INFORME TAREA COMPUACION

IMPLEMENTACION DE ESTIMACION DE PI USANDO EL METODO DE MONTECARLO

DAMY VILLEGAS

A00398942

Introducción

El método de Monte Carlo es una técnica estadística, la cual es utilizada para realizar simulaciones o estimaciones por medio del uso de números aleatorios. Este método es particularmente útil con respecto a situaciones donde es casi imposible obtener resultados exactos a través de métodos analíticos. En este informe, se detallará claramente el método de Monte Carlo, al igual que como se aplica en un modelo que es Cliente-Maestro-Trabajadores, y la estrategia de distribución utilizada para optimizar la carga de trabajo.

1. **Descripción del Método de Monte Carlo**

Este Método es una técnica de simulación la cual utiliza valores aleatorios para solucionar problemas que son difíciles de tratar mediante técnicas determinísticas. En esta tarea se utilizó el método de Monte Carlo para poder estimar el valor de Pi. La idea consiste en producir puntos aleatorios dentro de un cuadrado que contiene un círculo para asi determinar la proporción de puntos que caen dentro del círculo, utilizando la siguiente relación:

$$\boldsymbol{\pi} \approx \mathbf{4} \times \frac{\text{Puntos dentro del Circulo}}{\text{Puntos Tottales}}$$

2. Diseño del Sistema Cliente-Maestro-Trabajador

El sistema sigue una arquitectura Cliente-Maestro-Trabajador, la cual es útil para escenarios donde el proceso de datos puede dividirse en varios “trabajadores” que estén realizando la misma tarea, pero de manera independiente.

Cliente

En esta clase se puede ver la inicialización del sistema, puesto que configura una lista de trabajadores y un maestro que los coordina, para luego solicitar al maestro que realice el cálculo de Pi y asi mostrar el resultado.

Maestro

Esta clase coordina a los trabajadores, dividiendo la función de generar puntos y recopilar resultados de cada trabajador. Además, utiliza un ExecutorService para manejar los hijos que

tiene cada trabajador, asegurando que cada uno pueda ejecutar su tarea y que sea de forma paralela, en si esta clase hace lo siguiente:

- Divide el numero total de puntos en los trabajadores
- Ejecuta cada trabajador en un hilo separado
- Recopila los resultados para calcular la estimación de Pi

Trabajador

Esta clase es responsable de generar un subconjunto de puntos aleatorios y contar cuantos de estos puntos caen dentro del circulo de radio 1. Esto permite que el calculo se distribuya en paralelo, acelerando el procesamiento en comparación con una implementación secuencial.

3. Distribución de tareas y Nodos

La tarea principal que es generar N puntos aleatorios y contar cuántos caen dentro del círculo, se divide en varias sub-tareas de la siguiente manera:

- El **cliente** envía una solicitud al maestro para iniciar el cálculo.
- El **maestro** divide el número total de puntos N entre el número de trabajadores y asigna una cantidad de puntos a cada trabajador.
- Cada **trabajador** genera su subconjunto de puntos aleatorios de forma paralela, gracias al uso de ExecutorService.
- Al finalizar, el maestro suma los resultados de todos los trabajadores y calcula la estimación final de π\piπ.

Aquí se aprovecha la **paralelización** para realizar la simulación de Monte Carlo de la forma más eficiente posible y asi permitir que cada trabajador procese su subconjunto de puntos independientemente de los demás.

4. Resultados de las Pruebas y Análisis del Rendimiento

Para analizar el rendimiento del sistema distribuido se realizaron pruebas con 4 diferentes valores de N . En la siguiente tabla se presentan alguno de sus resultados, incluyendo la estimación de Pi y el tiempo de ejecución:

Número de Puntos (N)	Estimación de Pi	Tiempo de Ejecución (ms)
1000	3,18000	86 ms
10000	3,12840	5 ms
100000	3,14584	8 ms
1000000	3,14079	58 ms

Análisis del Rendimiento

Precisión:

Según los resultados obtenidos la precisión de la estimación de Pi mejora a medida que se aumenta el valor de N. Esto se debe a que un mayor numero de puntos permite una mejor

