



Centro de Enseñanza Técnica Industrial

Desarrollo de Software

Proyecto del segundo parcial

Jesús Alberto Aréchiga Carrillo

22310439 5N

Profesor

José Francisco Pérez Reyes

Octubre 2024

Guadalajara, Jalisco

Introducción

En la computación la cantidad de procesamiento depende de cuantas operaciones se tienen que hacer. Al tener una tarea con mucho procesamiento, se vuelve una carga pesada para el hardware. Al implementar concurrencia se aprovechan más los recursos disponibles del equipo y, de esa manera, hacer más eficiente el procesamiento y tardar menos tiempo haciendo la tarea que se dé.

También es importante mencionar que en ciertas ocasiones la concurrencia es menos eficiente que la secuencia cuando se trata de tareas no muy grandes. El proceso de la concurrencia (y del paralelismo) es dividir la tarea que hay en pedazos más pequeños y asignar una parte del hardware para que trabaje esa división de la tarea principal. Si la tarea es pequeña, se tarda más en dividir el problema y ejecutar una instancia con cada división del problema principal.

Desarrollo

El problema que se va a resolver son multiplicaciones de matrices, para esta tarea se tienen que utilizar matrices grandes para ver la diferencia.

Los códigos para utilizar son:

Dialog:

```
import java.awt.*;
import java.awt.event.*;
import java.io.FileWriter;
import java.io.IOException;
import javax.swing.*;

public class Dialog extends JDialog{

    private JLabel label1;
    private JButton btn1;
    private JTextArea textarea1;

    public Dialog(boolean modal, int[][] matrix, String str) {

        setLayout(null);
        setBounds(480,50,900,750);
        //setAlwaysOnTop(true);

        textarea1=new JTextArea();
        String content;
        content = MatrixMult.print2D(matrix, 100_000);
        textarea1.setText(content);
        textarea1.setFont(new Font("Consolas", 0, 10));
```

```

        JScrollPane scrollableTextArea = new JScrollPane(textarea1);
        scrollableTextArea.setBounds(10,50,860,580);
        scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        add(scrollableTextArea);

        //Etiqueta
        label1 = new JLabel(str);
        label1.setIcon(new ImageIcon(getClass().getResource("/images/result.png")));
        label1.setFont(new Font("Segoe UI", 0, 24));
        label1.setBounds(10,10,600,36);
        add(label1);

        btn1 = new JButton();
        btn1.setText("Guardar");
        btn1.setBounds(400,650,100,25);
        btn1.addActionListener((ActionEvent e) -> {
            try {
                String nombre;
                nombre = JOptionPane.showInputDialog(null, "Guardar como:",
"Matriz" + str.substring(str.length() - 1));
                if ("".equals(nombre) || nombre == null) return;
                nombre += ".txt";
                try (FileWriter myWriter = new FileWriter(nombre)) {
                    for (int[] matrix1 : matrix) {
                        for (int j = 0; j < matrix[0].length; j++) {
                            myWriter.write(Integer.toString(matrix1[j]) + "
");
                        }
                        myWriter.write("\n");
                    }
                    System.out.println("Successfully wrote to the file.");
                } catch (IOException ex) {
                    System.out.println("An error occurred.");
                }
            });
            add(btn1);
        }
    }
}

