

GUIA #6

PATRONES Y TECNOLOGÍAS DE CONCURRENCIA Y PARALELISMO

TALLER DE PROGRAMACIÓN
OLGA LUCÍA ROA BOHÓRQUEZ

KAROL STEFANY ORDOÑEZ PEÑALOZA
YEFERSON PIÑEROS PEÑA

22/10/2024

PREGUNTAS ORIENTADORAS

¿Cuáles fueron los aprendizajes obtenidos al realizar esta guía?, liste como mínimo 3 aprendizajes y relaciónelos con su futuro quehacer profesional.

- Aprender a manejar múltiples tareas concurrentes mediante el uso de threads usando herramientas como “lock” y “ReentrantLock” es útil para el desarrollo de algoritmos de aplicaciones que requieren muchas tareas y conexiones entre estas mismas.
- La implementación de paralelismo que nos permite optimizar procesos computacionales usando varios hilos para reducir el tiempo de ejecución es esencial en áreas como análisis de datos y IA.
- Simular estos escenarios problemáticos o desafiantes de la vida real y darles resolución con los temas vistos en clase nos prepara para afrontar problemas de resolución lógica en diferentes áreas de trabajo del software que requieren modelados de estos escenarios.

¿Dónde presentó mayor dificultad resolviendo la guía? y ¿cómo lo resolvieron? ¿Cuáles fueron las estrategias de solución?

- Se presentó dificultad en la comprensión del uso de los “threads” al ser un tema nuevo, pero se pudo solucionar buscando información en diferentes fuentes y códigos que los implementan para tener una mayor claridad de su uso.

ACTIVIDAD DE TRABAJO AUTÓNOMO

1. ¿Qué es paralelismo en datos?

- “El paralelismo de datos es un paradigma informático paralelo en el que una tarea grande se divide en subtareas más pequeñas, independientes y procesadas simultáneamente”(purestorage. 2024. p1.) De esta manera, diferentes unidades de procesamiento realizan una misma operación en varios datos simultáneamente, lo cual hace más eficiente el trabajo y mejora la velocidad informática.

El paralelismo de datos funciona en 5 pasos los cuales son: Dividir en fragmentos, que consiste en hacer un gran conjunto de datos en pequeños grupos. Procesamiento distribuido, ya divididos los datos, se refiere cada uno

con un procesador o subprocesador (dada la exigencia del trabajo) esto para que hagan hacer trabajo paralelo. Procesamiento simultáneo, cada procesador trabaja en su fragmento lo que reduce el tiempo de trabajo en general.

Replicación, una operación es aplicada a todos los fragmentos de manera independiente. Agregación, una vez finalizados todos los procesos anteriores, se unifican para obtener un resultado en conjunto que será la entrega final.

2. ¿Qué es paralelismo en tareas?

- “En el Runtime de simultaneidad, una tarea es una unidad de trabajo que realiza un trabajo específico y que se suele ejecutar en paralelo con otras tareas. Una tarea se puede descomponer en tareas adicionales más específicas que se organizan en un grupo de tareas.”(Microsoft Learn. 2024)

Generalmente podemos encontrar el paralelismo de tareas representado con grafos y también subgrafos de tareas los cuales son asignados en diferentes procesadores. su eficiencia depende en la manera en la que se aborde y divida el grafo. En pocas palabras es un problema divisible en subtareas que se realizan de manera concurrente mediante los diferentes componentes de proceso, como los procesadores e hilos.

3. Implementación de ejemplo sobre paralelismo en tareas y datos.

- **Paralelismo en datos:** Sumaremos los elementos de un array de enteros en paralelo para aprovechar el procesamiento múltiple. Para esto, vamos a dividir un array en partes pequeñas por medio de “Forkjoinpool”. Luego, usaremos la clase “SumaParalela”, que extiende “RecursiveTask<Integer>”. Esta clase define cómo dividir el array y sumar sus elementos.

Si el tamaño de la parte del array es menor o igual al umbral que definimos como “límite”, se realiza la suma de manera secuencial, pero si el tamaño es mayor, el array se divide en dos partes, y se crean tareas paralelas para sumar cada una de ellas.

Por último, los resultados de las sumas parciales se combinan para obtener el resultado total.