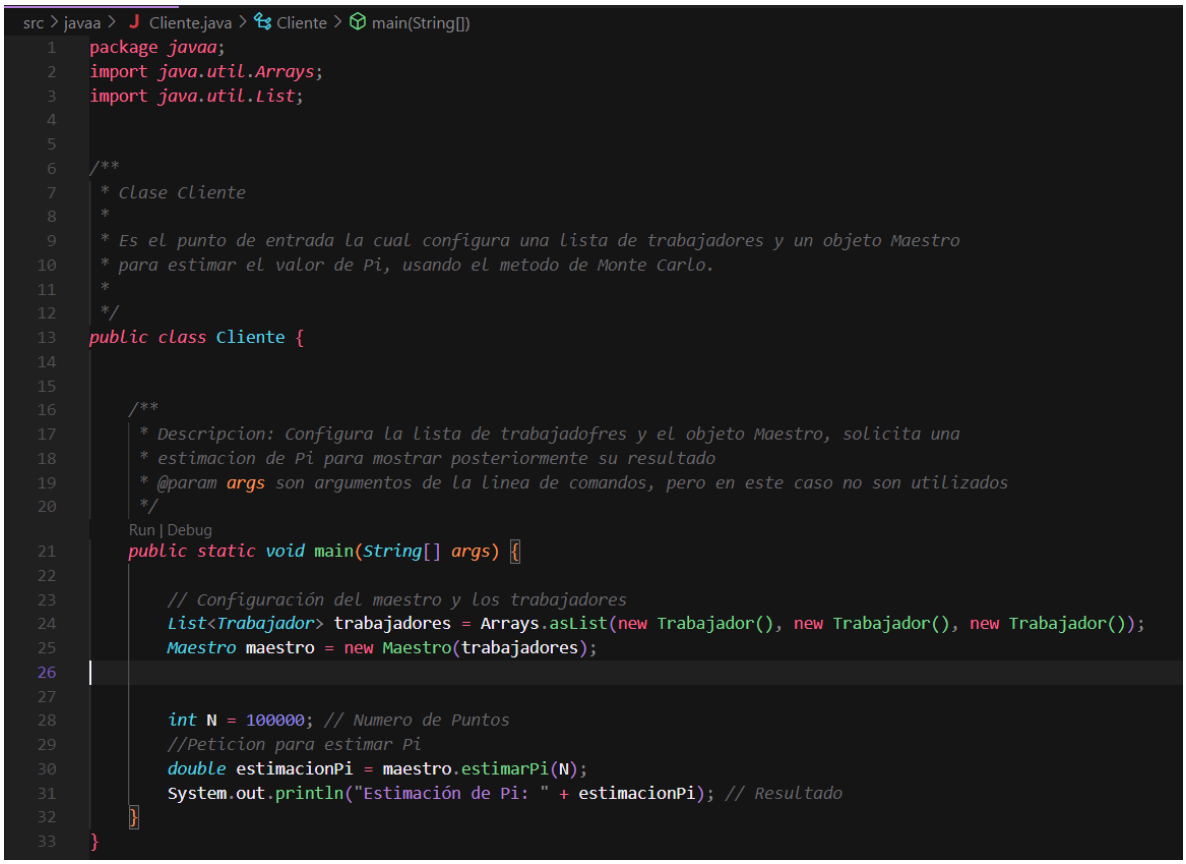
representación estadística del área del círculo dentro del cuadrado. Como se puede ver en las pruebas realizadas, la estimación de pi para N = 1,000,000 fue de 3.14079, el cual es un valor cercano al valor real de Pi, aproximadamente de 3.14159.

Tiempo de Ejecución

El tiempo de ejecución varia significativamente con los distintos valores de N. La implementación distribuida que utiliza múltiples trabajadores para procesar los puntos de manera paralela demuestra ser más eficiente a que si lo hacemos con una ejecución secuencial.

5. Código Fuente de la implementación

Cliente



Maestro

```java
package javaa;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.*;

/**
 * Clase Maestro
 *
 * Cordina una lista de trabajadores para realizar una estimacion
 * del valor de pi, usando la implementacion del metodo de montecarlo.
 *
 */
public class Maestro {
    private List<Trabajador> trabajadores;


    /**
     * Descripcion: Constructor de la clase Maestro
     * @param trabajadores lista de objetos ded trabajador que realizaran la generacion de puntos.
     */
    public Maestro(List<Trabajador> trabajadores) {
        this.trabajadores = trabajadores;
    }
```

```java
public class Maestro {
    }

    /**
     * Descripcion: Estima el valor de pi dividiendo la tarea para asi generar
     * numeros aleatorios.
     *
     * @param N numero de puntos que se generan para la estimacion
     * @return la estimacion del valor de Pi
     */
    public double estimarPi(int N) {
        int m = trabajadores.size();
        int puntosTrabajador = N / m;
        ExecutorService executor = Executors.newFixedThreadPool(m);
        List<Future<Integer>> resultados = new ArrayList<>();

        // Distribuye el trabajo (Trabajadores)
        for (Trabajador trabajador : trabajadores) {
            Callable<Integer> tarea = () -> trabajador.generarPuntos(puntosTrabajador);
            resultados.add(executor.submit(tarea));
        }

        // Resultados recolectados
        int puntosDentroCirculo = 0;
        for (Future<Integer> resultado : resultados) {
            try {
                puntosDentroCirculo += resultado.get();
            } catch (InterruptedException | ExecutionException e) {
                e.printStackTrace();
            }
        }

        // Se estima Pi
        executor.shutdown();
        return 4.0 * puntosDentroCirculo / N;
    }
}
```