```

MatrixMult:

```
import java.awt.Color;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.ThreadLocalRandom ;
import javax.swing.*.*;

public class MatrixMult{

    public static int flag = 0;
    public static ExecutorService ex;
    public Proyecto2 proyecto;

    public MatrixMult(Proyecto2 p){
        this.proyecto = p;
    }

    public static int[][] multiply(int m1[][], int m2[][], JProgressBar
pb, JLabel estado) {
        if (m1[0].length != m2.length) {
            throw new IllegalArgumentException("Las matrices no son
compatibles para la multiplicación.");
        }

        int rows = m1.length;
        int cols = m2[0].length;
        int commonDim = m1[0].length;
        int[][] res = new int[rows][cols];

        pb.setMaximum(rows);
        estado.setText("Procesando...");
        estado.setBackground(new Color(0, 255, 255));

        for (int i = 0; i < rows; i++) {
            pb.setValue(i);
            for (int k = 0; k < commonDim; k++) {
                int temp = m1[i][k];
                for (int j = 0; j < cols; j++) {
                    res[i][j] += temp * m2[k][j];
                }
            }
        }

        pb.setValue(rows);
        estado.setText("Finalizado");
    }
}
```

```

        estado.setBackground(new Color(120, 255, 120));

        return res;
    }

    public static int[][] multiplyConcurrent(int m1[][], int m2[][], int
nThreads, JProgressBar[] pbArr, JLabel[] estadoArr) {
        if (m1[0].length != m2.length) {
            throw new IllegalArgumentException("Las matrices no son
compatibles para la multiplicación.");
        }

        int rows = m1.length;
        int cols = m2[0].length;
        int commonDim = m1[0].length;
        int[][] res = new int[rows][cols];
        CountDownLatch latch = new CountDownLatch(nThreads);

        ExecutorService executor =
Executors.newFixedThreadPool(nThreads);
        int blockSize = (int) Math.ceil((double) rows / nThreads);

        for (int h = 0; h < nThreads; h++) {
            final int istart = h * blockSize;
            final int iend = Math.min(istart + blockSize, rows);
            final int threadIndex = h;

            executor.execute(() -> {
                pbArr[threadIndex].setMinimum(istart);
                pbArr[threadIndex].setMaximum(iend);
                estadoArr[threadIndex].setText("Procesando...");
                estadoArr[threadIndex].setBackground(new Color(0, 255,
255));

                for (int i = istart; i < iend; i++) {
                    pbArr[threadIndex].setValue(i);
                    for (int k = 0; k < commonDim; k++) {
                        int temp = m1[i][k];
                        for (int j = 0; j < cols; j++) {
                            res[i][j] += temp * m2[k][j];
                        }
                    }
                }

                pbArr[threadIndex].setValue(iend);
                estadoArr[threadIndex].setText("Finalizado");
            });
        }
    }

```

```

        estadoArr[threadIndex].setBackground(new Color(120, 255,
120));

        latch.countDown();
    });
}

try {
    latch.await();
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
} finally {
    executor.shutdown();
}

return res;
}

public static int[][] generarMatriz(int largo, int ancho){
    if (largo == 0 || ancho == 0)
        throw new ArithmeticException("Dimensiones no válidas");

    int[][] m = new int[largo][ancho];

    //Inicializa la matriz en 0
    for (int i = 0; i < largo ; i++ ) {
        for (int j = 0; j < ancho ; j++ ) {
            m[i][j] = ThreadLocalRandom.current().nextInt(-20, 9);
        }
    }
    return m;
}

public static String print2D(int mat[][], int max){
    String str = "";
    if (mat == null) return str;

    for (int[] mat1 : mat) {
        for (int j = 0; j < mat1.length; j++) {
            str += (mat1[j] + " ");
        }
        str += "\n";
        if (str.length() > max) {
            str += "... [" + mat.length + " X " + mat[0].length +
"]\n...\n";
            for (int j = 0; j < mat1.length; j++) {

```

```

        str += (mat[mat.length-1][j] + " ");
    }
    return str;
}
}
return str;
}

public static void main(String[] args) {
    System.out.println("=====
=====");
    long start, time;
    int[][] m1;
    int[][] m2;
    m1 = generarMatriz(1000,500);
    m2 = generarMatriz(500,1000);

    start = System.nanoTime();
    multiplyConcurrent(m1,m2,10,null,null);
    time = System.nanoTime() - start;
    System.out.printf("Primero: %.1f ms\n", (double) time /
1_000_000);

    System.out.println("\n=====
=====\\n");

    start = System.nanoTime();
    //multiply(m1,m2,null,null);
    time = System.nanoTime() - start;
    System.out.printf("Segundo: took %.1f ms\n", (double) time /
1_000_000);
    //print2D(m1);
}
}
}

```

Proyecto2:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public final class Proyecto2 extends JPanel {
    public static void main(String args[]) {

        Proyecto2 proyecto = new Proyecto2();

        JFrame frame = new JFrame ("MyPanel");
        frame.setSize(1020,900);
        frame.setLocation(450, 50);
        frame.setTitle("Multiplicacion de matrices");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add (proyecto);
        frame.setResizable(false);
        frame.setVisible(true);
    }

    public int maxHilos = 20;

    //Componentes Visuales
    private JLabel labelTitulo;

    private JButton imgM1;
    private JButton imgM2;

    public JLabel labelM1;
    public JLabel labelM2;

    private JButton btnGenerarM1;
    private JButton btnGenerarM2;

    private JButton btnCalculoSecuencial;
    private JButton btnCalculoConcurrente;
    private JButton btnCalculoSyc;

    private JLabel labelSecuencial;
    private JLabel jLabel1;

    private JLabel imgHiloSec;
    private JLabel labelHiloSec;
    private JProgressBar pbSec;
    private JLabel labelResSec;
    private JButton btnResultadoSec;
```