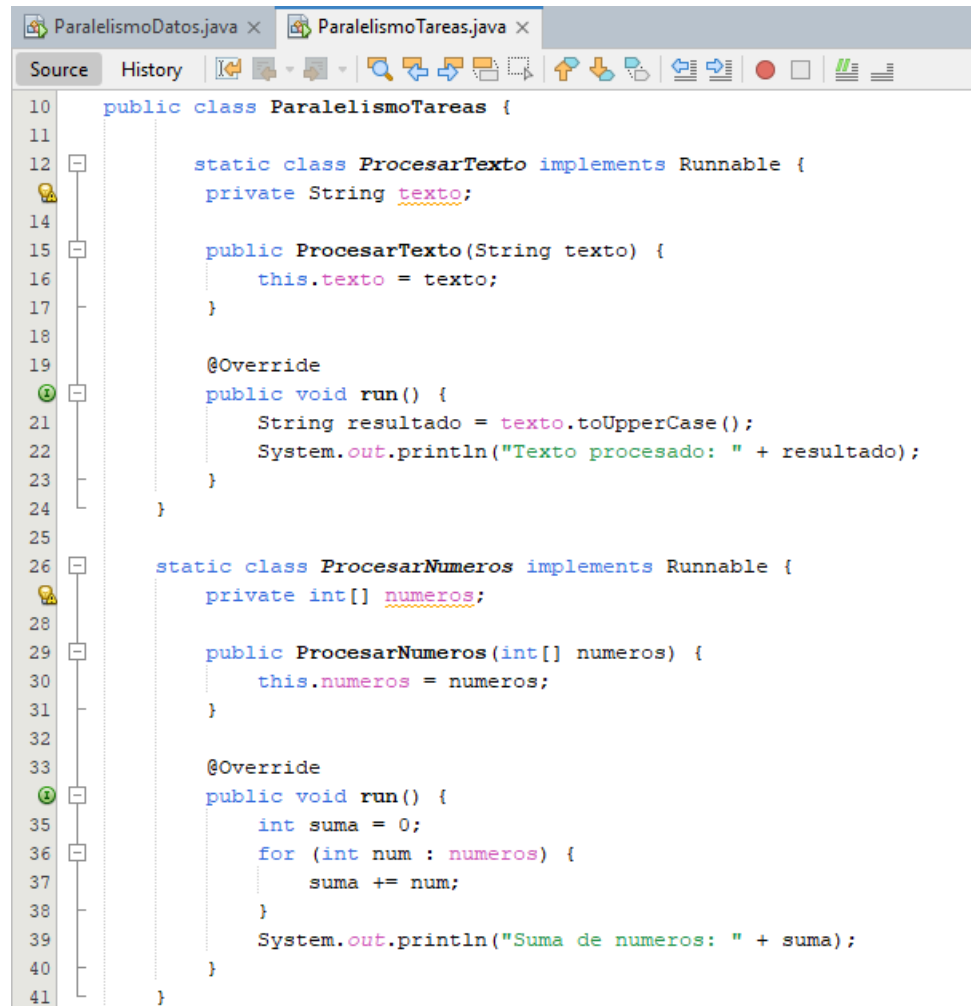
```
ParalelismoDatos.java x ParalelismoTareas.java x
Source History
10 import java.util.concurrent.RecursiveTask;
11 import java.util.concurrent.ForkJoinPool;
12
13 public class ParalelismoDatos {
14
15     static class SumaParalela extends RecursiveTask<Integer> {
16         private final int[] array;
17         private final int inicio, fin;
18         private static final int LIMITE = 2;
19
20         public SumaParalela(int[] array, int inicio, int fin) {
21             this.array = array;
22             this.inicio = inicio;
23             this.fin = fin;
24         }
25
26         @Override
27         protected Integer compute() {
28             if (fin - inicio <= LIMITE) {
29                 int suma = 0;
30                 for (int i = inicio; i < fin; i++) {
31                     suma += array[i];
32                 }
33                 return suma;
34             } else {
35                 int medio = (inicio + fin) / 2;
36                 SumaParalela tareaIzquierda = new SumaParalela(array, inicio, medio);
37                 SumaParalela tareaDerecha = new SumaParalela(array, medio, fin);
38
39                 tareaIzquierda.fork();
40                 int sumaDerecha = tareaDerecha.compute();
41                 int sumaIzquierda = tareaIzquierda.join();
42
43                 return sumaIzquierda + sumaDerecha;
44             }
45         }
46     }
47
48     public static void main(String[] args) {
49         ForkJoinPool pool = new ForkJoinPool();
50         int[] array = {1, 2, 3, 4, 5, 6, 7, 8};
51
52         SumaParalela tarea = new SumaParalela(array, 0, array.length);
53         int resultado = pool.invoke(tarea);
54
55         System.out.println("Resultado final de la suma: " + resultado);
56     }
57 }
```

Salida:

```
Output - EjemploParalelismoEnDatosyTareas (run)
run:
Resultado final de la suma: 36
BUILD SUCCESSFUL (total time: 0 seconds)
```

