Implementando Algoritmo de Monte Carlo com Paralelismo em C++

Danilo Sanchez Tuzita
(danilo_st@hotmail.com)

I. Resumo

Esse é um trabalho que tem como objetivo implementar os algoritmo de Monte Carlo em *C++* para integração numérica usando métodos de paralelismo para aumentar a performance.

II. Introdução

O cálculo numérico de integrais pode ser mais fácil e rápido de se obter um resultado do que calcular de forma analítica. Porém deve-se levar em conta que esse cálculo não será completamente preciso.

III. Teoria

Para o entendimento desse trabalho é necessário conhecimentos básico de cálculo.

IV. Proposta e Implementação

O Método de Monte Carlo utiliza de números aleatórios e chance para calcular uma integral. É tirado várias amostras em pontos aleatórios da função dentro do intervalo que se quer integrar a função e calculado a média das amostras, como demonstra a fórmula 1, onde n é a quantidade de "chutes" e x_i um valor aleatório no intervalo $[a, b]$.

$$\int_a^b f(x)\,dx \approx (b-a) \times \frac{1}{n} \times \sum_{i=1}^{n} f(x_i) \tag{1}$$

Uma das desvantagens desse algorítimo é que ela é muito dependente da função que se quer integrar, se essa não for "bem comportada" é possível que a média se desvie drasticamente pois por chance foi escolhido um x_i num pico ou vale da função, fazendo com que o resultado não seja tão próximo ao valor real. Porém isso pode ser combatido aumentando o valor de n.

O Método de Monte Carlo, também pode calcular integrais multidimensionais, porém, diferentemente das integrações unidimensionais, é preciso de uma função auxiliar g que retornará se as coordenadas passadas estão dentro ou não da função.

Por exemplo, considerando que se quer calcular o valor de π, teremos a função g demonstrada pela fórmula 2, calculamos a aproximação de π utilizando a fórmula 3.

$$g(x,y) = \begin{cases} 1, & \text{se } x^2 + y^2 \leq 1 \\ 0, & \text{caso contrario} \end{cases} \tag{2}$$

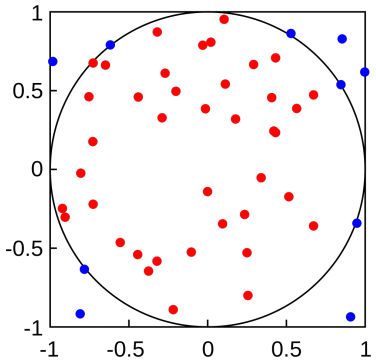

Fig. 1. Representação do Método Monte Carlo para cálculo de pi. Fonte: Wikipédia

$$\int_a^b \int_c^d f(x,y)\,dx\,dy \approx (b-a) \times (d-c) \times \frac{1}{n} \times \sum_{i=1}^{n} g(x_i, y_i) \tag{3}$$

Esse cálculo pode ser representado pela figura 1, onde os pontos vermelhos indicam que as coordenadas aleatórias geradas estão dentro da área de interesse. Com essas amostras é calculado a proporção de quantas das coordenadas caíram dentro da área do circulo vezes a área total do domínio de teste.

A. Paralelismo

Pelo fato de cada iteração do algorítimo usar valores aleatórios para o seu calculo, as iterações se tornam totalmente independentes uma das outras. Com isso, nesse trabalho, foi implementado o método de Monte Carlo com paralelismo em mente, pois a maioria de suas operações são independentes. Foi utilizado a biblioteca MPI em C++. Cada nó processa $\frac{1}{n}$ iterações e no final é calcula-se a média da solução de cada nó.

V. Resultados

Para testar o Método proposto, foi calculado a integral de duas funções e o volume de uma intersecção entre um toroide e um cubo. Para os testes foi utilizado apenas uma máquina com um *i5-6500 3.2GHz, 4 núcleos*, apesar da biblioteca suportar processamento em paralelo em múltiplas máquinas. Cada experimento consiste do cálculo de cada integral para iterações $n = \{10^2 - 10^8\}$, com diferentes contagens de *threads*.

TABLE I
RESULTADOS EXPERIMENTO 1A

Experiment 1a.			
Threads	**1**	**Threads**	**2**
Iterations	**Result**	**Iterations**	**Result**
100	3.062506	100	3.120475
1000	3.129705	1000	3.129128
10000	3.143852	10000	3.143251
100000	3.143789	100000	3.143075
1000000	3.141701	1000000	3.140935
10000000	3.141583	10000000	3.141560
100000000	3.141569	100000000	3.141486
Time	*16.46843 s*	*Time*	*8.161513 s*
Threads	**4**	**Threads**	**8**
Iterations	**Result**	**Iterations**	**Result**
100	3.111144	100	2.999217
1000	3.099512	1000	3.167914
10000	3.151772	10000	3.115333
100000	3.137877	100000	3.134642
1000000	3.142671	1000000	3.138915
10000000	3.141144	10000000	3.142072
100000000	3.141652	100000000	3.141468
Time	*4.921759 s*	*Time*	*4.652310 s*