```

private JLabel labelNumeroHilos;
private JButton btnRemoverHilo;
private JButton btnAgregarHilo;
private JLabel labelResConc;
private JButton btnResultadoConc;

private JProgressBar[] pbsConc = new JProgressBar[maxHilos];
private JLabel[] labelsHiloConc = new JLabel[maxHilos];

//Variables
public int nHilos;
public int[][] m1;
public int[][] m2;
public int[][] resultSec;
public int[][] resultConc;

public int rowsM1;
public int colsM1;
public int rowsM2;
public int colsM2;
public boolean error = false;

public ThreadMultSecuencial hiloSecuencial;
public ThreadMultConcurrente hiloConcurrente;

public Proyecto2(){

    nHilos = 10;
    crearComponentes();

    m1 = MatrixMult.generarMatriz(1000, 1000);
    m2 = MatrixMult.generarMatriz(1000, 1000);

    inicio();

    //Generar M1
    btnGenerarM1.addActionListener((ActionEvent e) -> {
        labelM1.setText("Generando...");
        String inp1 = JOptionPane.showInputDialog(null, "Matriz
A\nIngresa el numero de filas: ");
        String inp2 = JOptionPane.showInputDialog(null, "Matriz
A\nIngresa el numero de columnas: ");
        try{
            rowsM1 = Integer.parseInt(inp1);
            colsM1 = Integer.parseInt(inp2);
            m1 = MatrixMult.generarMatriz(rowsM1, colsM1);

```

```

        }catch(NumberFormatException ex){
JOptionPane.showMessageDialog(null,"Matriz A\nValores no
validos","Error",JOptionPane.ERROR_MESSAGE);}

        if (m1 != null){
            rowsM1 = m1.length;
            colsM1 = m1[0].length;
            String str = "<html><body><div style=\"text-align:center;
width:80px;\>[" + rowsM1 + " X " + colsM1 + "]\>";
            int aux = m1.length;
            if (m1.length > 3)
                aux = 3;
            for (int i = 0; i < aux ; i++ ) {
                str += "<p style=\"white-space:nowrap;text-
align:center; width:80px;\>" + arrToStr(m1[i]) + "</p>";
            }

            str += "</body></html>";
            labelM1.setText(str);
        }
        else
            labelM1.setText("No hay ninguna matriz");
    });
    //Generar M2
    btnGenerarM2.addActionListener((ActionEvent e) -> {
        labelM2.setText("Generando...");
        String inp1 = JOptionPane.showInputDialog(null, "Matriz
B\nIngresa el numero de filas: ");
        String inp2 = JOptionPane.showInputDialog(null, "Matriz
B\nIngresa el numero de columnas: ");
        try{
            rowsM2 = Integer.parseInt(inp1);
            colsM2 = Integer.parseInt(inp2);
            m2 = MatrixMult.generarMatriz(rowsM2, colsM2);
        }catch(NumberFormatException ex){
            JOptionPane.showMessageDialog(null,"Matriz B\nValores no
validos","Error",JOptionPane.ERROR_MESSAGE);
        }

        if (m2 != null){
            rowsM2 = m2.length;
            colsM2 = m2[0].length;
            String str = "<html><body><div style=\"text-align:center;
width:80px;\>[" + rowsM2 + " X " + colsM2 + "]\></div>";
            int aux = m2.length;
            if (m2.length > 3)
                aux = 3;
            for (int i = 0; i < aux ; i++ ) {

```

```

        str += "<p style=\"white-space:nowrap;text-align:center; width:80px;\">" + arrToStr(m2[i]) + "</p>";
    }

    str += "</body></html>";
    labelM2.setText(str);
}
else
    labelM2.setText("No hay ninguna matriz");
});

//Calculo secuencial
btnCalculoSecuencial.addActionListener((ActionEvent e) -> {
    if (colsM1 != rowsM2){
        JOptionPane.showMessageDialog(null,"Las columnas de A y
las filas de B deben tener el mismo
tamaño","Error",JOptionPane.ERROR_MESSAGE);
        return;
    }
    limpiar();
    hiloSecuencial = new ThreadMultSecuencial();
    hiloSecuencial.start();
});

//Calculo concurrente
btnCalculoConcurrente.addActionListener((ActionEvent e) -> {
    if (colsM1 != rowsM2){
        JOptionPane.showMessageDialog(null,"Las columnas de A y
las filas de B deben tener el mismo
tamaño","Error",JOptionPane.ERROR_MESSAGE);
        return;
    }
    limpiar();
    hiloConcurrente = new ThreadMultConcurrente();
    hiloConcurrente.start();
});

