# Obtención de PI con el método de Montecarlo y ordenamiento de números con el algoritmo QuickSort usando hilos

Estudiantes:

Jared Miguel Hidalgo Esquivel

Konrad Benjamín Trejo Chávez

Reewos Erwin Talla Chumpitaz

Universidad Nacional de Ingeniería, Facultad de Ciencias

Curso:

CC462 A Sistemas Concurrentes y Distribuidos
Laboratorio 01

## Contenido



---

# 1 Introducción

El método consiste en el ordenamiento de las cifras del número pi, previamente hallado con el método de Montecarlo. Dicho ordenamiento va a realizarse con el algoritmo de quicksort. Todo el proceso será paralelizado en el lenguaje de java.

# 2 Marco Teórico

## 2.1 Método de Montecarlo

El método de Montecarlo es un método no determinista o estadístico numérico, usado para aproximar expresiones matemáticas complejas y costosas de evaluar con exactitud. El método se llamó así en referencia al Casino de Montecarlo (Mónaco) por ser "la capital del juego de azar", al ser la ruleta un generador simple de números aleatorios. El nombre y el desarrollo sistemático de los métodos de Montecarlo datan aproximadamente de 1944 y se mejoraron enormemente con el desarrollo de la computadora.

### 2.1.1 Cálculo de $\pi$ por Montecarlo

Consideremos al círculo unitario inscrito en el cuadrado de lado 2 centrado en el origen. Dado que el cociente de sus áreas es $\pi$, el valor de $\pi$ puede aproximarse usando Montecarlo de acuerdo al siguiente método:

- Dibuja un círculo unitario, y al cuadrado de lado 2 que lo inscribe.
- Lanza un número $n$ de puntos aleatorios uniformes dentro del cuadrado.
- Cuenta el número de puntos dentro del círculo, por ejemplo, puntos cuya distancia al origen es menor que 1.
- El cociente de los puntos dentro del círculo dividido entre $n$ es un estimado de, $\pi/4$. Multiplica el resultado por 4 para estimar $\pi$.

En este cálculo se tienen que hacer dos consideraciones importantes:

- Si los puntos no están uniformemente distribuidos, el método es inválido.
- La aproximación será pobre si solo se lanzan unos pocos puntos. En promedio, la aproximación mejora conforme se aumenta el número de puntos.

## 2.2 Algoritmo QuickSort

El ordenamiento rápido (quicksort en inglés) es un algoritmo de ordenacion creado por el científico británico en computación C. A. R. Hoare. El algoritmo trabaja de la siguiente forma:

- Elegir un elemento del conjunto de elementos a ordenar, al que llamaremos pivote.
- Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
- La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
- Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.
- Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.
- En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos sublistas de igual tamaño. En este caso, el orden de complejidad del algoritmo es $\Omega(nlogn)$.

- En el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de $O(n^2)$. El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas. Pero principalmente depende del pivote, si por ejemplo el algoritmo implementado toma como pivote siempre el primer elemento del array, y el array que le pasamos está ordenado, siempre va a generar a su izquierda un array vacío, lo que es ineficiente.
- En el caso promedio, el orden es $O(nlogn)$.

# 3 Metodología

Para el cálculo del número pi, se utilizó el método de Montecarlo. Este método consiste en determinar dicho número de forma estadística, generando puntos aleatorios dentro de un cuadrado de longitud 2. Luego, según la aproximación estadística, si los puntos son variables independientes entre sí, debería cumplirse que el número de puntos dentro del cuadrado sea proporcional al número de puntos generados en él, y de la misma manera, el número de puntos que estén dentro del círculo inscrito son proporcionales en la misma medida al área del mismo. Así, la división número de puntos dentro del círculo entre número de puntos totales nos brinda una aproximación de la razón entre las áreas, es decir, $\pi/4$. Entre más puntos sean generados, más exactitud tendrá el método, por lo que se realizará una paralelización entre 4 hilos para aumentar la precisión sin comprometer la eficiencia.

Para el ordenamiento de los números, se hizo uso del método de quicksort. Este método consiste en la partición recurrente del array contenedor de los números a ordenar. Se parte el array a la mitad y se fija un pivote equivalente al número central. Luego, se procede a enviar los números menores al pivote a la izquierda, mientras que los mayores son llevados a la derecha. Posteriormente, a cada mitad se le aplica quicksort de forma independiente de la otra, por lo que es posible paralelizarlo.

