Cálculo de Pi através do método de Monte Carlo

Sistemas Operativos – Ano Letivo 2018/2019

Diogo Mendes, 30003865

Vlas Safronov, 30003577

Manuel Duarte, 30004361

Moisés Espílio, 30003916

Professor: Gonçalo Valadão

Introdução

O método de Monte Carlo consiste em obter um resultado numérico através de amostragem aleatória, neste trabalho pretende-se calcular o valor de pi utilizando este método. Para esse efeito considera-se um quadrado no qual está inscrito um círculo. Sendo *r* o raio do círculo, o lado do quadrado é 2*r*. As áreas do quadrado e do círculo são respetivamente, A(quadrado) = (2r) 2 = 4r e A(círculo) = πr^2. Se dividirmos a área do círculo pela área do quadrado iremos obter $\frac{A\,(círculo)}{A\,(quadrado)} = \frac{\pi}{4}$, sendo que π = 4 × $\frac{A\,(círculo)}{A\,(quadrado)}$. Para esse efeito geram-se "n" pontos aleatórios dentro do quadrado, contando-se quantos desses estão no círculo e dividir essa contagem pelo total de pontos gerados. Desta forma iremos obter um valor próximo para a divisão entre $\frac{A\,(círculo)}{A\,(quadrado)}$.

Para a execução deste trabalho utilizou-se a linguagem de programação C, um sistema operativo Linux e considerou-se o valor de π real para o número de algarismos que obtemos (7) como sendo o valor tabelado no excel – 3,141593.

Descrição do trabalho realizado

Para realizar as funcionalidades desejadas começamos por incluir as bibliotecas stdio, stdlib, time e pthread. As duas primeiras permitem-nos usar as funções mais básicas da linguagem C enquanto que "time" nos permite fazer contagens de tempo e "pthread" é a biblioteca que permite o paralelismo (criando diversos threads).

Declaram-se duas variáveis globais, "quantidade" e "threads" que nos permitem variar o número de pontos que vamos gerar e o número de threads utilizado para todo o processo, estas variáveis nunca serão alteradas enquanto o programa estiver em execução.

Função main():

```
clock_t time_start, time_end;
void *Acirculo[threads];
srand(time(NULL));
```

Na primeira instrução as variáveis do tipo clock_t pertencem ao módulo "time" e são utilizadas para conter os valores temporais que a função clock() retorna. A segunda instrução cria um Array do tamanho da variável threads que receberá os valores retornados pelos threads. A terceira instrução prepara o gerador aleatório (rand()) para utilização.

As restantes instruções são executadas dentro de um círculo for que itera 20 vezes, este ciclo permite realizar várias simulações com uma só execução do programa poupando algum tempo na realização da experiência.

```
int Area=0;
double calc;
pthread_t ids[threads];     //cria thread ids para cad
time_start = clock();   //guarda o tempo antes de cria
for(int i=0; i<threads; ++i)
{
    pthread_create(&ids[i], NULL, &randomizer, NULL);
}
for(int i=0; i<threads; ++i)
{
    pthread_join(ids[i], &Acirculo[i]);  //aguarda que
}
```

Começa-se por criar um array com tamanho igual ao número de threads do tipo pthread_t que contem um thread id para cada thread que será criado. Regista-se o tempo atual através da função clock().

Recorrendo a um ciclo for criamos todos os threads (função pthread_create()) necessários com o respetivo thread id contido no array "ids" e realizando a função "randomizer" explicada em baixo. Novamente recorrendo ao ciclo for aguarda-se que os threads concluam a sua tarefa e recolhendo o valor do seu contador para o array Acirculo creado no início do programa.

Finalmente soma-se o total de pontos gerados dentro do círculo somando todos os valores dentro do array Acirculo, calcula-se o valor de pi (calc) afixando-o no ecrã, regista-se o tempo final, novamente recorrendo à função clock, e calcula-se a duração da experiência afixando-a no ecrã.

```
for(int i=0; i<threads; ++i)
{
    Area= *((int*)Acirculo[i]) + Area; //conta todos os pontos c
}
calc=(Area/quantidade)*4;           //calcula o valor de pi
printf("\n%f", calc);       //imprime o valor de pi no ecrã
time_end=clock();           //guarda o tempo no fim do programa
printf("\n%f",(double) (time_end - time_start)/CLOCKS_PER_SEC);
```

Função randomizer():

