jan. 2021.

The Open Group. Thread-safety and POSIX.1: Thread-safe Versions of POSIX.1 and C-language Functions. Disponível em: http://www.unix.org/whitepapers/reentrant.html. Acesso em: 7 jan. 2021.

Jonathan Borwein, David Bailey & Roland Girgensohn, *Experimentation in Mathematics - Computational Paths to Discovery*, A K Peters 2003, ISBN 1-56881-136-5, pages 28–30.