# 4 Resultados y Discusiones

Se utilizó todos los hilos disponibles para hallar el valor de Pi usando el método de Montecarlo, para la realización de esta prueba. se uso 4 hilos.

| hilo principal | Numero de Puntos | Tiempo de ejecución (ms) |
|---|---|---|
| 1 | 1000 | 0.03152 |
| 1 | 10000 | 0.03786 |
| 1 | 100000 | 0.06545 |
| 1 | 1000000 | 0.13591 |

| cantidad de hilos | Numero de Puntos | Tiempo de ejecución (ms) |
|---|---|---|
| 2 | 1000 | 0.0275604 |
| 2 | 10000 | 0.0327037 |
| 2 | 100000 | 0.0612136 |
| 2 | 1000000 | 0.2866 |
| 3 | 1000 | 0.0236724 |
| 3 | 10000 | 0.0237773 |
| 3 | 100000 | 0.0560241 |
| 3 | 1000000 | 0.2424781 |
| 4 | 1000 | 0.0179534 |
| 4 | 10000 | 0.0351654 |
| 4 | 100000 | 0.0469242 |
| 4 | 1000000 | 0.2295187 |

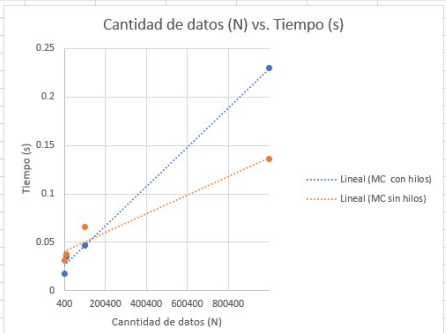


Figure 1: Resultados de tiempo (Montecarlo)

Se utilizó todos los hilos disponibles por el ordenador para la realización de esta prueba (4 hilos)

| QS sin hilos | nanosegundos | N | segundos |
|---|---|---|---|
| | 62133 | 100 | 0.000062133 |
| | 502633 | 1000 | 0.000502633 |
| | 1373266 | 5000 | 0.001373266 |
| | 2228666 | 10000 | 0.002228666 |
| | 9846133 | 50000 | 0.009846133 |
| | 14624966 | 100000 | 0.014624966 |

| QS con hilos | nanosegundos | N | segundos |
|---|---|---|---|
| | 216666 | 100 | 0.000216666 |
| | 326266 | 1000 | 0.000326266 |
| | 464033 | 5000 | 0.000464033 |
| | 674066 | 10000 | 0.000674066 |
| | 1472800 | 50000 | 0.0014728 |
| | 2580600 | 100000 | 0.0025806 |

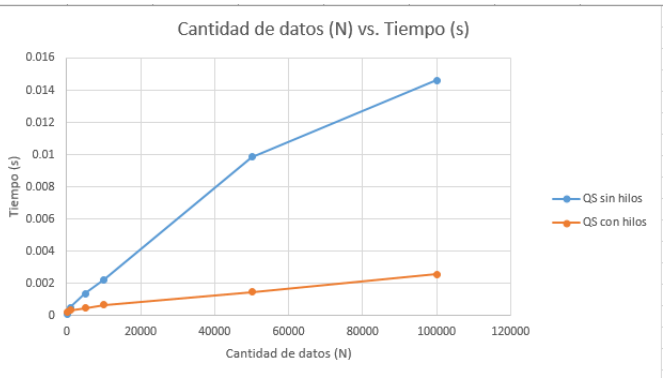


Figure 2: Resultados de tiempo (QuickSort)

# 5 Conclusiones

Vemos que el método de Montecarlo es bastante sencillo de realizar (y por ende rápido), sin embargo, no es muy exacto sino hasta tener bastantes puntos (se consiguió una precisión de 4 cifras exactas con 1000000 de puntos). Aún así, vemos que su rendimiento fue mejorado con la implementación de hilos. Se pudo ver una gran mejora en el rendimiento (tiempo) del algoritmo QuickSort con el uso de hilos, el cual podría mejorar si consideramos su variación de Random-QuickSort. Se pudo ver una mejora en el rendimiento (tiempo) del algoritmo usando el método de montecarlo con el uso de hilos, aunque para un estudio mejor se podría considerar trabajar con más hilos y datos.

# 6 Anexo Código

- **Repositorio:** `https://github.com/reewos/LAB-01-Calculo-de-PI-y-ordenamiento-en-paralelo`
- **Montecarlo:** `https://github.com/reewos/LAB-01-Calculo-de-PI-y-ordenamiento-en-paralelo/tree/master/MontecarloHilos/src/montecarlohilos`
- **Quicksort:** `https://github.com/reewos/LAB-01-Calculo-de-PI-y-ordenamiento-en-paralelo/tree/master/ordenamiento_paralelo/src/ordenamiento_paralelo`