A função “randomizer” será a função que cada thread irá executar. Nesta função declaram-se 4 variáveis locais, Acirculo – variável utilizada para contar o número de pontos do círculo da thread local - e x e y que serão as coordenadas do ponto gerado.

```
for (int i=0; i<=quantidade/threads; ++i) //gera pont
{
    x=(double)rand()*(1-(-1))/(double)RAND_MAX+(-1);
    y=(double)rand()*(1-(-1))/(double)RAND_MAX+(-1);
    if ((x*x+y*y)<=1)                   //verifica se
    {
        Acirculo++;  //incrementa a variável que guar
    }
}
*v=Acirculo;
return (void *) v;
```

Depois recorrendo a um ciclo for geramos o número de pontos desejado que será a quantidade total de pontos a dividir pelo número de threads que estão a realizar o cálculo. Gera-se um ponto entre dentro das dimensões do quadrado recorrendo à função rand() e alguma feitiçaria matemática, verificasse se o ponto está dentro do círculo (como o circulo é de centro no ponto (0,0) e o raio 1 podemos aplicar a função desta forma mais simplificada) e se estiver incrementa-se a variável de contagem.

Finalmente retornamos um apontador que indica a localização da variável de contagem.

Resultados

Para obtermos estes dados, corremos o programa 20 vezes para cada quantidade de pontos e cada número de threads.

Pontos = 20000

	2 Threads	4 Threads	6 Threads	8 Threads
Média Tempo(segundos)	0,017848263	0,01382125	0,01719695	0,01561385
Média Valor de Pi	3,1535053	3,1532	3,14817	3,141
Erro relativo(%)	37,92%	36,95%	20,94%	1,89%

Pontos = 100000

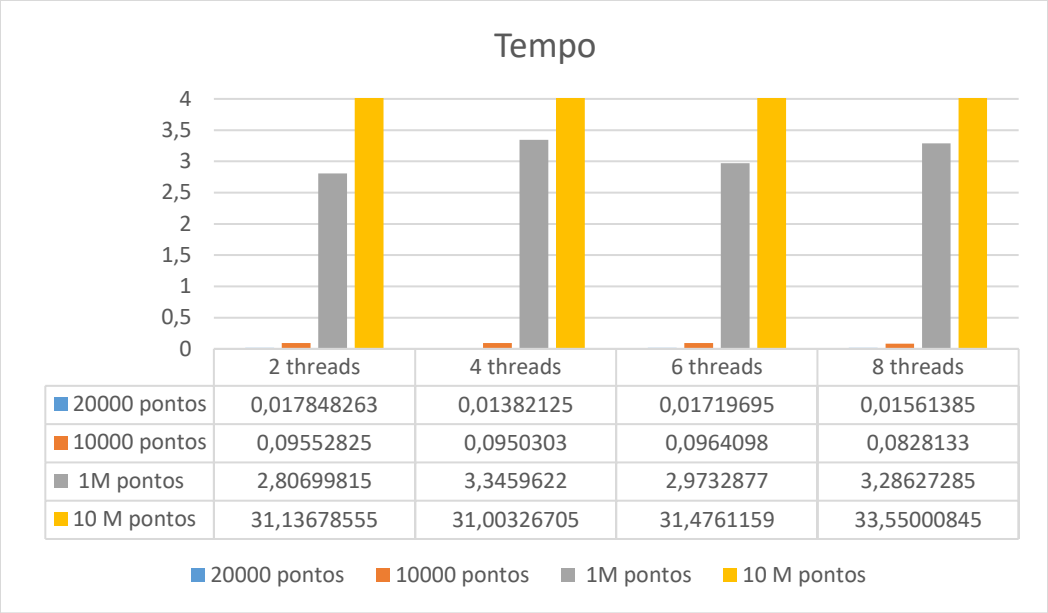
	2 Threads	4 Threads	6 Threads	8 Threads
Média Tempo(segundos)	0,09552825	0,0950303	0,0964098	0,0828133
Média Valor de Pi	3,141258	3,142688	3,141392	3,141614
Erro relativo(%)	1,07%	3,49%	0,64%	0,07%