//Calculo Sec&Conc
btnCalculoSyC.addActionListener((ActionEvent e) -> {
    if (colsM1 != rowsM2){
        JOptionPane.showMessageDialog(null,"Las columnas de A y
las filas de B deben tener el mismo
tamaño","Error",JOptionPane.ERROR_MESSAGE);
        return;
    }
    limpiar();
    hiloSecuencial = new ThreadMultSecuencial();
    hiloSecuencial.start();
    hiloConcurrente = new ThreadMultConcurrente();

```

```

        hiloConcurrente.start();
    });

    //Incrementar hilos
    btnAgregarHilo.addActionListener((ActionEvent e) -> {
        if ( nHilos < maxHilos) {
            nHilos += 1;
            labelNumeroHilos.setText("Numero de hilos: " + nHilos);
            pbsConc[nHilos-1].setMaximum(100);
            pbsConc[nHilos-1].setValue(0);
            pbsConc[nHilos-1].setVisible(true);
            labelsHiloConc[nHilos-1].setVisible(true);
            pbsConc[nHilos-1].setMaximum(100);
            pbsConc[nHilos-1].setMinimum(0);
            pbsConc[nHilos-1].setValue(0);
            labelsHiloConc[nHilos-1].setText("Hilo " + (nHilos-1));
            labelsHiloConc[nHilos-1].setBackground(new
java.awt.Color(200,200,200));
        }
    });

    //Decrementar hilos
    btnRemoverHilo.addActionListener((ActionEvent e) -> {
        if ( nHilos > 1) {
            pbsConc[nHilos-1].setVisible(false);
            labelsHiloConc[nHilos-1].setVisible(false);
            nHilos -= 1;
            labelNumeroHilos.setText("Numero de hilos: " + nHilos);
        }
    });

    //Ver resultado secuencial
    btnResultadoSec.addActionListener((ActionEvent e) -> {
        Dialog ventana = new Dialog(true, resultSec, "Resultado
algoritmo secuencial");
        ventana.setVisible(true);
    });

    //Ver resultado concurrente
    btnResultadoConc.addActionListener((ActionEvent e) -> {
        Dialog ventana = new Dialog(true, resultConc, "Resultado
algoritmo concurrente");
        ventana.setVisible(true);
    });

    //Ver matriz A
    imgM1.addActionListener((ActionEvent e) -> {
        Dialog ventana = new Dialog(true, m1, "Matriz A");
    });

```

```

        ventana.setVisible(true);
    });

    //Ver matriz B
    imgM2.addActionListener((ActionEvent e) -> {
        Dialog ventana = new Dialog(true, m2, "Matriz B");
        ventana.setVisible(true);
    });
}

public void crearComponentes(){
    //construct components
    labelTitulo = new JLabel();

    imgM1 = new JButton();
    imgM2 = new JButton();

    labelM1 = new JLabel();
    labelM2 = new JLabel();

    btnGenerarM1 = new JButton();
    btnGenerarM2 = new JButton();

    btnCalculoSecuencial = new JButton();
    btnCalculoConcurrente = new JButton();
    btnCalculoSyc = new JButton();

    labelSecuencial = new JLabel();
    jLabel1 = new JLabel();

    imgHiloSec = new JLabel();
    labelHiloSec = new JLabel();
    pbSec = new JProgressBar();
    labelResSec = new JLabel();
    btnResultadoSec = new JButton();

    labelNumeroHilos = new JLabel();
    btnRemoverHilo = new JButton();
    btnAgregarHilo = new JButton();
    labelResConc = new JLabel();
    btnResultadoConc = new JButton();

    for (int i = 0; i < maxHilos ; i++ ) {
        pbsConc[i] = new JProgressBar();
        labelsHiloConc[i] = new JLabel();
    }

    //adjust size and set layout

```

```

        setPreferredSize (new Dimension (944, 574));
        setLayout (null);

        //add components
        labelTitulo.setBackground(new Color(242, 242, 0));
        labelTitulo.setFont(new Font("Segoe UI", 0, 24));
        labelTitulo.setHorizontalAlignment(SwingConstants.CENTER);
        labelTitulo.setText("Multiplicacion de matrices");
        add(labelTitulo);

        imgM1.setIcon(new
ImageIcon(getClass().getResource("/images/matrix.png")));
        imgM1.setBorderPainted( false );
        imgM1.setOpaque(false);
        imgM1.setContentAreaFilled(false);
        add(imgM1);
        imgM2.setIcon(new
ImageIcon(getClass().getResource("/images/matrix.png")));
        imgM2.setBorderPainted( false );
        imgM2.setOpaque(false);
        imgM2.setContentAreaFilled(false);
        add(imgM2);