- **Paralelismo en tareas:** Procesaremos dos tareas diferentes en paralelo. una será convertir un texto a mayúsculas y la otra sumar los elementos de un array de números. Para esto, se crean dos clases que implementan la interfaz Runnable, una para convertir texto a mayúsculas “ProcesarTexto” y otra para

sumar los elementos de un array “ProcesarNumeros”. Cada una de estas clases se ejecuta en su propio Thread, lo que permite que ambas tareas se realicen simultáneamente. Utilizamos el método “start()” para iniciar la ejecución de cada hilo, y “join()” para asegurarnos de que el hilo principal espere a que ambos hilos terminen antes de continuar.



```
10 public class ParalelismoTareas {
11
12     static class ProcesarTexto implements Runnable {
13         private String texto;
14
15         public ProcesarTexto(String texto) {
16             this.texto = texto;
17         }
18
19         @Override
20         public void run() {
21             String resultado = texto.toUpperCase();
22             System.out.println("Texto procesado: " + resultado);
23         }
24     }
25
26     static class ProcesarNumeros implements Runnable {
27         private int[] numeros;
28
29         public ProcesarNumeros(int[] numeros) {
30             this.numeros = numeros;
31         }
32
33         @Override
34         public void run() {
35             int suma = 0;
36             for (int num : numeros) {
37                 suma += num;
38             }
39             System.out.println("Suma de numeros: " + suma);
40         }
41     }
42 }
```

```

41 L      }
42
43 public static void main(String[] args) {
44     String texto = "hola";
45     int[] numeros = {1, 2, 3, 4, 5};
46
47     Thread hiloTexto = new Thread(new ProcesarTexto(texto));
48     Thread hiloNumeros = new Thread(new ProcesarNumeros(numeros));
49
50     hiloTexto.start();
51     hiloNumeros.start();
52
53     try {
54         hiloTexto.join();
55         hiloNumeros.join();
56     } catch (InterruptedException e) {
57         e.printStackTrace();
58     }
59 }
60 }

```

Salida:

```

Output - EjemploParalelismoEnDatosyTareas (run)

run:
Texto procesado: HOLA
Suma de numeros: 15
BUILD SUCCESSFUL (total time: 0 seconds)

```

4. Principales problemas en paralelismo de datos y tareas.

- **Paralelismo en datos:** El paralelismo en datos se basa en dividir grandes volúmenes de datos para procesarlos simultáneamente. Los principales problemas son:
 1. Las dependencias generan la necesidad de sincronización entre los hilos, lo que puede reducir el paralelismo efectivo. Si un hilo debe esperar a que otro termine de procesar su parte de los datos, se genera un estancamiento.
 2. Los hilos que tienen menos trabajo van a terminar antes, quedando inactivos mientras los otros siguen procesando. Esto reduce la eficiencia general, ya que no todos los recursos están siendo utilizados de manera efectiva.
 3. La comunicación entre nodos puede ser lenta o costosa, y si la cantidad de datos que deben intercambiarse es muy grande, el tiempo dedicado a la comunicación puede reducir el beneficio del paralelismo.

- **Paralelismo en tareas:** El paralelismo en tareas implica dividir el trabajo en subtareas independientes que pueden ser ejecutadas simultáneamente. Los problemas principales son:
 1. Cuando las tareas dependen unas de otras, no pueden ejecutarse al mismo tiempo, lo que reduce la efectividad del paralelismo. Las tareas deben ejecutarse secuencialmente, lo que puede anular los beneficios del paralelismo.
 2. El uso de mecanismos de sincronización como "Lock" semáforos, o barreras puede introducir sobrecarga en el sistema y provocar situaciones como deadlocks (dos o más tareas se bloquea esperando recursos una de la otra) o livelocks (las tareas están activas pero no progresan).
 3. Si las tareas son demasiado pequeñas, el overhead asociado a la creación, gestión y sincronización de tareas puede superar el beneficio del paralelismo. Si son demasiado grandes, se pierde potencial de paralelismo, ya que hay menos tareas que ejecutar simultáneamente.
 4. Si el número de hilos o procesos es superior a la cantidad de tareas independientes, algunos de ellos quedarán inactivos, reduciendo la eficiencia. Además, algunas tareas pueden no ser divisibles en partes más pequeñas, lo que limita la escalabilidad del sistema.

