

**Licenciatura em Informática de Gestão**

**1º Ano – Pós-Laboral**

**Trabalho de Sistemas Operativos  
Maio 2018**

**Método MonteCarlo**

**Cálculo do Valor de Pi**

# **ÍNDICE**

[ÍNDICE 2](#_Toc515294534)

[INTRODUÇÃO 3](#_Toc515294535)

[PROBLEMAS E OBJECTIVOS A ATINGIR 3](#_Toc515294536)

[DESCRIÇÃO DO TRABALHO REALIZADO 4](#_Toc515294537)

[APRESENTAÇÃO DOS RESULTADOS 5](#_Toc515294538)

[CONCLUSÃO 6](#_Toc515294539)

[APÊNDICES 7](#_Toc515294540)

[CÓDIGO FONTE 7](#_Toc515294541)

# **INTRODUÇÃO**

## **PROBLEMAS E OBJECTIVOS A ATINGIR**

Com este trabalho ficamos que existem vários métodos para conseguirmos aproximar uma função através da utilização de escolhas aleatórias na realização de cálculos. Para isto aprendemos que a este tipo de motodologias dá-se o nome de simulação de Monte Carlo.

Com o objectivo de resolver o problema colocado neste trabalho tentamos inicialmente colocar em práctica os conhecimentos, conceitos e competências adquiridos no âmbito desta cadeira nas aulas teoricas e práticas.

Aquando da interpretação do enunciado, é imperativo identificar os parâmetros pelos quais este se rege, estabelecendo desde cedo tarefas e compreender o objectivo final da Simulação. Todos os parâmetros de funcionamento desta simulação são à partida definidos no enunciado fornecido pelo professor.

O principal objectivo a que este grupo se propôs foi o de conseguir finalizar a simulçação em código C, com todas as funções e correspondendo a todos os requisitos que constam do enunciado.

# **DESCRIÇÃO DO TRABALHO REALIZADO**

Optamos por tentar em conjunto definir uma metodologia de trabalho que assentava em, irmos conseguindo o entendimento do código que íamos gerando, e a definição de etapas.

1. A primeira etapa, teve com base um trabalho de pesquisa, no aprofundamento do que é a simulação de Monte Carlo. Para isso vimos vídeos (tal como este <https://www.youtube.com/watch?v=GNAeZqoLfYw>)
2. A segunda etapa partiu de um principio básico da programação que “A Roda já foi inventada”. Seguindo esta premissa, tentamos entender sobretudo ver as diferentes metodologias de implementação, mas sempre com o objectivo de criarmos o nosso próprio código e acima de tudo ultrapassando as dificuldades que nos iam aparecendo ao longo do desenvolvimento.
3. Na terceira etapa definimos como objectivo conseguir calcular o valor de PI através de threads e Ciclos estáticos. Nesta etapa as grandes dificuldades centrarm-se sobretudo em como organizar o código. Para isso definimos que o mesmo deveria estar divido em funções que resolveriam os diferentes problemas para conseguirmos ter um resultado final.
4. Na quarta etapa, a nossa missão foi a de criar ciclos e matrizes que podessem transformar os dados (ciclos e threads) que até aqui eram estáticos em valores com um maior dinamismo, visto esta multiplicidade de valores era um dos pontos essências no enunciado. Tendo toda a estrutura estática criada, o passo dado foi no sentido de colocar os ciclos ao serviço das varáveis que se queriam de um valor mutável.
5. Por fim, em termos de aplicação, e visto que era mais uma das solicitações do professor, integramos a contagem de tempo de cada ciclo.
6. A última etapa deste trabalho foi a elaboração deste documento.

# **APRESENTAÇÃO DOS RESULTADOS**

Os resultados apresentados serão sempre diferentes pois estamos perante uma simulação de um valor.

Os que decidimos colocar neste trabalho estão expressos na tabela abaixo:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **2 Threads** | | | **4 Threads** | | | **6 Threads** | | | **8 Threads** | | |
|  | **Pi** | **Dif. vs Real** | **Tempo** | **Pi** | **Dif. vs Real** | **Tempo** | **Pi** | **Dif. vs Real** | **Tempo** | **Pi** | **Dif. vs Real** | **Tempo** |
| **500** | 2,564539 | -0,577053 | 0,000261 | 3,139394 | -0,002198 | 0,951463 | 3,142572 | 0,000980 | 2,596882 | 3,197790 | 0,056198 | 5,070626 |
| **20000** | 3,122892 | -0,018700 | 0,002334 | 3,139396 | -0,002196 | 0,954549 | 3,142575 | 0,000983 | 2,601495 | 3,197277 | 0,055680 | 5,076446 |
| **100000** | 3,129870 | -0,011722 | 0,012432 | 3,139413 | -0,002179 | 0,969136 | 3,143245 | 0,001653 | 2,623778 | 3,197136 | 0,055544 | 5,104627 |
| **1000000** | 3,139369 | -0,002223 | 0,951261 | 3,139338 | -0,002254 | 1,115688 | 3,149791 | 0,008199 | 2,845430 | 3,199026 | 0,057434 | 5,387097 |
| **10000000** | 3,139394 | -0,002198 | 0,951463 | 3,142572 | 0,000980 | 2,596582 | 3,198331 | 0,056739 | 5,070281 | 3,230147 | 0,088555 | 8,236724 |

# **CONCLUSÃO**

Em conclusão, pensamos que este trabalho foi de resolução muito complexa para os nossos conhecimentos e limitações, mas nem por isso foi impeditivo de atingirmos os objectivos a que nos propusemos no início do semestre.