        labelM1.setHorizontalAlignment(SwingConstants.CENTER);
        labelM1.setFont(new Font("Segoe UI", 0, 12));
        labelM1.setText("No hay ninguna matriz");
        add(labelM1);
        labelM2.setHorizontalAlignment(SwingConstants.CENTER);
        labelM2.setFont(new Font("Segoe UI", 0, 12));
        labelM2.setText("No hay ninguna matriz");
        add(labelM2);

        btnGenerarM1.setIcon(new
ImageIcon(getClass().getResource("/images/dice-game-icon.png")));
        btnGenerarM1.setText("Generar matriz");
        add(btnGenerarM1);
        btnGenerarM2.setIcon(new
ImageIcon(getClass().getResource("/images/dice-game-icon.png")));
        btnGenerarM2.setText("Generar matriz");
        add(btnGenerarM2);

        btnCalculoSecuencial.setText("Calculo secuencial");
        add(btnCalculoSecuencial);

        btnCalculoConcurrente.setText("Calculo concurrente");
        add(btnCalculoConcurrente);

        btnCalculoSyC.setText("Calculo secuencial y concurrente");
        add(btnCalculoSyC);

```

```

        labelSecuencial.setBackground(new Color(242, 242, 0));
        labelSecuencial.setFont(new Font("Segoe UI", 0, 18));
        labelSecuencial.setHorizontalAlignment(SwingConstants.CENTER);
        labelSecuencial.setText("Calculo secuencial");
        add(labelSecuencial);

        jLabel1.setBackground(new Color(242, 242, 0));
        jLabel1.setFont(new Font("Segoe UI", 0, 18));
        jLabel1.setHorizontalAlignment(SwingConstants.CENTER);
        jLabel1.setText("Calculo concurrente");
        add(jLabel1);

        //--- Secuencial ---//

        imgHiloSec.setIcon(new
ImageIcon(getClass().getResource("/images/thread1.png")));
        add(imgHiloSec);

        labelHiloSec.setHorizontalAlignment(SwingConstants.CENTER);
        labelHiloSec.setText("Estado hilo");
        labelHiloSec.setOpaque(true);
        labelHiloSec.setBackground(new java.awt.Color(200,200,200));
        add(labelHiloSec);

        pbSec.setStringPainted(true);
        add(pbSec);

        labelResSec.setFont(new Font("Segoe UI", 0, 16));
        labelResSec.setText("Resultado:");
        labelResSec.setHorizontalAlignment(SwingConstants.CENTER);
        add(labelResSec);

        btnResultadoSec.setText("Ver resultado algoritmo secuencial");
        btnResultadoSec.setIcon(new
ImageIcon(getClass().getResource("/images/result.png")));
        add(btnResultadoSec);

        //--- Concurrente ---//

        labelNumeroHilos.setFont(new Font("Segoe UI", 0, 16));
        labelNumeroHilos.setText("Numero de hilos: " + nHilos);
        add(labelNumeroHilos);

        btnRemoverHilo.setIcon(new
ImageIcon(getClass().getResource("/images/minus.png")));
        add(btnRemoverHilo);

```

```

        btnAgregarHilo.setIcon(new
ImageIcon(getClass().getResource("/images/add.png")));
        add(btnAgregarHilo);

        labelResConc.setFont(new Font("Segoe UI", 0, 16));
        labelResConc.setText("Resultado:");
        labelResConc.setHorizontalAlignment(SwingConstants.CENTER);
        add(labelResConc);

        btnResultadoConc.setText("Ver resultado algoritmo concurrente");
        btnResultadoConc.setIcon(new
ImageIcon(getClass().getResource("/images/result.png")));
        add(btnResultadoConc);

        for (int i = 0; i < maxHilos ; i++ ) {
            pbsConc[i].setStringPainted(true);
            add(pbsConc[i]);
            labelsHiloConc[i].setIcon(new
ImageIcon(getClass().getResource("/images/thread2.png")));
            labelsHiloConc[i].setText("Hilo " + i);
            labelsHiloConc[i].setOpaque(true);
            labelsHiloConc[i].setBackground(new
java.awt.Color(200,200,200));
            add(labelsHiloConc[i]);
        }