PROCEDIMIENTO Y METODOLOGÍA DE LA PRÁCTICA

1. Analice y comprenda el siguiente diagrama de clases; cree un programa en java que permita simular un call Center haciendo uso de Hilos.

```

1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this
4   */
5  package punto_1;
6  import java.util.Random;
7  import java.util.concurrent.locks.Lock;
8  import java.util.concurrent.locks.ReentrantLock;
9
10 /**
11  *
12  * @author Karito
13  */
14
15 class Telefono {
16     private String numero;
17
18     public Telefono(String numero) {
19         this.numero = numero;
20     }
21
22     public String getNumero() {
23         return numero;
24     }
25 }
26
27 class Llamada {
28     private Telefono telefono;
29     private boolean esEntrante;
30     private int duracion;

```

```

31
32     public Llamada(Telefono telefono, boolean esEntrante) {
33         this.telefono = telefono;
34         this.esEntrante = esEntrante;
35         this.duracion = new Random().nextInt(10) + 1;
36     }
37
38     public void realizarLlamada() {
39         if (esEntrante) {
40             System.out.println("Llamada entrante desde el telefono: " + telefono.getNumero());
41         } else {
42             System.out.println("Llamada saliente hacia el telefono: " + telefono.getNumero());
43         }
44     }
45
46     public int getDuracion() {
47         return duracion;
48     }
49 }
50
51 class Persona {
52     private String nombre;
53
54     public Persona(String nombre) {
55         this.nombre = nombre;
56     }
57
58     public void realizarLlamada(Llamada llamada) {
59         System.out.println(nombre + " esta realizando una llamada.");
60         llamada.realizarLlamada();

```



```

61     }
62
63     public String getNombre() {
64         return nombre;
65     }
66 }
67
68 class Asistente {
69     private String nombre;
70
71     public Asistente(String nombre) {
72         this.nombre = nombre;
73     }
74
75     public String getNombre() {
76         return nombre;
77     }
78
79     public void atenderLlamada(Persona persona, Llamada llamada) {
80         System.out.println("Asistente " + nombre + " esta atendiendo la llamada de " + persona.getNombre());
81     }
82 }
83
84 class CallCenter implements Runnable {
85     private Persona persona;
86     private Llamada llamada;
87     private Asistente asistente;
88     private static Lock lock = new ReentrantLock();
89
90     public CallCenter(Persona persona, Llamada llamada, Asistente asistente) {

```

```

91         this.persona = persona;
92         this.llamada = llamada;
93         this.asistente = asistente;
94     }
95
96     @Override
97     public void run() {
98         try {
99             int tiempoEspera = new Random().nextInt(5) + 1;
100             System.out.println("\n*****");
101             System.out.println(persona.getNombre() + " esta en espera por " + tiempoEspera + " segundos.");
102             Thread.sleep(tiempoEspera * 1000);
103
104             lock.lock();
105             try {
106                 asistente.atenderLlamada(persona, llamada);
107                 persona.realizarLlamada(llamada);
108                 System.out.println("La llamada durara " + llamada.getDuracion() + " segundos.");
109                 Thread.sleep(llamada.getDuracion() * 1000);
110                 System.out.println("Llamada finalizada por " + persona.getNombre());
111             } finally {
112                 lock.unlock();
113             }
114             System.out.println("*****\n");
115         } catch (InterruptedException e) {
116             e.printStackTrace();
117         }
118     }
119 }
120

```

```

121 public class PUNTO_1 {
122
123     /**
124      * @param args the command line arguments
125      */
126     public static void main(String[] args) {
127         Telefono telefonol = new Telefono("3186701159");
128         Telefono telefono2 = new Telefono("3209671659");
129         Telefono telefono3 = new Telefono("3204048789");
130
131         Llamada llamada1 = new Llamada(telefonol, true);
132         Llamada llamada2 = new Llamada(telefono2, false);
133         Llamada llamada3 = new Llamada(telefono3, true);
134
135         Persona personal = new Persona("Yeferson");
136         Persona persona2 = new Persona("Karol");
137         Persona persona3 = new Persona("Carlos");
138
139         Asistente asistentel = new Asistente("Lucas");
140         Asistente asistente2 = new Asistente("Dania");
141         Asistente asistente3 = new Asistente("Lina");
142
143         CallCenter callCenter1 = new CallCenter(personal, llamada1, asistentel);
144         CallCenter callCenter2 = new CallCenter(persona2, llamada2, asistente2);
145         CallCenter callCenter3 = new CallCenter(persona3, llamada3, asistente3);
146
147
148         Thread thread1 = new Thread(callCenter1);
149         Thread thread2 = new Thread(callCenter2);
150         Thread thread3 = new Thread(callCenter3);
151
152
153         thread1.start();
154         thread2.start();
155         thread3.start();
156
157
158         try {
159             thread1.join();
160             thread2.join();
161             thread3.join();
162         } catch (InterruptedException e) {
163             e.printStackTrace();
164         }
165
166         System.out.println("Simulacion del CallCenter finalizada.");
167     }
168 }
169