O nosso espectro de conhecimento alargou-se significativamente não só ao nível da programação, mas também ao nível de algoritmos e metodologias de simulações e previsões que actualmente são utilizadas em inúmeras áreas da nossa vida, não só a nível matemático, como a nível financeiro entre outros.

# **APÊNDICES**

## **CÓDIGO FONTE**

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#include <time.h>

#define BILLION 1E9 //Definição de número de ciclos por Segundo no CPU

#define NUMTHR\_MAX 8 // Numero máximo de threads

#define PI 3.141592 //valor de pi até a 15a casa depois da virgula, apenas para fins de comparacao com o valor obtido no fim

//agrega as variáveis a serem utilizadas para threads e para ciclos

typedef struct t\_nfo {

int tid;

int ciclos;

};

int num\_ptos [NUMTHR\_MAX]; // Total de pontos da thread

int num\_ptos\_cir[NUMTHR\_MAX]; // Pontos no circulo

void \*calcula (void \*param); // sorteia e verifica os pontos pela thread

int main (int argc, char \*argv[]) {

struct timespec requestStart, requestEnd;

clock\_gettime(CLOCK\_REALTIME, &requestStart);

int j;

int i;

double pi = 0; // contem o valor de pi

int totalp = 0; // total de pontos

int totalc = 0; // pontos no circulo

double valor\_pi=0,erro=0;

pthread\_t tid[NUMTHR\_MAX]; // ID das threads

// MATRIZ PARA AS THREADS

int threads[] = {2, 4, 6, 8};

int treads\_size = (int)sizeof(threads)/sizeof(threads[0]);

// MATRIZ PARA OS CICLOS

int ciclos[] = {500, 20000, 100000, 1000000, 10000000};

int ciclos\_size = (int)sizeof(ciclos)/sizeof(ciclos[0]);

for (int t = 0; t < treads\_size; t++) {

for (int c = 0; c < ciclos\_size; c++) {

printf("\n== %i TREADS ==|== %i CICLOS ==\n", threads[t], ciclos[c]);

// Para todas as threads

for (i = 0; i < threads[t]; i++) {

struct t\_nfo nfo;

nfo.tid = i;

nfo.ciclos = ciclos[c];

// cria a i-esima thread

pthread\_create(&tid[i], NULL, calcula, &nfo);

}

// Para cada thread

for (i = 0; i < threads[t]; i++) {

// espera que as threads terminem

pthread\_join(tid[i], NULL);

// Imprime na tela a qtde de pontos no círculo

// e no total de cada thread

printf("\nPontos no Circulo %d : %d ", i + 1, num\_ptos\_cir[i] + 1);

printf("\nTotal de Pontos %d : %d ", i + 1, num\_ptos[i] + 1 );

}

for (i = 0; i < threads[t]; i++) {

totalp += num\_ptos[i]; // totalp = totalp + num\_ptos[i]

totalc += num\_ptos\_cir[i]; // totalc = totalp + num\_ptos[i]

}

// Calcula o valor de pi e imprime na tela

valor\_pi = 4.0 \* (((double) totalc) / ((double) totalp));

// transforma totalp

// e totalc em double

//definição da qualidade de cada estimatíva

erro = valor\_pi - PI;

printf("\n\nValor de pi:%f \n", valor\_pi);

printf("Diferença em Relação ao PI ideal : %.6f \n\n",erro);

clock\_gettime(CLOCK\_REALTIME, &requestEnd);

// Cálculo do tempo dispendido

double acum = ((requestEnd.tv\_sec - requestStart.tv\_sec) + (requestEnd.tv\_nsec - requestStart.tv\_nsec) / BILLION);

printf( "Tempo Acumulado: %f\n", acum );

}

}

return 0;

}

void \*calcula (void \*param) {

int i;

struct t\_nfo nfo = \*((struct t\_nfo \*)param);

double x,y,quad;

num\_ptos[nfo.tid] = 0;

num\_ptos\_cir[nfo.tid] = 0;

for (i = 0; i<nfo.ciclos; i++) {

x = drand48(); // sorteia um número de 0 a 1

y = drand48(); // sorteia um número de 0 a 1

quad = ((x\*x) + (y\*y));

// Se a soma dos quadrados for menor que R = 1

// então caiu no círculo

if (quad <= 1)

num\_ptos\_cir[nfo.tid] ++; // conta pontos no círculo

num\_ptos[nfo.tid] ++; // incrementa os pontos totais da thread N (0 a 9)

}

pthread\_exit(0);

}