        //set component bounds

        labelTitulo.setBounds(360, 10, 290, 35);

        imgM1.setBounds(250, 70, 64, 64);
        imgM2.setBounds(530, 70, 64, 64);

        labelM1.setBounds(330, 70, 121, 60);
        labelM2.setBounds(610, 70, 121, 60);

        btnGenerarM1.setBounds(255, 140, 195, 28);
        btnGenerarM2.setBounds(535, 140, 195, 28);

        btnCalculoSecuencial.setBounds(255, 220, 475, 23);
        btnCalculoConcurrente.setBounds(255, 250, 475, 23);
        btnCalculoSyc.setBounds(255, 280, 475, 23);

        labelSecuencial.setBounds(140, 330, 260, 20);
        jLabel1.setBounds(570, 330, 260, 20);

        imgHiloSec.setBounds(200, 370, 128, 128);

```



```

        labelHiloSec.setBounds(140, 500, 250, 30);
        pbSec.setBounds(140, 550, 250, 50);
        labelResSec.setBounds(90, 740, 350, 22);
        btnResultadoSec.setBounds(90, 780, 350, 50);

        labelNumeroHilos.setBounds(560, 370, 180, 22);
        btnRemoverHilo.setBounds(750, 370, 30, 30);
        btnAgregarHilo.setBounds(790, 370, 30, 30);
        labelResConc.setBounds(520, 740, 350, 22);
        btnResultadoConc.setBounds(520, 780, 350, 50);

        for (int i = 0; i < maxHilos/2 ; i++ ) {
            pbsConc[i*2].setBounds(470, 420+i*30, 145, 25);
            labelsHiloConc[i*2].setBounds(620, 420+i*30, 100, 25);
        }
        for (int i = 0; i < maxHilos/2 ; i++ ) {
            pbsConc[i*2+1].setBounds(730, 420+i*30, 145, 25);
            labelsHiloConc[i*2+1].setBounds(880, 420+i*30, 100, 25);
        }
    }

    public void limpiar(){
        pbSec.setMaximum(100);
        pbSec.setValue(0);
        labelHiloSec.setText("Estado hilo");
        labelHiloSec.setBackground(new java.awt.Color(200,200,200));

        for (int i = 0; i < maxHilos ; i++ ) {
            pbsConc[i].setMaximum(100);
            pbsConc[i].setMinimum(0);
            pbsConc[i].setValue(0);
            labelsHiloConc[i].setText("Hilo " + i);
            labelsHiloConc[i].setBackground(new
java.awt.Color(200,200,200));
        }

    }

    public void inicio(){
        if (m1 != null){
            rowsM1 = m1.length;
            colsM1 = m1[0].length;
            String str = "<html><body><div style=\"text-align:center;
width:80px;\">>[" + rowsM1 + " X " + colsM1 + " ]";
            int aux = m1.length;
            if (m1.length > 3)
                aux = 3;
            for (int i = 0; i < aux ; i++ ) {

```

```

        str += "<p style=\"white-space:nowrap;text-align:center;
width:80px;\">>" + arrToStr(m1[i]) + "</p>";
    }

    str += "</body></html>";
    labelM1.setText(str);
}else
    labelM1.setText("No hay ninguna matriz");

    if (m2 != null){
        rowsM2 = m2.length;
        colsM2 = m2[0].length;
        String str = "<html><body><div style=\"text-align:center;
width:80px;\">>[" + rowsM2 + " X " + colsM2 + "]\>";
        int aux = m2.length;
        if (m2.length > 3)
            aux = 3;
        for (int i = 0; i < aux ; i++ ) {
            str += "<p style=\"white-space:nowrap;text-align:center;
width:80px;\">>" + arrToStr(m2[i]) + "</p>";
        }

        str += "</body></html>";
        labelM2.setText(str);
    }else
        labelM2.setText("No hay ninguna matriz");

    for (int i = 0; i < maxHilos ; i++ ) {
        pbsConc[i].setVisible(false);
        labelsHiloConc[i].setVisible(false);
    }

    for (int i = 0; i < nHilos ; i++ ) {
        pbsConc[i].setVisible(true);
        labelsHiloConc[i].setVisible(true);
    }
}

public static String arrToStr(int[] arr){
    String str = "";
    for (int i = 0; i < arr.length ; i++ ) {
        str += arr[i] + " ";
    }
    return str;
}

```

