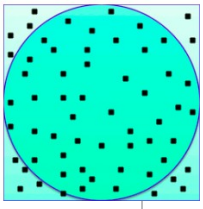
Universidade Autónoma de Lisboa

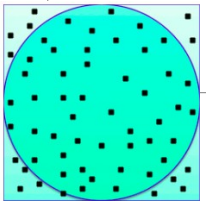
Engenharia Informática

SISTEMAS OPERATIVOS



CÁLCULO DE PI

MÉTODO DE MONTE CARLO



Docente: Gonçalo Valadão Matias

10 de Junho de 2022

Introdução

No sentido de aplicar conceitos adquiridos ao longo do semestre em relação à unidade curricular de “Sistemas Operativos“, foi-nos proposto a elaboração de um trabalho prático.

Relativamente ao enunciado, é pretendida a implementação do algoritmo de simulação de “Monte Carlo”, que funcionará como estimativa do valor de π, recorrendo a várias “threads” e pontos.

Considerando um quadrado e um círculo nele definido, recorrendo ao método de “Monte Carlo”, um certo valor de pontos aleatórios serão gerados. O objetivo é percebermos quantos desses pontos estão dentro do círculo, dividindo essa contagem pelo total de pontos gerados. Multiplicando-se esse resultado por 4, obtém-se assim o valor aproximado de π.

De acordo com o enunciado, temos 4 amostras de pontos e “threads” de execução, obtendo-se vários valores aproximados de π. Temos como principal objetivo analisar, comparar e comentar os resultados obtidos.

Este tipo de simulações são usadas para modelar sistemas financeiros, simular redes de telecomunicações, calcular resultados para integrais de alta dimensão. A técnica de “Monte Carlo” é também usada na modelagem de uma ampla gama de sistemas físicos na vanguarda da pesquisa científica, por exemplo.

Este trabalho foi desenvolvido pelos alunos Paulo Correia (30002232), Eduardo Araújo (30008290), Pedro Amaral (30008241) e João Lucas (30008215), do curso de Engenharia Informática, do 1ºano.

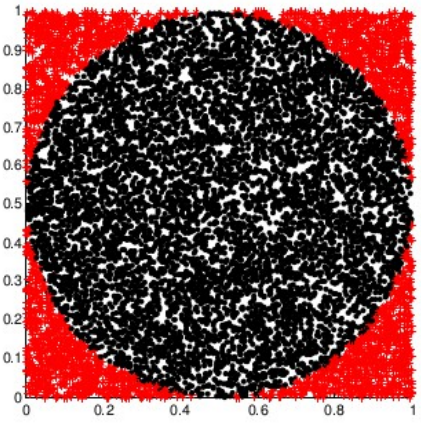


Imagem do site reseachgate.net

Descrição técnica

O trabalho realizado começa por apresentar a assinatura da função “get_pi()”. Esta será usada como “loop” de inicialização de “threads” e será também processada posteriormente pelas “threads”.

```
void *get_pi(void *truns);
```

De seguida, no “main” colocamos primeiramente as várias variáveis que iremos utilizar para descobrirmos aquilo que precisamos, como o valor “aproximado de PI”, tempo de execução, entre outros...

Relativamente à função “clock_gettime”, foi importada da biblioteca “time.h” e irá servir para calcularmos o tempo de execução do programa. Criamos a estrutura “*timespec”,* com as variáveis “start” e “finish”, onde guardamos o tempo inicial e final de execução do programa.

Criando um “array” de “thread_ids”, fazemos uma “memory allocation” do “array” do tamanho de cada “pthread_t” vezes o número de threads que irão ser iniciados.

```
pthread_t *thread_array;
thread_array = malloc(nthreads * sizeof(pthread_t));
nruns = npoints / nthreads;
```

De seguida, começamos a criar o primeiro ciclo, que irá funcionar como “loop” de inicialização de “threads”. Estas, irão correr a função “get_pi” e recebem como parâmetro, o número de pontos que cada “thread” deve gerar.

