Arquitetura Avançada para Computação

Daniel Ribeiro dos Santos – 20170157528

Laboratório 02

O presente laboratório visou a implementação de um código em C para calcular o valor de pi. Utiliza-se o método de Monte Carlo para a obtenção do valor, e, para isso, aproxima-se a circunferência por N retângulos.

Temos como objetivo principal do laboratório a implementação de um código utilizando múltiplas threads.

O código é, simplificadamente, o desenvolvimento da seguinte integral:

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

Dado o sistema operacional do estudante (Windows), escolheu-se implentar o código por meio do Windows Subsystem for Linux, utilizando Visual Studio Code como IDE.

A máquina utilizada possui 4 núcleos e 8 processadores lógicos.

Código Serial

Para o código seria, não utilizamos a biblioteca pthreads.h. O código está mostrado nos anexos. Observemos que, maior a quantidade de retângulos utilizados (N), maior a precisão do código. Para $N = 10^8$, temos $\pi = 3.141593$, requerindo 2.078125 segundos para concluir a operação.

Código em Paralelo

Sua implementação teve a mesma lógica do caso serial, mas adaptando-o para múltiplas threads.

Sabendo que, para cada thread, foi-se utilizado a função mutex, o qual bloqueia o uso da cpu, observa-se o melhor desempenho quando o número de threads utilizadas for igual ao número de processadores lógicos da máquina. Caso utilizemos mais threads, o tempo de execução diminui proporcionalmente.

Para $N = 10^8$, temos $\pi = 3.141593$. Contudo, a função utilizada anteriormente para computar o tempo de execução não pode ser utilizada no caso multithread. Ainda, por observação, nota-se que o tempo de execução é menor que 1s.

As tabelas abaixo mostram os tempos necessários para executar os códigos com N retângulos.

Serial	
N	t
100	0.023
1000	0.028
10000	0.039
100000	0.044
1000000	0.048
10000000	0.231
100000000	1.928
10^9	24.373

Parallel (4 Threads)	
N	t
100	0.041
1000	0.045
10000	0.029
100000	0.027
1000000	0.040
10000000	0.103
100000000	0.783
10^9	6.959

Parallel (2 Threads)	
N	t
100	0.028
1000	0.032
10000	0.026
100000	0.060
1000000	0.040
10000000	0.159
100000000	0.1.079
10^9	11.055

Parallel (8 threads)	
N	t
100	0.033
1000	0.034
10000	0.032
100000	0.033
1000000	0.032
10000000	0.078
100000000	0.488
10^9	4.472

Anexos

Código Serial

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>


int main(){
clock_t t;
t = clock();

float N = 1000000000;
double dx = 1/N;
double sum = 0;

for (int i = 0; i <N ;i++){
    sum = sum + (4/(1+pow(i*dx,2)))*dx;
}

t = clock() - t;
double tempo = ((double)t)/CLOCKS_PER_SEC;

printf("pi = %f\n Tempo de Execucao: %f \n",sum,tempo);
}
```

Códiigo Paralelo

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <pthread.h>
#include <time.h>
#define N 100000000
#define num_thread 8

const float N1 = N;
double dx = 1/N1;
double pi = 0;
pthread_t thread[num_thread];
pthread_mutex_t gLock;


void *compute(void *arg){
    int num = *((int*)arg);
    double partial_sum = 0;

    for (int i = num;i<N;i+=num_thread){
    partial_sum += (4/(1+pow((i)*dx,2)))*dx;
    }
    pthread_mutex_lock(&gLock);
    pi += partial_sum;
    pthread_mutex_unlock(&gLock);

    return 0;
}


int main(){

clock_t t = clock();
int i,tNum[num_thread];
pthread_mutex_init(&gLock, NULL);

for ( i = 0; i < num_thread; i++ ) {
tNum[i] = i;
pthread_create(&thread[i],NULL,compute,(void*)&tNum[i]);
}

for (int i = 0; i< num_thread; i++){
    pthread_join(thread[i],NULL);
}

    printf("\npi = %lf \n", pi);
    t = clock()-t;
    double tempo = ((double)t)/CLOCKS_PER_SEC;
    printf("Tempo de execucao: %f\n",tempo);
    pthread_mutex_destroy(&gLock);
    return 0;
}
```