```

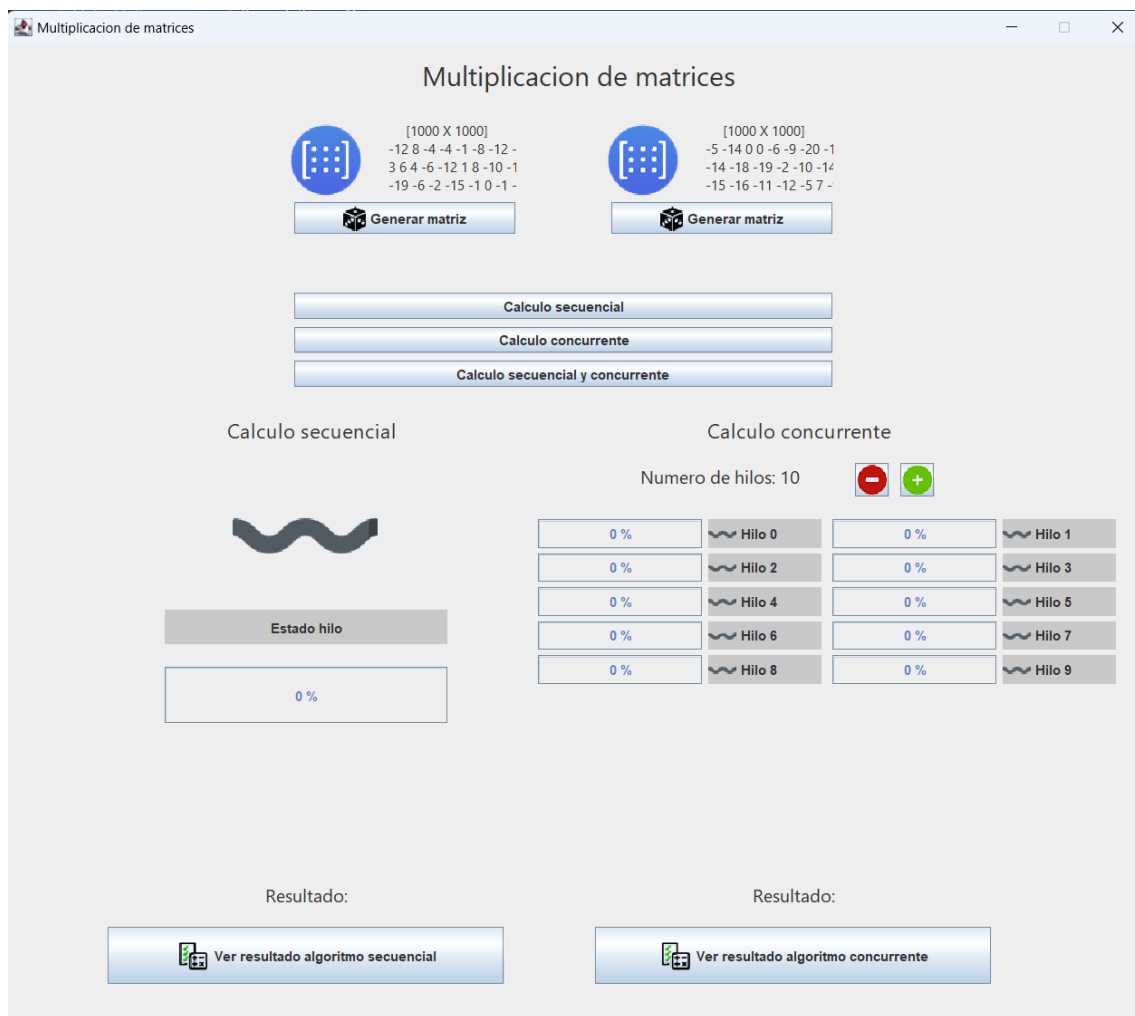
    }

    public class ThreadMultSecuencial extends Thread {
        long start, time;
        @Override
        public void run(){
            start = System.nanoTime();
            resultSec = MatrixMult.multiply(m1,m2,pbSec, labelHiloSec);
            time = System.nanoTime() - start;
            labelResSec.setText("Resultado despues de " + (double) time /
1_000_000 + "ms");
        }
    }