```

```
Output - PUNTO_1 (run)

run:

*****

*****

*****

Carlos esta en espera por 5 segundos.
Karol esta en espera por 5 segundos.
Yeferson esta en espera por 5 segundos.
Asistente Lucas esta atendiendo la llamada de Yeferson
Yeferson esta realizando una llamada.
Llamada entrante desde el telefono: 3186701159
La llamada durara 2 segundos.
Llamada finalizada por Yeferson
*****

Asistente Lina esta atendiendo la llamada de Carlos
Carlos esta realizando una llamada.
Llamada entrante desde el telefono: 3204048789
La llamada durara 9 segundos.
Llamada finalizada por Carlos
*****

Asistente Dania esta atendiendo la llamada de Karol
Karol esta realizando una llamada.
Llamada saliente hacia el telefono: 3209671659
La llamada durara 9 segundos.
Llamada finalizada por Karol
*****

Simulacion del CallCenter finalizada.
BUILD SUCCESSFUL (total time: 25 seconds)
```

2. Desarrolle el siguiente ejercicio: “simular el proceso de cobro de un supermercado; es decir, unos clientes van con un carro lleno de productos y una cajera les cobra los productos, pasándose uno a uno por el escáner de la caja registradora. En este caso la cajera debe de procesar la compra cliente a cliente”

```

1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this
4   */
5   package punto_2;
6   import java.util.LinkedList;
7   import java.util.Queue;
8
9   /**
10    *
11    * @author Karito
12    */
13
14   class Producto {
15       private String nombre;
16       private double precio;
17
18       public Producto(String nombre, double precio) {
19           this.nombre = nombre;
20           this.precio = precio;
21       }
22
23       public String getNombre() {
24           return nombre;
25       }
26
27       public double getPrecio() {
28           return precio;
29       }
30   }

```

```

31
32 class Cliente {
33     private String nombre;
34     private Queue<Producto> carro;
35
36     public Cliente(String nombre) {
37         this.nombre = nombre;
38         this.carro = new LinkedList<>();
39     }
40
41     public void agregarProducto(Producto producto) {
42         carro.add(producto);
43     }
44
45     public Queue<Producto> getCarro() {
46         return carro;
47     }
48
49     public String getNombre() {
50         return nombre;
51     }
52 }
53
54 class Cajera implements Runnable {
55     private String nombre;
56     private Cliente cliente;
57
58     public Cajera(String nombre, Cliente cliente) {
59         this.nombre = nombre;
60         this.cliente = cliente;

```

```

61     }
62
63     @Override
64     public void run() {
65         System.out.println("Cajera " + nombre + " está procesando la compra de " + cliente.getNombre());
66         double total = 0;
67
68         while (!cliente.getCarro().isEmpty()) {
69             Producto producto = cliente.getCarro().poll();
70             System.out.println("Escaneando producto: " + producto.getNombre() + " - $" + producto.getPrecio());
71             total += producto.getPrecio();
72
73             // Simular tiempo de escaneo
74             try {
75                 Thread.sleep(1000); // 1 segundo por producto
76             } catch (InterruptedException e) {
77                 e.printStackTrace();
78             }
79         }
80
81         System.out.println("Compra de " + cliente.getNombre() + " procesada. Total a pagar: $" + total);
82         System.out.println("-----");
83     }
84 }
85
86 public class PUNTO_2 {
87
88     /**
89     * @param args the command line arguments
90     */

```

```

91 public static void main(String[] args) {
92     // Crear productos
93     Producto p1 = new Producto("Manzana", 0.50);
94     Producto p2 = new Producto("Pan", 1.50);
95     Producto p3 = new Producto("Leche", 2.30);
96     Producto p4 = new Producto("Huevos", 1.70);
97     Producto p5 = new Producto("Arroz", 0.90);
98
99     // Crear clientes y agregar productos a su carro
100     Cliente clientel = new Cliente("Carlos");
101     clientel.agregarProducto(p1);
102     clientel.agregarProducto(p2);
103     clientel.agregarProducto(p3);
104
105     Cliente cliente2 = new Cliente("Ana");
106     cliente2.agregarProducto(p4);
107     cliente2.agregarProducto(p5);
108
109     // Crear hilos para cajeras que procesan a los clientes
110     Thread cajeral = new Thread(new Cajera("Lucía", clientel));
111     Thread cajera2 = new Thread(new Cajera("Marta", cliente2));
112
113     // Iniciar hilos
114     cajeral.start();
115     cajera2.start();
116
117     // Esperar a que todos los hilos terminen
118     try {
119         cajeral.join();
120         cajera2.join();
121
122     } catch (InterruptedException e) {
123         e.printStackTrace();
124     }
125
126     System.out.println("Simulación del supermercado finalizada.");
127 }
128