É também neste momento que é inserido o “thread_id", da thread iniciada, no array.

```
for (i = 0; i < nthreads; i++) {
    pthread_create(&thread_array[i], NULL, *get_pi, (void *) &nruns);
}
```

Em relação ao segundo ciclo, o programa irá aguardar que cada “thread” termine.
Os valores retornados por cada “thread” serão guardados no apontador “pcount”.
Descobrimos o valor de PI com base nos pontos que estão dentro do círculo, bem como com a implementação da fórmula de cálculo de PI.

Sobre o cálculo do erro relativo, utilizámos a função “fabs()”, da biblioteca “math.h” para que o valor da variável “abs_err” seja positivo. Fizemos as devidas operações matemáticas, de acordo com o que era pedido para este cálculo. Obtemos o erro absoluto e de seguida, o erro relativo, multiplicando este por 100, obtendo a percentagem.

Conforme mencionado anteriormente, terminamos a contagem de tempo de execução usando o “&finish”. A variável “elapsed” irá conter o valor do delta entre o tempo final e inicial da execução do programa. A divisão por “1000000000.0” procura encontra um valor “mais rigoroso”.

```
// Terminar a contagem do tempo de execução do programa e determinar o delta.
clock_gettime(CLOCK_MONOTONIC, &finish);
elapsed = (finish.tv_sec - start.tv_sec);
elapsed += (finish.tv_nsec - start.tv_nsec) / 1000000000.0;
```

Obtendo já todos os valores necessários, como o valor aproximado, tempo de execução e erro relativo, enviamos os valores para o terminal.

Conforme mencionada no primeiro parágrafo, a função “get_pi” irá receber a variável “truns”. Declarámos um “void”, pois qualquer que seja a variável que chega às funções, será sempre convertido para “void”.

De seguida declaramos as variáveis que servirão para gerarmos coordenadas e sabermos quais serão os valores que estarão dentro do círculo. Neste caso, estamos a referir-nos ao “x”,“y”,”z”,”count” e às duas “seed’s” (para “x” e para “y”).

A variável “z” irá receber o valor da operação do “x^2” com o “y^2”. O “count” irá conter o número de pontos que estão dentro do circulo, de acordo com a condição “z <= 1” .

```
void *get_pi (void *truns) {
    double x, y, z, pi;
    int i, count = 0;
    long seedx, seedy;
```

Para obtermos os valores de “x” e “y”, foi utilizada a função rand_r() que irá gerar um número aleatório a partir de uma “seed”. A “semente” é um número inteiro que será utilizado como base para os valores aleatórios, o que faz com que o seu valor tenha de ser diferente para cada “thread”, caso contrário todas as “threads” irão gerar os mesmos valores de “x” e “y”. O valor da “seed” de “x” será o “ID” de cada “thread” e a “seed” de “y”, o resultado da subtração do “ID” da “thread” e o “ID” do processo, desta forma as “seed’s” de cada “thread” serão diferentes.

Para que os valores das coordenadas sejam entre 0 e 1, os valores devolvidos pela função “rand_r” são divididos pelo número 2147483647, o número mais alto do tipo “int”.

```
int *thread_runs = (int *) truns;
for (i = 0; i < *thread_runs; i++) {
    x = (double)rand_r((unsigned int*)seedx)/RAND_MAX; //INT_MAX 2147483647
    y = (double)rand_r((unsigned int*)seedy)/RAND_MAX;
    z = x * x + y * y;
    if (z <= 1) count++;
}
```

Por fim, alocámos memória para o apontador “result”, que irá receber um valor inteiro da variável “count”. Este passo torna-se necessário pois as variáveis locais não podem ser retornadas, isto porque o seu espaço de memória é apagado quando a função termina.

```
    int* result = malloc(sizeof(int));
    *result = count;
    return (void *) result;
}
```