    public class ThreadMultConcurrente extends Thread {
        long start, time;
        @Override
        public void run(){
            start = System.nanoTime();
            resultConc =
MatrixMult.multiplyConcurrent(m1,m2,nHilos,pbsConc,labelsHiloConc);
            time = System.nanoTime() - start;
            labelResConc.setText("Resultado despues de " + (double) time
/ 1_000_000 + "ms");
        }
    }
}

```

Se ejecuta el código de Proyecto2 y se tiene la siguiente interfaz:




La interfaz muestra la interacción y el control de la aplicación, número de hilos, estado de los hilos, estado en tiempo real del procesamiento (secuencial o concurrente), características del problema a resolver.

Se resuelve el mismo problema en igualdad de condiciones con las mismas características tanto en secuencial como concurrente.

La información de los tiempos en los que se han tardado ambos algoritmos se muestra claramente en la parte inferior de la interfaz.


Y el objetivo más importante es que se muestre cómo es que al ejecutar la tarea de manera concurrente es más efectivo que el secuencial en la mayoría de los casos.


Primero se hará la prueba con pocas filas y se irá incrementando la cantidad de números que la matriz tenga.



[10 X 10]


-17 -4 -3 8 -8 -16 -7 -
-7 6 7 8 6 4 -10 -10 -1
-18 -17 -15 0 -19 -15

 Generar matriz




[10 X 10]

-2 1 0 -11 -17 -17 -6 -
3 -2 6 -3 -17 2 4 -5 -7
-1 6 -14 7 -15 2 -20 -!

 Generar matriz

Se inicia con una matriz de 10 x 10 y se irá incrementando con cada instancia.

Calculo secuencial















Finalizado

100 %

Resultado despues de 0.1396ms


Calculo concurrente

Numero de hilos: 10  

100 %	 Finalizado	100 %	 Finalizado
100 %	 Finalizado	100 %	 Finalizado
100 %	 Finalizado	100 %	 Finalizado
100 %	 Finalizado	100 %	 Finalizado
100 %	 Finalizado	100 %	 Finalizado


Resultado despues de 3.1428ms


Como se puede ver, la ejecución en secuencial es más rápida que la concurrente. Como se había mencionado anteriormente, al intentar hacerlo concurrente, se tarda mucho en dividir el problema y asignarlo a hilos diferentes para llevar a cabo la tarea, pero este resultado va a ir cambiando a medida que la matriz se hace más grande.



[230 X 230]


-17 -4 -13 -10 -1 -5 -
-1 -11 3 -4 1 -19 6 2 -
4 -17 -10 7 -9 -14 -7 -

 Generar matriz



[230 X 230]

-2 -4 -12 -11 -19 -18
-12 -19 -1 -3 -17 -15
5 -5 -8 -19 5 0 5 -11 -


 Generar matriz

Calculo secuencial

Calculo concurrente

Calculo secuencial y concurrente

Calculo secuencial















Finalizado

100 %

Resultado despues de 1.0207ms

Calculo concurrente

Numero de hilos: 10  

100 %	 Finalizado	100 %	 Finalizado
100 %	 Finalizado	100 %	 Finalizado
100 %	 Finalizado	100 %	 Finalizado
100 %	 Finalizado	100 %	 Finalizado
100 %	 Finalizado	100 %	 Finalizado

Resultado despues de 0.9093ms

Más o menos al multiplicar matrices de 230×230 cada matriz, ambas ejecuciones se resuelven en la misma cantidad de tiempo. A partir de este punto, la ejecución concurrente será más rápida que la secuencial.

[500 X 500]

-11 8 6 -7 0 -5 -6 -20
-14 -3 0 -3 6 -12 -6 -7
6 -4 -11 -17 -5 4 -18 -

Generar matriz

[500 X 500]

5 2 -11 -14 -4 0 -15 -
0 -7 -11 -16 -15 -11 -
-2 -12 4 0 -11 6 -12 6

Generar matriz

Calculo secuencial

Calculo concurrente

Calculo secuencial y concurrente

Calculo secuencial

Finalizado

100 %

Calculo concurrente

Numero de hilos: 10

-

+

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

Resultado despues de 9.4937ms

Resultado despues de 4.3142ms

[1000 X 1000]

4 1 -19 -7 -10 1 2 3 -2
7 -8 -9 7 -4 -11 -8 -12
5 -15 -14 8 -8 -18 -8 -

Generar matriz

[1000 X 1000]

-20 -4 -3 4 2 8 -11 -9
8 6 -7 -20 2 -18 -15 0
-13 -5 -12 -15 -5 -4 -;

Generar matriz

Calculo secuencial

Calculo concurrente

Calculo secuencial y concurrente

Calculo secuencial

Finalizado

100 %

Calculo concurrente

Numero de hilos: 10

-

+

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

100 %

Finalizado

Resultado despues de 66.3916ms

Resultado despues de 21.761ms

Conclusiones:

La concurrencia en computación permite optimizar el uso de los recursos disponibles al dividir tareas complejas en subtareas más pequeñas, logrando una mayor eficiencia y reduciendo los tiempos de procesamiento en tareas con altas demandas computacionales. Sin embargo, para tareas pequeñas, la concurrencia puede ser menos eficiente que la ejecución secuencial, ya que el costo de dividir y gestionar las subtareas puede superar el beneficio obtenido. Por lo tanto, la implementación de concurrencia debe evaluarse cuidadosamente según la naturaleza y tamaño de la tarea para garantizar un uso óptimo del hardware.