Cálculo de Pi y Merge-sort usando hilos

Alumnos::
Ayrton Coronado - acoronadoh@uni.pe
Flores Huamaní - lfloresh@uni.pe
Barrientos Porras - herlees.barrientos.p@uni.pe
Universidad Nacional de Ingeniería, Facultad de Ciencias,

Curso:
CC462 Sistemas concurrentes y distribuidos
Laboratorio 1

Resumen

El presente trabajo usa threads sincronizados en java para mejorar la velocidad al obtener resultados en los cálculos de algunos algoritmos. Usamos el cálculo de pi por integrales intentando coincidir con la mayor cantidad de decimales gracias al uso de BigDecimal de java y el ordenamiento por Merge-sort que usa un máximo de cuatro threads y distribuye la recursión en un runnable que pertenece a cada thread. Comparamos los resultados de los cálculos entre el método secuencial, para dos hilos y cuatro hilos. Obteniendo una mejora del 65% paralelizando, para el Merge-sort obtenemos una mejora de alrededor de 40% en la velocidad de los cálculos, lo cual confirma la utilidad del uso de hilos en estos algoritmos.

Palabras clave: Pi, merge sort, threads, java.

Contenidos

1 Introducción

En muchas ocasiones uno se puede preguntar si realmente vale la pena aprender la computación concurrente, quisiera realmente responder que no a esta pregunta ya que al principio es complicado entender su funcionamiento, pero esta época donde los procesadores poseen varios threads es casi una obligación poder aprovechar esta ventaja. Vamos a mostrar como dos algoritmos altamente paralelizables y muy conocidos pueden aprovechar los threads del procesador para incrementar su velocidad de cálculo. Usamos java ya que su manejo de threads está muy desarrollado y posee una clase importante para su manejo la cual es Threads y la interfaz Runnable.
Desarrollamos la implementación del cálculo de Pi mediante el algoritmo de Integrales de Riemann y Ordenamiento por el algoritmo MergeSort.Los cuales compararemos tanto en secuencial como en paralelo.

2 Marco Teórico

2.1 Creando Threads y sincronizándolos en java

Hay dos formas de crear un nuevo hilo de ejecución. Una es declarar una clase como una subclase de Thread. Esta subclase debería anular el método run de la clase Thread. Entonces se puede asignar e iniciar una instancia de la subclase. Por ejemplo, un hilo que calcula números primos mayores que un valor establecido podría escribirse de la siguiente manera:

```
class PrimeThread extends Thread {

    long minPrime;

    PrimeThread(long minPrime) {

        this.minPrime = minPrime;

    }

    public void run() {

        // compute primes larger than minPrime

        . . .

    }

}
```

El siguiente código crearía un hilo y lo comenzaría a ejecutar:

```
PrimeThread p = nuevo PrimeThread (143);

    p.start ();
```

La otra forma de crear un hilo es declarar una clase que implemente la interfaz Runnable. Esa clase luego implementa el método run. Luego se puede asignar una instancia de la clase, pasarla como argumento cuando creamos Thread e iniciarlo. El mismo ejemplo en este otro estilo tiene el siguiente aspecto:

```
class PrimeRun implements Runnable {
        long minPrime;
        PrimeRun(long minPrime) {
            this.minPrime = minPrime;
        }
        public void run() {
            // compute primes larger than minPrime
             . . .
        }
    }
```

El siguiente código crea un thread y lo inicia:

```
 PrimeRun p = new PrimeRun(143);
    new Thread(p).start();
```