Por fim, tendo já sido explicado todo o contexto do código aplicado, consideramos importante explicar as bibliotecas usadas. Em relação as bibliotecas “stdio.h”, “stdlib.h” e “pthread.h”, estas são as responsáveis pelo I/O, a leitura de parâmetros vindos da terminal e utilização de “threads” no programa, respetivamente. A “unistd.h” aplica-se na utilização da função “getpid ()”. A biblioteca “unistd.h” contém a função “getpid()” que é utilizada em “get_pi()”, “time.h” que é necessária para determinarmos o tempo de execução e “math.h” para podermos usar a função “fabs()”.

```c
#include <stdio.h> // Header para operações I/O
#include <pthread.h> // Header para threads
#include <stdlib.h> // Header para a função atol()
#include <unistd.h> // Header para função getpid()
#include <time.h> // Header para função clock_gettime()
#include <math.h> //Header para a função fabs()
```

Apresentação dos resultados

Para 20 000 pontos:

```
lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 20000 2

_____________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 20000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
2               15783           3.15660000      0.000806s       0.47769877


lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 20000 4

_____________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 20000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
4               15709           3.14180000      0.000861s       0.006600156


lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 20000 6

_____________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 20000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
6               15697           3.13940000      0.000797s       0.069794215


lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 20000 8

_____________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 20000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
8               15470           3.09400000      0.002054s       1.5149212
```

Para 100 000 pontos:

```
lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 100000 2

_____  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____

Para 100000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
2               78569           3.14276000      0.002416s       0.037157904


lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 100000 4

_____  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____

Para 100000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
4               78665           3.14660000      0.002999s       0.1593889


lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 100000 6

_____  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____

Para 100000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
6               78755           3.15020000      0.002221s       0.27398047


lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 100000 8

_____  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____

Para 100000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
8               78749           3.14996000      0.002630s       0.26634103
```

Para 1 000 000 pontos:

```
lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 1000000 2

______________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 1000000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
2               784730          3.13892000      0.027399s       0.085073084


lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 1000000 4

______________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 1000000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
4               785287          3.14114800      0.017707s       0.01415365


lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 1000000 6

______________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 1000000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
6               785174          3.14069600      0.020342s       0.028541256


lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 1000000 8

______________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 1000000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
8               785188          3.14075200      0.022257s       0.026758719
```

Para 10 000 000 pontos:

```
lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 10000000 2

_____ CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____

Para 10000000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
2               7857234         3.14289360      0.211988s       0.041410521


lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 10000000 4

_____ CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____

Para 10000000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
4               7855251         3.14210040      0.170490s       0.016162185


lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 10000000 6

_____ CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____

Para 10000000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
6               7855799         3.14231960      0.184067s       0.023139538


lucas@lucas-LeNovo:~/Desktop/ValorPI$ ./ValorPI 10000000 8

_____ CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____

Para 10000000 pontos

Nº Threads      Pontos_dentro   Aprox. Pi       Tempo           Erro relativo
8               7853588         3.14143520      0.177798s       0.0050117895
```

Análise crítica dos resultados

Avaliando os resultados obtidos e considerando os pontos e “threads” indicados, de um modo geral, os tempos de execução diminuem com o aumento do número de “threads”, algo que era esperado por nós, tendo em conta a matéria lecionada pelo professor, durante as aulas. Em relação aos valores avaliados, é normal termos alguns

valores que estão um pouco “fora” do padrão, tendo em conta que os valores são gerados aleatoriamente.

Sobre a “aproximação de PI”, à medida que vamos tendo mais pontos, o resultado vai-se aproximando do valor de PI. De acordo com a nossa primeira amostra, verificamos que, para 8 “threads”, em relação a cada número de pontos:

Para 20 000 – **3.094**
Para 100 000 – **3.1499**
Para 1 000 000 – **3.1407**
Para 10 000 000 - **3.1414**

Tendo em conta que o **valor de PI é 3.1415**, verificamos que os nossos resultados se estão a aproximar do valor original, à medida que vamos aumentando o número de pontos.

No diz respeito ao “erro relativo”, verificamos que este diminui com o aumento dos pontos gerados. Podemos perceber que o menor valor para este campo, se verifica para os 10 000 000 pontos (0.0050117895).
De acordo com a nossa primeira amostra, verificamos que, para 8 “threads”, em relação a cada número de pontos:

Para 20 000 – **1.51**
Para 100 000 – **0.2663**
Para 1 000 000 – **0.0267**
Para 10 000 000 - **0.0050**

De forma a confirmar a nossa avaliação em relação ao “erro relativo”, decidimos gerar 10 amostras de pontos e calcular a respetiva média dos “erros relativos”.