```

Output - PUNTO_2 (run)

```

run:
Cajera Marta está procesando la compra de Ana
Cajera Lucía está procesando la compra de Carlos
Escaneando producto: Manzana - $0.5
Escaneando producto: Huevos - $1.7
Escaneando producto: Pan - $1.5
Escaneando producto: Arroz - $0.9
Escaneando producto: Leche - $2.3
Compra de Ana procesada. Total a pagar: $2.6
-----
Compra de Carlos procesada. Total a pagar: $4.3
-----
Simulación del supermercado finalizada.
BUILD SUCCESSFUL (total time: 3 seconds)

```

3. Se le solicita realizar un ejemplo: productor/ consumidor que trabaje con 2 Threads, el primer Thread generará números aleatorios entre 1 y 100 que serán leídos y multiplicados por 2 en el segundo Thread el cual imprimirá el resultado.

```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this
4   */
5   package punto_3;
6
7   /**
8    *
9    * @author Karito
10   */
11  public class PUNTO_3 {
12
13      /**
14       * @param args the command line arguments
15       * @throws java.lang.InterruptedException
16      */
17      public static void main(String[] args) throws InterruptedException {
18          int N = 50;
19          double[][] A = generateMatrix(N, N);
20          double[][] B = generateMatrix(N, N);
21          double[][] C = new double[N][N];
22
23          // Imprimir matriz A
24          System.out.println("Matriz A:");
25          printMatrix(A);
26
27          // Imprimir matriz B
28          System.out.println("Matriz B:");
29          printMatrix(B);
30      }
```

```
31      // Multiplicación secuencial
32      long startTime = System.currentTimeMillis();
33
34      multiplyMatricesSequential(A, B, C, N);
35
36      long endTime = System.currentTimeMillis();
37      System.out.printf("Tiempo de ejecución secuencial: %.4f segundos\n", (endTime - startTime) / 1000.0);
38
39      // Imprimir matriz C (resultado secuencial)
40      System.out.println("Matriz C (Resultado Secuencial):");
41      printMatrix(C);
42
43      // Multiplicación con 4 hilos
44      double[][] C_parallel = new double[N][N];
45      startTime = System.currentTimeMillis();
46
47      multiplyMatricesParallel(A, B, C_parallel, N);
48
49      endTime = System.currentTimeMillis();
50      System.out.printf("Tiempo de ejecución con 4 hilos: %.4f segundos\n", (endTime - startTime) / 1000.0);
51
52      // Imprimir matriz C (resultado paralelo)
53      System.out.println("Matriz C (Resultado con 4 Hilos):");
54      printMatrix(C_parallel);
55  }
56
57  // Método secuencial para multiplicar matrices
58  private static void multiplyMatricesSequential(double[][] A, double[][] B, double[][] C, int N) {
59      for (int i = 0; i < N; i++) {
60          for (int j = 0; j < N; j++) {
```

```

61         C[i][j] = 0;
62         for (int k = 0; k < N; k++) {
63             C[i][j] += A[i][k] * B[k][j];
64         }
65     }
66 }
67
68
69 // Método multithread para multiplicar matrices
70 private static void multiplyMatricesParallel(double[][] A, double[][] B, double[][] C, int N) throws InterruptedException {
71     int numThreads = 4;
72     Thread[] threads = new Thread[numThreads];
73
74     for (int t = 0; t < numThreads; t++) {
75         final int threadId = t;
76         threads[t] = new Thread(() -> {
77             int chunkSize = N / numThreads;
78             int startRow = threadId * chunkSize;
79             int endRow = (threadId == numThreads - 1) ? N : (threadId + 1) * chunkSize; // Asegurar que el último hilo procesa
80
81             for (int i = startRow; i < endRow; i++) {
82                 for (int j = 0; j < N; j++) {
83                     C[i][j] = 0;
84                     for (int k = 0; k < N; k++) {
85                         C[i][j] += A[i][k] * B[k][j];
86                     }
87                 }
88             }
89         });
90         threads[t].start();