TABLE II
RESULTADOS EXPERIMENTO 1B

Experiment 1b.			
Threads	**1**	**Threads**	**2**
Iterations	**Result**	**Iterations**	**Result**
100	1.035658	100	1.061828
1000	1.034073	1000	1.024322
10000	1.045930	10000	1.047984
100000	1.044759	100000	1.046520
1000000	1.045265	1000000	1.045285
10000000	1.045237	10000000	1.045382
100000000	1.045275	100000000	1.045272
Time	*14.970445 s*	*Time*	*7.268767 s*
Threads	**4**	**Threads**	**8**
Iterations	**Result**	**Iterations**	**Result**
100	1.073535	100	1.023526
1000	1.060576	1000	1.024151
10000	1.044722	10000	1.034570
100000	1.045832	100000	1.041491
1000000	1.044776	1000000	1.046187
10000000	1.045154	10000000	1.044556
100000000	1.045333	100000000	1.045391
Time	*4.518436 s*	*Time*	*4.447252 s*

A. Experimento 1

As funções calculadas com o método de Monte Carlo foram as seguintes integrais 4 e 5:

$$\int_0^1 \frac{4}{1+x^2}\,dx \tag{4}$$

$$\int_0^1 \sqrt{x+\sqrt{x}}\,dx \tag{5}$$

1) a: A tabela I demonstra os resultados obtidos para o calculo da integral 4. Nota-se que quanto mais iterações o resultado mais se aproxima de π, o resultado analítico dessa integral. Também pode-se notar o tempo significantemente mais curto para os processamentos em paralelo.

2) b: A tabela II demonstra os resultados obtidos para o calculo da integral 5. Diferentemente do Experimento 1a. o cálculo dessa integral de modo geral, com apenas 100 iterações, já se aproxima consideravelmente do resultado final. Isso de deve ao fato dessa função ser mais "comportada" do que a função do experimento anterior.

TABLE III
RESULTADOS EXPERIMENTO 2

Experiment 2.			
Threads	**1**	**Threads**	**2**
Iterations	**Result**	**Iterations**	**Result**
100	22.68	100	25.2
1000	21.672	1000	22.512
10000	22.8228	10000	21.504
100000	22.04412	100000	22.02816
1000000	22.084272	1000000	22.063776
10000000	22.118292	10000000	22.094268
100000000	22.094325	100000000	22.104509
Time	*50.871556 s*	*Time*	*27.623872 s*
Threads	**4**	**Threads**	**8**
Iterations	**Result**	**Iterations**	**Result**
100	13.44	100	35
1000	19.152	1000	12.096
10000	21.672	10000	22.3104
100000	22.12224	100000	22.33056
1000000	22.143072	1000000	22.010688
10000000	22.094318	10000000	22.084138
100000000	22.094678	100000000	22.09153
Time	*14.845529 s*	*Time*	*14.564261 s*

B. Experimento 2

Para o cálculo do volume da intersecção de um toroide com um cubo, foi dado a fórmula 6 e a figura 2. Com isso podemos descobrir que o cálculo do volume da intersecção pedida pode ser descrita pela fórmula 7.

$$g(x,y,z) = \begin{cases} 1, & \text{se } z^2 \times \left(\sqrt{x^2+y^2}-3\right) \leq 1 \\ 0, & \text{caso contrario} \end{cases} \tag{6}$$

$$\int_{-1}^{1}\int_{-3}^{4}\int_{1}^{4} f(x,y,y)\,dx\,dy\,dz \approx$$

$$(4-3)\times(4-(-3))\times(1-(-1))\times\frac{1}{n}\times\sum_{i=1}^{n} g(x_i,y_i,z_i) \tag{7}$$

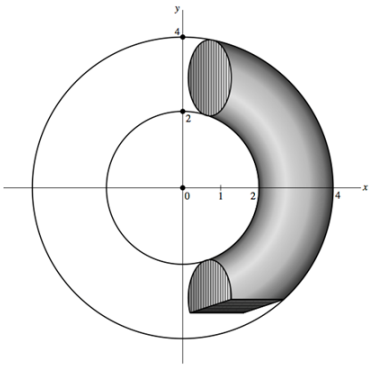

Fig. 2. Representação da intersecção do toroide e cubo a ser calculado

Os resultados desse experimento são demonstrados na Tabela III, para esse experimento nota-se que é necessário uma quantidade razoavelmente maior para se ter uma boa acurácia, se comparado aos experimentos anteriores. Isso se deve ao fato de ser uma integral de uma função multidimensional e seu domínio de onde pode ser amostrado valores é também significativamente maior do que o nos experimentos anteriores.

Assim como nos experimentos anteriores, pode se observar que o tempo de processamento cai consideravelmente de acordo com a quantidade de nós.

VI. Conclusão

A utilização de métodos de integração numérica pode ser muito útil quando é difícil calcular a integral analiticamente especialmente o calculo de integrais em múltiplas dimensões.

O Método de Monte Carlo resolve esse problema atingindo uma precisão razoável a um baixo custo computacional. Além disso pode-se usar paralelismo para o processamento, pois suas operações são majoritariamente independentes.