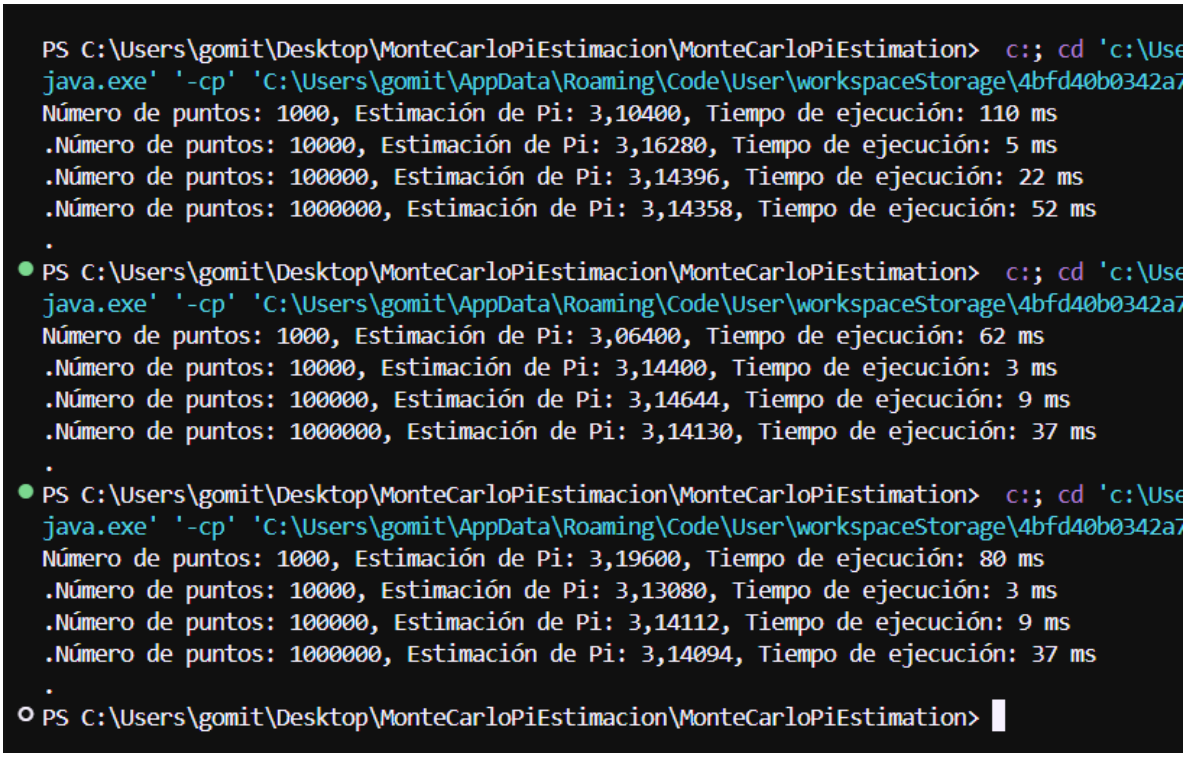
Trabajador

```java
package javaa;
import java.util.Random;

/**
 * Clase Trabajador
 *
 * Esta clase se encarga de generar puntos aleatorios dentro de un cuadrado de lado 2
 * centrado en el origen, contanto cuantos puntos de ellos caen dentro de un circulo de
 * raido 1 centrado en el origen
 */
public class Trabajador {

    //Genera numero aleatorios
    private Random random = new Random();

    /**
     * Desripcion: Genera N puntos en el espacio y cuenta cuantos de esos
     * caen dentro del circulo (radio 1
     *
     * @param N numero de puntos que se generaran
     * @return numero de puntos que caen dentro del circulo de radio 1
     */
    public int generarPuntos(int N) {
        int dentroCirculo = 0;
        for (int i = 0; i < N; i++) {
            double x = random.nextDouble() * 2 - 1; // Generar punto en [-1, 1]
            double y = random.nextDouble() * 2 - 1;
            if (x * x + y * y <= 1) {
                dentroCirculo++;
            }
        }
        return dentroCirculo;
    }
}
```

Pruebas

```java
package tests;
import java.util.Arrays;
import java.util.List;
import javaa.Trabajador;
import javaa.Maestro;
Ejecución y depuración (Ctrl+Mayús+D)

public class Pruebas {

    Run | Debug
    public static void main(String[] args) {
        // Configuración del maestro y los trabajadores
        List<Trabajador> trabajadores = Arrays.asList(new Trabajador(), new Trabajador(), new Trabajador());
        Maestro maestro = new Maestro(trabajadores);

        // Diferentes valores de N para realizar las pruebas
        int[] puntos = { 1_000, 10_000, 100_000, 1_000_000};

        // Realizar pruebas
        for (int N : puntos) {
            // Medir tiempo de ejecución
            long inicioT = System.currentTimeMillis();
            double estimacionPi = maestro.estimarPi(N);
            long finT = System.currentTimeMillis();
            long duracion = finT - inicioT;

            // Imprimir resultados
            System.out.printf(format:"Número de puntos: %d, Estimación de Pi: %.5f, Tiempo de ejecución: %d ms%n.", N, estimacionPi, duracion);
        }
    }
}
```

Algunos Resultados de las Pruebas



Conclusiones

El método de Monte Carlo además de ser eficiente permite realizar estimaciones precisas de Pi, en tiempos muy razonables, a medida que se incrementa el numero de puntos generados. La precisión y reducción en el tiempo de ejecución dejan ver que esta técnica es adecuada para diferentes aplicaciones que requieran de cálculos numéricos intensivos.