**Actividad de programación: Calculando Pi con threads en Java**

Esteban Pérez Herrera (A01204739)

Tecnológico de Monterrey, Campus Querétaro

*A01204739@itesm.mx*

11 de septiembre de 2018

**Resumen**

Durante la realización de esta práctica se realizó el mismo proceso, en este caso calcular el valor de PI, tanto de manera secuencial como paralela, donde la última fue capaz de lograr un mejor desempeño.

**1. Introducción**

Hoy en dai el equipo de computo cuenta con la capacidad de ejecutar múltiples tareas a de manera paralela gracias a que posee varios procesadores que fisicamente se encargan de ello, esto con el fin de acelerar la tarea previamente ejecutada de manera secuencial. La tarea seleccionada fue el calculo de PI al aproximar el valor de la integral a través de la aproximación por rectangulos al sumar las alturas de los rectangulos y mutiplicarlos por el ancho.

**2. Implementación**

* Equipo utilizada.
  + CPU: I7 4710HQ, 4C,8TH
  + Memoria: 16GB RAM
  + OS: Ubutu 16.04 LTS

Primero se encuentra el código usado para el secuencial:

Dentro de la clase PI\_non\_thread se declaran sus variables privadas, NUM\_RECTS que recibe y PI\_Value que entrega, además de su constructor que inicializa estas variables cuando es creado un objeto de esta clase, además de la función encargada de devolver el valor de PI una vez calculada:

public class PI\_non\_thread{

private int NUM\_RECTS;

private double PI\_value;

public PI\_non\_thread(int NUM\_RECTS){

this.NUM\_RECTS = NUM\_RECTS;

this.PI\_value = 0;

}

public double getPI\_value(){

return PI\_value;

}

Posteriomente para el calculo de pi, se uso el metodo mencionado previamente donde se aproximo el valor de la integral por medio del área de rectangulos:

public void calculatePI(){

double mid, height, sum=0, i;

double width=1.0 / (double)NUM\_RECTS;

for (i = 0; i < NUM\_RECTS; i++) {

mid = (i + 0.5) \* width;

height = 4.0 / (1.0 + (mid \* mid));

sum += height;

}

PI\_value = width \* sum;

}

}

Dentro del main de esta implementación primero se creó la constate de NUM\_RECTS que define la cantidad de rectangulos usados para la aproximación, además de las variables necesarias para tomar el tiempo de ejecución y se declaró un objeto de la clase PI\_non\_thread.

public class PI\_non\_thread\_Main{

private static int NUM\_RECTS = 100\_000\_000;

public static void main(String[] args) {

long startTime, stopTime;

double acum = 0;

PI\_non\_thread x = new PI\_non\_thread(NUM\_RECTS);

En este ciclo simplelemente se ejecuta el calculo de PI 10 veces seguidas tomando el tiempo de cada una, para después mostrar el valor de PI y mostrar el tiempo promedio que tomó su ejecución.

for (int i = 0; i < 10; i++) {

startTime = System.currentTimeMillis();

x.calculatePI();

stopTime = System.currentTimeMillis();

acum += (stopTime - startTime);

}

System.out.printf("PI = %.15f\n", x.getPI\_value());

System.out.printf("avg time = %.5f ms\n", (acum / 10));

}

}

Luego en la implementación paralela se empezo por la creación de la clase, en este caso definida para la implementación de threads, aunque no por ello más complicada que la usada previamente, empezando nuevamente por sus variables, sólo que en este caso también hay que considerar que recibira un start y un end, que delimitan la rebanada de trabajo en el calculo de pi que le tocará, además de su función getResult() para devoler el dato.

public class PI\_thread extends Thread {

private int start, end;

private int NUM\_RECTS;

private double result;

public PI\_thread(int start, int end, int NUM\_RECTS){

super();

this.start=start;

this.end=end;

this.NUM\_RECTS=NUM\_RECTS;

this.result=0;

}

public double getResult(){

return result;

}

El for se ejecuta de manera muy similar en la implementación secuencial, pero aquí en lugar de calcular toda la integral, solamente calcula una parte suya delimitada por start y end

public void run(){

result=0;

double mid, height, sum=0, i;

double width=1.0 / (double)NUM\_RECTS;

for (i = start; i < end; i++) {

mid = (i + 0.5) \* width;

height = 4.0 / (1.0 + (mid \* mid));

sum += height;

}

result = width \* sum;

}

}

Ya creada la clase se procedio al main del programa, al igual que el secuencial se encuentra la variable NUM\_RECTS que define la cantida de rectangulos a usar para aproximar el valor de PI, pero también aparece la variable MAXTHREADS que almacena la cantidad de threads presentes en el equipo, posteriormente esta el main con las variables necesarias para tomar el tiempo y la implementación de la clase PI\_thread para ser usada en el procesamiento paralelo, además de variables como que define el tamaño del trabajo por cada thread.

public class PI\_thread\_Main{

private static int NUM\_RECTS = 100\_000\_000;

private static final int MAXTHREADS = Runtime.getRuntime().availableProcessors();

public static void main(String args []){

PI\_thread threads[];

int block;

double startTime, stopTime, total = 0;

double acum = 0;

block = NUM\_RECTS/MAXTHREADS;

threads = new PI\_thread[MAXTHREADS];

La parte siguiente define el uso de procesamiento paralelo, de incio se encuentra el for que lleva a cabo dicho procesamiento 10 veces para tener un valor más acertado del tiempo de ejecución para inmediatamente entrar de lleno a los thtreads, dado que el primer for dentro se encarga de crear los threads y asignarles su carga de trabajo.

for (int j=1; j<=10 ; j++ ) {

for (int i=0; i<threads.length; i++) {

if (i != threads.length - 1) {

threads[i] = new PI\_thread((i \* block), ((i + 1) \* block), NUM\_RECTS);

} else {

threads[i] = new PI\_thread((i \* block), NUM\_RECTS, NUM\_RECTS);

}

}

Ya creados los threads ya es posible empezar el procesamiento y tomar el tiempo de los mismos.

startTime = System.currentTimeMillis();

for (int i = 0; i < threads.length; i++) {

threads[i].start();

}

Ya que los threads esta corriendo se procede a esperar que los mismos se terminen de ejecutar, ya que terminan se toma el tiempo de ejecución total.

for (int i = 0; i < threads.length; i++) {

try {

threads[i].join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

stopTime = System.currentTimeMillis();

acum += (stopTime - startTime);

Terminados las 10 corridas del calculo de PI se suman todos los resultados de los threads para obtener el valor de pi, dicho valor se imprime junto con el tiempo que tomo el programa en promedio.

if (j == 10) {

total = 0;

for (int i = 0; i < threads.length; i++) {

total += threads[i].getResult();

}

}

}

System.out.printf("sum = %.15f\n", total);

System.out.printf("avg time = %.5f ms\n", (acum / 10));

}

}

**3. Resultados**

Las pruebas de los programas fueron realizadas tanto de manera local como en el servidor, ambos demostraron una mejora en la implemetación paralela sobre la secuencial, aunque claramente la diferencia es relativa al hardware en la que se implemente.



**3. Referencias**

Pérez, P. O. (n.d.). Computing Pi.