```

```

91     }
92
93     // Esperar a que todos los hilos terminen
94     for (int t = 0; t < numThreads; t++) {
95         threads[t].join();
96     }
97
98
99 // Generar una matriz de tamaño filas x columnas con valores aleatorios
100 private static double[][] generateMatrix(int rows, int cols) {
101     double[][] matrix = new double[rows][cols];
102     for (int i = 0; i < rows; i++) {
103         for (int j = 0; j < cols; j++) {
104             matrix[i][j] = Math.random() * 100; // Valores entre 0 y 100
105         }
106     }
107     return matrix;
108 }
109
110 // Método para imprimir la matriz
111 private static void printMatrix(double[][] matrix) {
112     int rows = matrix.length;
113     int cols = matrix[0].length;
114     for (int i = 0; i < rows; i++) {
115         for (int j = 0; j < cols; j++) {
116             System.out.printf("%.2f\t", matrix[i][j]);
117         }
118         System.out.println();
119     }
120     System.out.println();

```


Output - PUNTO_3 (run) ×



run:

Matriz A:

39,65	66,78	82,07	73,86	53,44	32,00	87,19	36,90	74,26	75,13
47,89	11,31	45,33	2,77	90,98	39,28	44,16	86,06	74,80	24,28
55,33	89,65	91,70	44,42	6,63	48,53	12,86	28,86	65,98	68,52
83,38	65,25	18,01	3,69	94,39	87,04	37,73	81,89	72,85	42,64
90,35	17,70	60,94	40,20	39,60	28,34	66,30	36,96	54,68	5,93
81,99	2,78	80,76	65,54	98,93	33,25	35,54	60,33	90,33	32,46
56,22	37,80	97,46	9,64	25,99	23,39	12,44	14,21	39,83	52,11
65,58	40,01	6,81	28,37	55,12	51,40	91,02	66,46	40,06	83,58
84,03	33,92	33,40	1,74	37,61	23,24	5,90	85,66	51,86	94,91
58,68	47,61	48,58	15,27	96,70	35,40	20,86	14,25	26,12	37,71
11,46	37,74	91,75	25,76	29,93	12,35	90,47	74,81	96,82	26,88
30,73	17,97	85,48	86,90	40,86	68,14	71,95	91,53	40,80	79,10
58,84	16,83	27,05	50,30	26,25	69,90	85,63	2,41	65,47	80,74
37,66	57,00	71,97	62,89	78,60	50,76	82,61	96,06	98,81	28,38
21,21	30,70	98,53	53,00	89,87	4,80	80,70	31,49	58,76	73,50
2,36	43,13	56,62	34,02	57,28	97,77	71,14	89,19	73,25	7,11
28,22	9,63	58,93	57,35	98,74	43,44	82,38	57,35	6,58	39,50
98,38	21,06	67,78	13,90	82,34	28,08	9,99	78,22	22,67	65,61
49,00	36,51	67,25	17,75	62,07	12,76	86,38	66,48	81,84	57,49
26,37	22,91	86,61	4,08	89,89	7,02	68,75	63,97	23,80	91,66

Output - PUNTO_3 (run) ×



Matriz B:

43,83	36,65	58,51	39,34	43,38	86,19	68,80	13,07	36,68	1,55
97,72	34,24	53,22	71,34	67,52	19,36	34,17	14,81	49,38	42,60
89,57	5,68	28,86	43,77	79,35	85,14	18,01	87,12	12,97	71,34
64,78	47,58	76,51	61,07	24,87	78,56	40,30	53,48	55,45	68,55
30,85	30,65	19,16	17,29	9,00	53,43	52,74	72,85	3,85	81,34
62,30	71,11	43,55	95,44	41,27	2,17	67,17	68,04	67,68	19,12
33,98	84,34	5,28	2,58	52,66	47,44	97,04	21,25	68,40	14,49
49,22	84,93	49,05	30,36	42,12	8,55	90,95	83,28	81,09	51,39
45,44	97,93	28,22	20,43	5,12	9,46	7,81	85,68	26,30	70,11
65,67	62,58	90,79	69,88	81,19	14,97	94,11	53,93	12,71	81,55
82,03	42,07	22,46	32,91	32,43	16,77	75,69	94,22	84,56	61,24
44,21	0,44	1,56	62,02	30,59	42,13	68,37	65,91	71,51	48,00
4,55	71,40	25,48	84,02	86,89	24,16	44,88	57,78	73,09	41,48
40,17	22,44	13,45	24,72	5,55	55,25	69,10	57,12	35,17	78,21
51,94	12,68	76,91	79,40	79,58	42,33	22,71	37,56	91,24	66,47
70,19	52,81	75,02	75,84	75,84	96,74	4,04	93,95	48,11	77,22
37,95	1,52	40,29	91,33	74,87	19,85	74,61	20,89	92,15	19,90
73,64	74,35	67,28	65,96	40,79	90,58	64,60	64,83	80,17	80,11
37,20	97,84	5,79	0,32	96,47	21,20	25,18	79,24	56,94	59,38
36,33	14,85	88,98	37,91	69,94	23,41	33,16	77,33	10,48	36,15