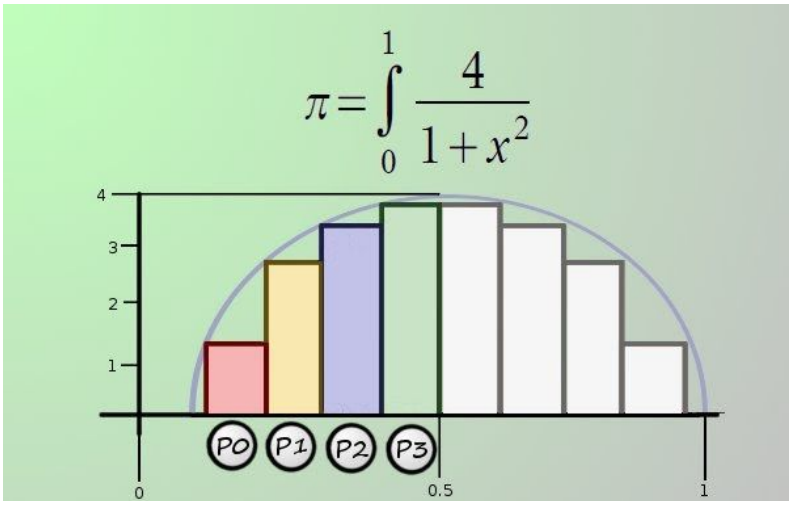
Para sincronizar los threads, supongamos que ya tenemos creados e iniciados threads, podemos iniciarlos y sincronizarlos de la siguiente manera:

```
t1.start();
t2.start();
t3.start();
try {
          t1.join();
          t2.join();
          t3.join();
       } catch (InterruptedException e) {
                  e.printStackTrace();
       }
```

Donde el thread t2 esperará a que termine de ejecutar el thread t1 y el thread t3 esperará a que termine de ejecutar el thread t2.

2.2 Cálculo de pi por integrales

El método de la aproximación de la integral del Cálculo de Pi, La aproximación de una integral mediante la suma de Riemann permite dividir el trabajo en unidades independientes en la que mayor número de particiones mayor precisión.



Pero lo que estaríamos calculando es:

$$\Pi = \sum_{x=0}^{N} \frac{4}{1+x^2} \Delta x$$

El algoritmo que usaremos será el siguiente:

- Dividimos la longitud de 0 a 1 en la cantidad de Particiones a tratar N.
- Hacemos el cálculo para cada partición de la sumatoria.
- Finalmente sumamos todas los resultados de las particiones.

2.3 Algoritmo merge sort

Conceptualmente funciona de la siguiente manera:

- Si la longitud de la lista es 0 o 1, entonces ya está ordenada. En otro caso:
- Dividir la lista desordenada en dos sublistas de aproximadamente la mitad del tamaño.
- Ordenar cada sublista recursivamente aplicando el ordenamiento por mezcla.
- Mezclar las dos sublistas en una sola lista ordenada.

Graficamente:

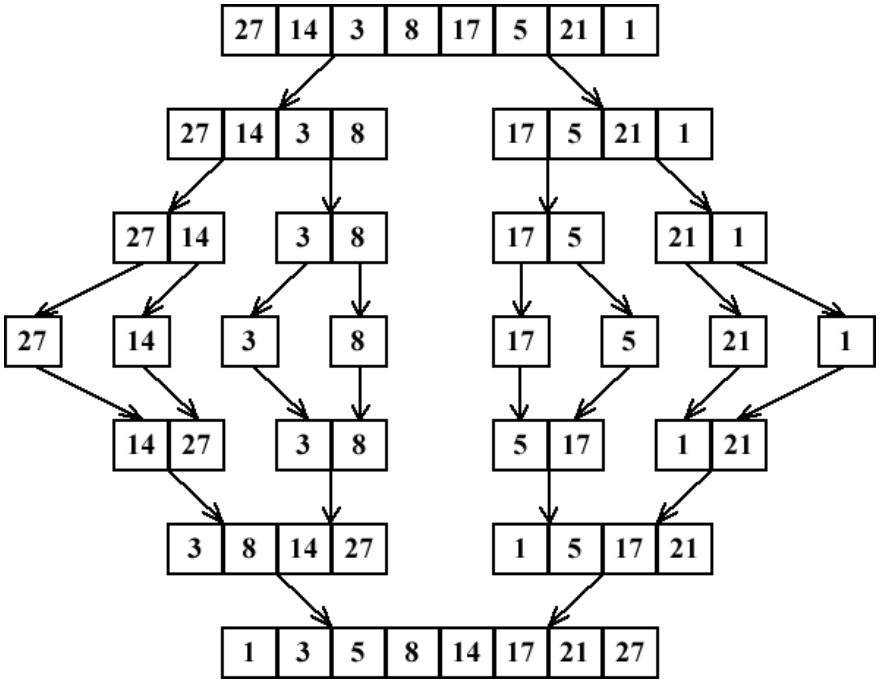


Figura 1: Merge sort de manera grafica

3 Metodología

3.1 Merge Sort

Para paralelizar merge sort implementamos un runnable en cada hilo que tomara una parte de la recursión hasta el máximo de hilos que pueda dividir.

3.2 Calculo de pi

El tipo/objeto BigDecimal de Java nos permite hacer operaciones aritméticas con un número indefinido de cifras

significativas sin importar el epsilon o error de máquina, esto nos permite obtener un valor experimental de Pi con

menor sesgo respecto a su valor real.

4 Results and Discussion

Results:

- Para el algoritmo merge sort incrementando la cantidad de hilos mejoramos la velocidad de un hilo a dos hilos usando 98304000 en 44.64 % y de dos hilos a cuatro hilos en 41.38 %. Como se muestra en la siguiente tabla:

		Tiempo(ms)		
	Elementos	un hilo	dos hilos	cuatro hilos
1	1500	1	1	1
2	3000	1	1	1
3	6000	2	1	2

4	12000	3	1	2
5	24000	4	2	2
6	48000	8	6	2
7	96000	15	8	4
8	192000	36	12	8
9	384000	59	26	15
10	768000	105	57	30
11	1536000	253	112	69
12	3072000	514	237	142
13	6144000	1156	495	323
14	12288000	2224	1141	701
15	24576000	4175	2141	1294
16	49152000	8415	4621	2710
17	98304000	17633	9760	5721

Tabla 1: Muestra los elementos ordenados y el tiempo que uso para ordenar usando un determinada cantidad de hilos.

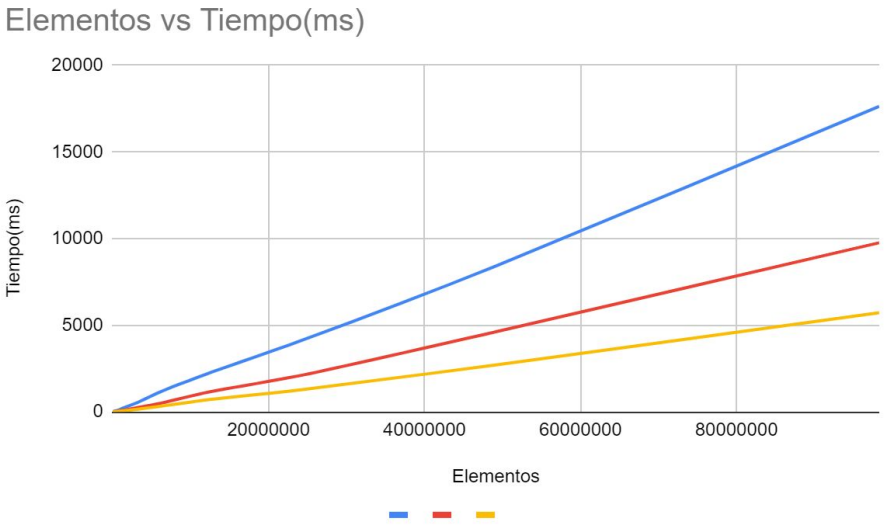
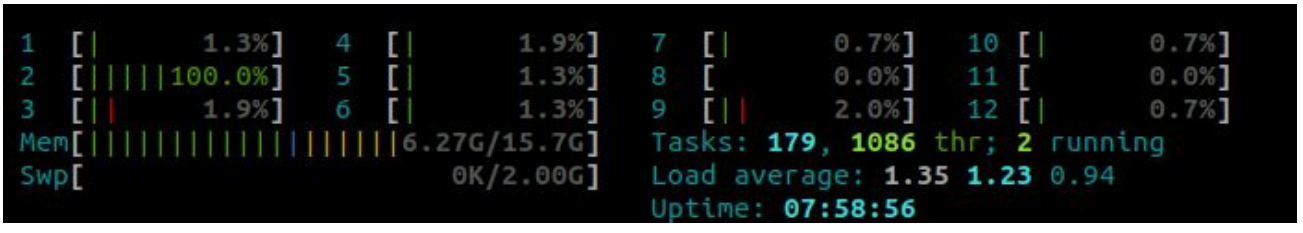
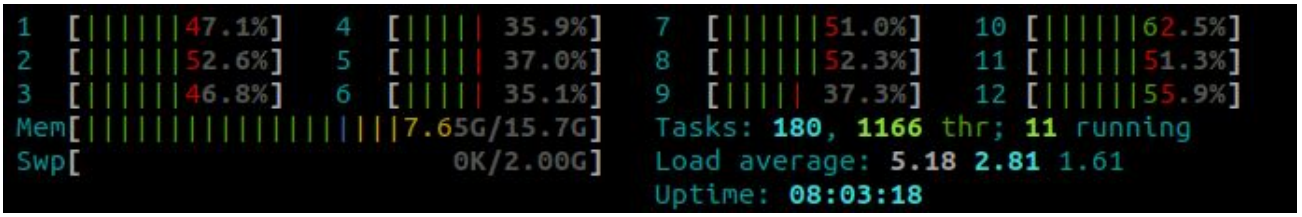