10 Amostras:

```
_____________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 20 000 pontos

Nº Threads      Média do erro relativo
2               0.4993

Nº Threads      Média do erro relativo
4               0.215860895

Nº Threads      Média do erro relativo
6               0.309846663

Nº Threads      Média do erro relativo
8               0.968064755
```

```
_____________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 100 000 pontos

Nº Threads      Média do erro relativo
2               0.202250283

Nº Threads      Média do erro relativo
4               0.083558254

Nº Threads      Média do erro relativo
6               0.267996237

Nº Threads      Média do erro relativo
8               0.252925218
```

```
_____________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 1 000 000 pontos

Nº Threads      Média do erro relativo
2               0.045921294

Nº Threads      Média do erro relativo
4               0.038362389

Nº Threads      Média do erro relativo
6               0.053174302

Nº Threads      Média do erro relativo
8               0.258363397
```

```
_____________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 10 000 000 pontos

Nº Threads      Média do erro relativo
2               0.015828277

Nº Threads      Média do erro relativo
4               0.020768765

Nº Threads      Média do erro relativo
6               0.016937906

Nº Threads      Média do erro relativo
8               0.013112458
```

De acordo com a média das 10 amostras, verificamos que, para 8 “threads”, em relação a cada número de pontos:

Para 20 000 – **0.9680**
Para 100 000 – **0.2529**
Para 1 000 000 – **0.2583**
Para 10 000 000 - **0.01311**

Comprovando a 1ª análise, verificamos que o “erro relativo” diminui com o aumento do pontos gerados.

Comparação Geral

Ainda sobre o “erro relativo”, apresentando uma análise geral dos resultados, mostramos as respetivas tabelas para 20 000 pontos e para 10 000 000 pontos:

```
_____________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 20 000 pontos

Nº Threads      1ª Amostra de erro relativo     Média do erro relativo  Desvio padrão
2               0.47769877                      0.4993                  0.030570159

Nº Threads      1ª Amostra de erro relativo     Média do erro relativo  Desvio padrão
4               0.006600156                     0.215860895             0.4504024347

Nº Threads      1ª Amostra de erro relativo     Média do erro relativo  Desvio padrão
6               0.069794215                     0.309846663             0.4312535941

Nº Threads      1ª Amostra de erro relativo     Média do erro relativo  Desvio padrão
8               1.5149212                       0.968064755             0.5558033121
```

```
_____________  CÁLCULO DE PI - MÉTODO DE MONTE CARLO_____________________

Para 10 000 000 pontos

Nº Threads      1ª Amostra de erro relativo     Média do erro relativo  Desvio padrão
2               0.041410521                     0.015828277             0.2033395302

Nº Threads      1ª Amostra de erro relativo     Média do erro relativo  Desvio padrão
4               0.016162185                     0.020768765             0.031964895

Nº Threads      1ª Amostra de erro relativo     Média do erro relativo  Desvio padrão
6               0.023139538                     0.016937906             0.0476514514

Nº Threads      1ª Amostra de erro relativo     Média do erro relativo  Desvio padrão
8               0.0050117895                    0.013112458             0.0707422396
```

Conclusão

Podemos afirmar que considerámos o trabalho bastante interessante. Além de conhecermos mais um conceito que poderá vir a ser útil (método de Monte Carlo), conseguimos consolidar conhecimentos que fomos adquirindo durante as aulas da dísciplina de “Sistemas Operativos”. Sobretudo, em relação às “threads” e à linguagem “C”.

Inicialmente existiam algumas dúvidas, sobre a forma como iríamos construir o trabalho, mas gradualmente, essas dificuldades foram sendo resolvidas.

Bibliografia

- PDF “SO-threads” - Professor Gonçalo Valadão
- Imagem da página da “Introdução”- site reseachgate.net