Tiempo de ejecución secuencial: 0,0010 segundos

Matriz C (Resultado Secuencial):

62082,80	56562,67	49160,35	56801,43	60424,81
49887,21	51321,95	36007,76	43345,23	49231,30
57532,74	40630,86	45225,39	50659,45	50665,29
56452,33	56823,11	49147,81	56332,67	60575,39
50477,44	40135,67	41979,33	44171,34	46875,30
52759,47	51313,83	41985,21	45755,95	46132,59
51006,10	31329,29	36963,20	48971,97	45047,37
51668,17	53801,18	42027,79	47334,40	54467,87
48358,96	44342,79	47134,09	46772,35	50239,14
45318,58	41118,64	41053,74	44974,09	52298,30
49539,38	50209,98	36955,29	41279,01	55249,74
56986,53	47101,68	53647,93	57854,99	57981,05
51699,24	46388,63	46729,03	49923,40	47613,90
58260,57	52879,79	40043,37	46937,09	43780,09
56688,39	47927,15	45741,95	49796,20	55187,12
54297,52	55461,86	47083,13	51606,90	53613,92
40473,61	40184,12	35211,79	40748,19	44504,23
56426,45	48237,32	47575,78	55838,86	58209,45
57269,01	49138,84	45436,21	50353,31	51604,25
49252,69	43750,05	33845,92	37444,99	47274,32

Tiempo de ejecución con 4 hilos: 0,0320 segundos

Matriz C (Resultado con 4 Hilos):

62082,80	56562,67	49160,35	56801,43	60424,81
49887,21	51321,95	36007,76	43345,23	49231,30
57532,74	40630,86	45225,39	50659,45	50665,29
56452,33	56823,11	49147,81	56332,67	60575,39
50477,44	40135,67	41979,33	44171,34	46875,30
52759,47	51313,83	41985,21	45755,95	46132,59
51006,10	31329,29	36963,20	48971,97	45047,37
51668,17	53801,18	42027,79	47334,40	54467,87
48358,96	44342,79	47134,09	46772,35	50239,14
45318,58	41118,64	41053,74	44974,09	52298,30
49539,38	50209,98	36955,29	41279,01	55249,74
56986,53	47101,68	53647,93	57854,99	57981,05
51699,24	46388,63	46729,03	49923,40	47613,90
58260,57	52879,79	40043,37	46937,09	43780,09
56688,39	47927,15	45741,95	49796,20	55187,12
54297,52	55461,86	47083,13	51606,90	53613,92
40473,61	40184,12	35211,79	40748,19	44504,23
56426,45	48237,32	47575,78	55838,86	58209,45
57269,01	49138,84	45436,21	50353,31	51604,25
49252,69	43750,05	33845,92	37444,99	47274,32

BUILD SUCCESSFUL (total time: 0 seconds)

DIALOGO DEL VIDEO:

"Let me explain a few key concepts in programming: threads, parallelism, and concurrency. A thread is essentially a single sequence of execution within a program. Think of it as a path that the program takes to perform tasks. Most modern programs use multiple threads to make things more efficient. For example, while one thread is handling user input, another could be processing data in the background.

Now, let's talk about concurrency. Concurrency happens when multiple threads are in progress at the same time. However, it doesn't necessarily mean they are running simultaneously. It just means they're making progress, often switching between each other rapidly. The system decides which thread runs at any given moment, so they appear to be running at the same time, even if they're actually taking turns.

Parallelism, on the other hand, is a step beyond concurrency. Parallelism is when multiple threads or tasks are truly running at the exact same time. To achieve this, you need multiple processors or cores. Each core can handle a different task simultaneously, which leads to faster performance for tasks like large data processing or complex calculations.

In short, concurrency is about managing multiple tasks at the same time, whereas parallelism is about executing them at the same time. Both concepts are vital in modern programming, especially when performance and efficiency are key."

LINK DEL VIDEO: 📺 WIN_20241022_23_02_55_Pro-VEED.mp4

"<https://drive.google.com/file/d/1xyQdc5BC1RWIoFzitH0DzXP-u83cmB/view?usp=sharing>"

BIBLIOGRAFÍA

1. <https://www.purestorage.com/la/knowledge/what-is-data-parallelism.html>. - PURESTORAGE(Octubre-2024).¿Qué es el paralelismo de datos?
2. <https://learn.microsoft.com/es-es/cpp/parallel/concrt/task-parallelism-concurrency-runtime?view=msvc-170>. - Microsoft Learn(octubre 2024). Paralelismo de tareas (Runtime simultaneidad)