Figura 2: Gráfico de los elementos vs tiempo.

La línea azul es para un hilo, la roja para dos hilos y la amarilla para cuatro hilos





- Para el algoritmo de pi tenemos la siguiente tabla:

		Tiempo(ms)	
	Elementos	un hilo	cuatro hilos
1	1000	22	26
2	2000	9	25
3	4000	9	31
4	8000	23	53
5	16000	23	31
6	32000	47	45
7	64000	105	142
8	128000	145	158
9	256000	226	242
10	512000	418	181
11	1024000	723	322
12	2048000	1397	565
13	4096000	2738	1148
14	8192000	5510	2011
15	16384000	12600	4766
16	32768000	26515	10423
17	65536000	54602	22113
18	131072000	107906	41623
19	262144000	229490	82739
20	524288000	474998	162023

Tabla N2: Muestra los experimentos realizados y el tiempo que uso para un determinada cantidad de hilos.

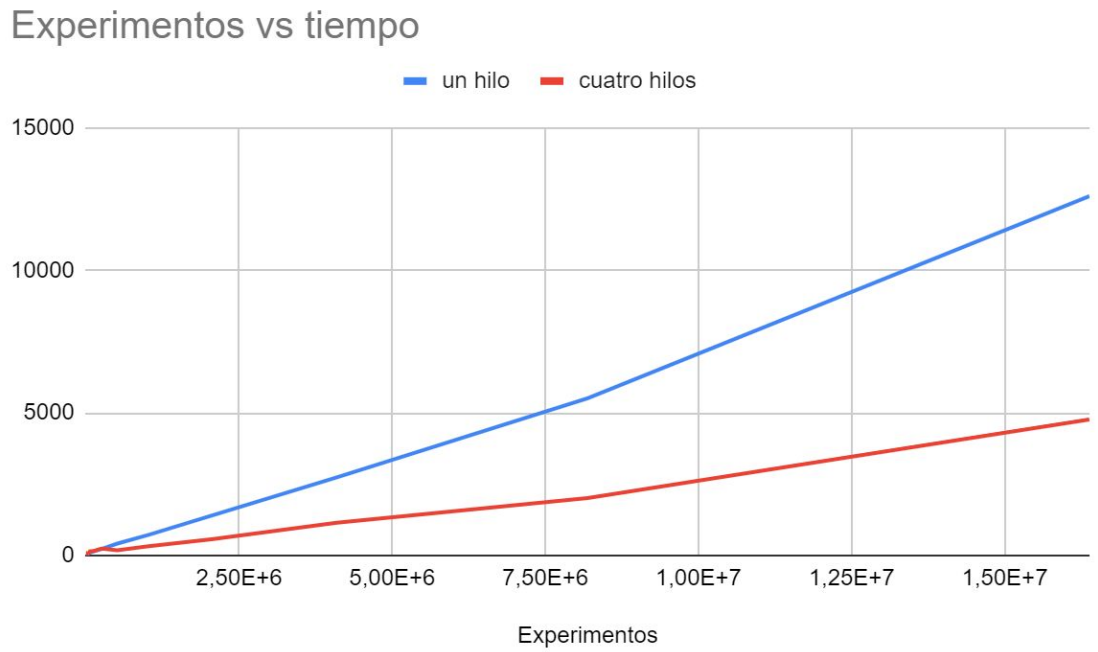
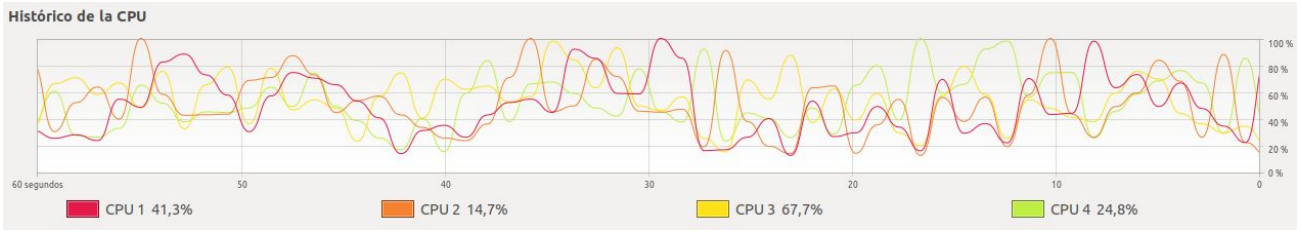
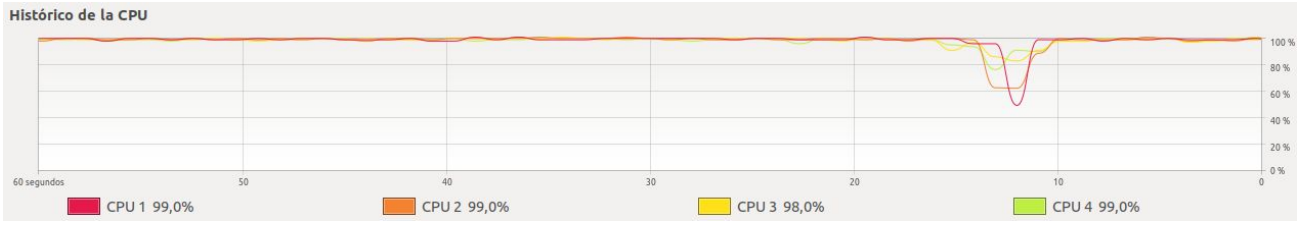


Figura 2: Gráfico de los experimentos vs tiempo

Comportamiento del algoritmo Pi Secuencial



Comportamiento de Pi Paralelo



Se observa que en el cálculo de Pi Secuencial a pesar de estar programado para 1 hilo se usan todos, pero siempre uno consume mucho más que el resto.

En el cálculo de Pi Paralelo todos hilos presentan el similar %uso .

5 Conclusiones

- Java posee herramientas para manejar de threads muy desarrolladas que facilitan implementar los algoritmos.

- Usar hilos para el algoritmo de Merge-sort mejora considerablemente su tiempo de ejecución ya que es altamente paralelizable.

- En el algoritmo del Cálculo de Pi mejora enormemente el tiempo de ejecución al paralelizarlo ya que se aprovechan mejor los recursos obteniendo una mejora del 65%.

6 Anexo Código

- https://github.com/lfloresh/CC462Concurrentes

7 Anexo Documentación

- https://lsi.ugr.es/jmantas/ppr/tutoriales/tutorial mpi.php?tuto=03 pi
- https://www.geeksforgeeks.org/merge-sort/