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 adaptar o algoritmo de Monte Carlo, implementados no relatório anterior usando métodos de paralelismo para aumentar a performance.

II. Introdução

O algorítimo de integração numérica de Monte Carlo é um ótimo candidato para ser processado em paralelo, pois muitas das operações desse algorítimo são independentes umas das outras.

III. Teoria

Para o entendimento desse trabalho é necessário conhecimentos básico de computação paralela.

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.

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 uma amostra em um pico ou vale da função, fazendo com que o resultado não seja tão próximo ao valor real. Porém isso pode ser combatido aumentando a quantidade de iterações.

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 reexecutado os experimentos do relatório anterior. 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*.

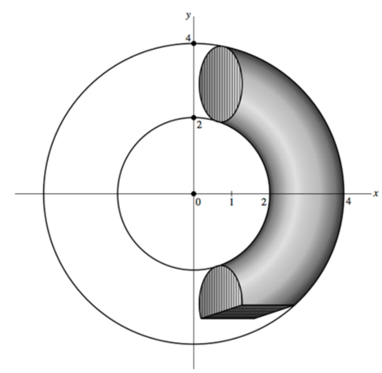

Fig. 1. Representação da intersecção do toroide e cubo a ser calculado

A. Experimento 1

As funções calculadas com o método de Monte Carlo foram as seguintes integrais 1 e 2:

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

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

A tabela I demonstra os resultados obtidos para o calculo da integral 1 e a tabela II demonstra os resultados obtidos para o calculo da integral 2.

B. Experimento 2

Para o cálculo do volume da intersecção de um toroide com um cubo, foi dado a fórmula 3 e a figura 1. Com isso podemos descobrir que o cálculo do volume da intersecção pedida pode ser descrita pela fórmula 4.

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

$$\int_{-1}^{1}\int_{-3}^{4}\int_{1}^{4} f(x,y,y)\,dx\,dy\,dz \approx$$
$$(4-3)\times(4-(-3))\times(1-(-1))\times\frac{1}{n}\times\sum_{i=1}^{n} g(x_i,y_i,z_i) \tag{4}$$

Os resultados desse experimento são demonstrados na Tabela III.

TABLE I
RESULTADOADOS EXPERIMENTO 1A

Experimento 1a.			
Threads	**1**	**Threads**	**2**
Iterações	**Resultado**	**Iterações**	**Resultado**
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
Tempo Total	*16.46843 s*	*Tempo Total*	*8.161513 s*
Threads	**4**	**Threads**	**8**
Iterações	**Resultado**	**Iterações**	**Resultado**
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
Tempo Total	*4.921759 s*	*Tempo Total*	*4.652310 s*

TABLE II
RESULTADOADOS EXPERIMENTO 1B

Experimento 1b.			
Threads	**1**	**Threads**	**2**
Iterações	**Resultado**	**Iterações**	**Resultado**
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
Tempo Total	*14.970445 s*	*Tempo Total*	*7.268767 s*
Threads	**4**	**Threads**	**8**
Iterações	**Resultado**	**Iterações**	**Resultado**
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
Tempo Total	*4.518436 s*	*Tempo Total*	*4.447252 s*

TABLE III
RESULTADOADOS EXPERIMENTO 2

Experimento 2.			
Threads	**1**	**Threads**	**2**
Iterations	**Resultado**	**Iterations**	**Resultado**
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
Tempo Total	*50.871556 s*	*Tempo Total*	*27.623872 s*
Threads	**4**	**Threads**	**8**
Iterations	**Resultado**	**Iterations**	**Resultado**
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
Tempo Total	*14.845529 s*	*Tempo Total*	*14.564261 s*

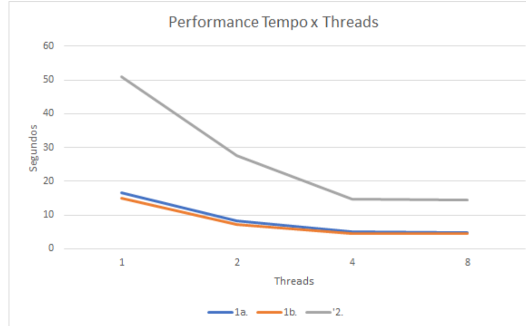

Fig. 2. Performance dos Experimentos

VI. CONCLUSÃO

Pode-se notar o tempo é significantemente mais curto quando os algorítimos são executados em paralelo. Pode se notar que o tempo de processamento tem a tendência de ser duas vezes mais rápido quando também é duplicado a quantidade de *threads*, até o ponto onde a quantidade de *threads* excede a quantidade de núcleos de processamento como demonstra a Figura 2.

Com o desenvolvimento desse trabalho pode-se concluir que o Método de Monte Carlo de integração numérica atinge uma precisão razoável a um baixo custo computacional. Além disso pode-se usar paralelismo para o processamento, pois suas operações são majoritariamente independentes.