Pontos = 1 milhão

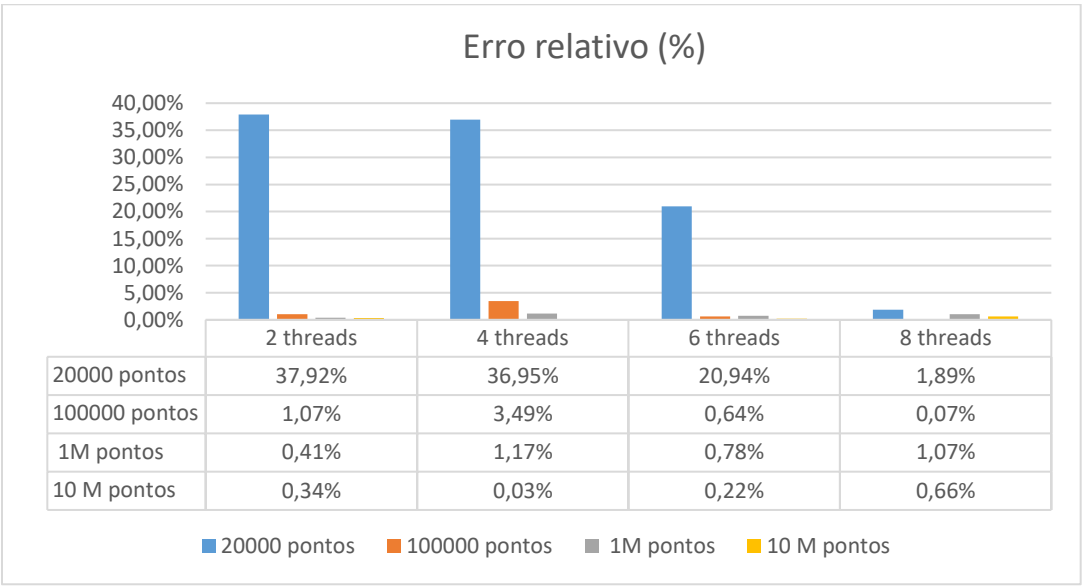
	2 Threads	4 Threads	6 Threads	8 Threads
Média Tempo(segundos)	2,80699815	3,3459622	2,9732877	3,28627285
Média Valor de Pi	3,1417224	3,1412246	3,1418364	3,141929
Erro relativo(%)	0,41%	1,17%	0,78%	1,07%

Pontos = 10 milhões

	2 Threads	4 Threads	6 Threads	8 Threads
Média Tempo(segundos)	31,13678555	31,00326705	31,4761159	33,55000845
Média Valor de Pi	3,1414861	3,1415831	3,1416623	3,14138555
Erro relativo(%)	0,34%	0,03%	0,22%	0,66%



	2 threads	4 threads	6 threads	8 threads
20000 pontos	0,017848263	0,01382125	0,01719695	0,01561385
10000 pontos	0,09552825	0,0950303	0,0964098	0,0828133
1M pontos	2,80699815	3,3459622	2,9732877	3,28627285
10 M pontos	31,13678555	31,00326705	31,4761159	33,55000845



	2 threads	4 threads	6 threads	8 threads
20000 pontos	37,92%	36,95%	20,94%	1,89%
100000 pontos	1,07%	3,49%	0,64%	0,07%
1M pontos	0,41%	1,17%	0,78%	1,07%
10 M pontos	0,34%	0,03%	0,22%	0,66%

Pontos	Valor médio aproximado de pi (ignorando o número de threads)	Erro relativo (%)
20000	3,148969	23,47906%
100000	3,141738	0,46266%
1M	3,141678	0,27200%
10M	3,141529	0,20185%
Valor real de PI(7 algarismos)	3,141593	

Análise dos Resultados

Em primeiro lugar, podemos verificar que o tempo (em segundos) que o cpu demorava a calcular o resultado, aumenta com a quantidade de pontos. No caso das threads, verificamos que 4 threads é, normalmente, a opção mais rápida, excepto no caso em que tínhamos 1 millhão de pontos, onde 2 threads foi o melhor em termos de tempo, isto demonstra que um número maior de threads nem sempre é favorável á velocidade de processamento das tarefas.

Em relação ao cálculo do valor de pi, através da simulação do monte carlo, a média do valor aproximado de pi é tanto mais próxima quanto maior for a quantidade de pontos. Podemos verificar isto através da tabela final e da tabela dos erros relativos, pois ambas comprovam este facto.

Bibliografia

C Standard Library Reference Tutorial, https://www.tutorialspoint.com/c_standard_library/index.htm, acedido em 25/05/2019

Monte Carlo method, https://en.wikipedia.org/wiki/Monte_Carlo_method, acedido em 25/05/2019