



Centro de Enseñanza Técnica Industrial

Desarrollo de Software

Proyecto del tercer parcial

Jesús Alberto Aréchiga Carrillo

22310439 5N

Profesor

José Francisco Pérez Reyes

Diciembre 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.

En el caso del paralelismo, pasa exactamente lo mismo, la división de problemas y la comunicación toman tiempo, de tal manera que es necesario plantear una manera de reducir esos tiempos. Para este proyecto, se va a reducir el tiempo del paralelismo comunicando las matrices desde un inicio, de tal manera que la única cosa que queda por comunicar es que se empiece el proceso.

Desarrollo

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

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 Server.MatrixMultiplierInterface;
import java.awt.*;
import java.awt.event.*;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.util.ArrayList;

```

```
import java.util.List;
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(1600,900);
        frame.setLocation(250, 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;

    private List<String> servidoresConectados = new ArrayList<>();

    //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 btnCalculoParalelo;
    private JButton btnCalculoSyc;
    private JButton btnCalculoSCP;

    private JLabel tituloSecuencial;
    private JLabel tituloConcurrente;
    private JLabel tituloParalelo;

    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];

private JLabel imgHiloPar;
private JLabel labelsHiloPar;
private JLabel labelResPar;
private JButton btnResultadoPar;

private JPanel servidoresDisponibles;
private JButton btnAgregarServidor;
private JButton btnRemoverServidor;
private JLabel[] ipServidores = new JLabel[3];
private JLabel[] serverIcons = new JLabel[3];
private JLabel[] disponibleIcons = new JLabel[3];

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

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

public ThreadMultSecuencial hiloSecuencial;
public ThreadMultConcurrente hiloConcurrente;
public ThreadMultParalelo hiloParalelo;

public Proyecto2(){

    nHilos = 10;
    nServers = 0;

```

```

        crearComponentes();

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

        inicio();

        //verificarServidores();

        //Generar M1
        btnGenerarM1.addActionListener((ActionEvent e) -> {
            if (servidoresConectados.isEmpty()) {
                JOptionPane.showMessageDialog(null, "No hay servidores disponibles para realizar el cálculo.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }
            labelM1.setText("Generando...");
            String inp1 = JOptionPane.showInputDialog(null, "Matriz A\nIngresa el número de filas: ");
            String inp2 = JOptionPane.showInputDialog(null, "Matriz A\nIngresa el número 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 válidos", "Error", JOptionPane.ERROR_MESSAGE);
            }

            if (m1 != null) {
                // Actualizar la etiqueta con la representación de la matriz generada
                rowsM1 = m1.length;
                colsM1 = m1[0].length;
                String str = "<html><body><div style=\"text-align:center; width:80px;\">[" + rowsM1 + " X " + colsM1 + "]</div>";
                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);
            }
        });
    }
}

```

```

        //Enviar las partes de la matriz A a los servidores
conectados
        enviarPartesMatrizA(m1, servidoresConectados);
    } else {
        labelM1.setText("No hay ninguna matriz");
    }
});

//Generar M2
btnGenerarM2.addActionListener((ActionEvent e) -> {
    if (servidoresConectados.isEmpty()) {
        JOptionPane.showMessageDialog(null, "No hay servidores
disponibles para realizar el cálculo.", "Error",
JOptionPane.ERROR_MESSAGE);
        return;
    }
    labelM2.setText("Generando...");
    String inp1 = JOptionPane.showInputDialog(null, "Matriz
B\nIngresa el número de filas: ");
    String inp2 = JOptionPane.showInputDialog(null, "Matriz
B\nIngresa el número 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
válidos", "Error", JOptionPane.ERROR_MESSAGE);
    }

    if (m2 != null) {
        // Actualizar la etiqueta con la representación de la
matriz generada
        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);

        // Enviar la matriz B completa a los servidores
conectados
    }
});

```

```

        enviarMatrizBServidores(m2, servidoresConectados);
    } 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 paralelo
btnCalculoParalelo.addActionListener((ActionEvent e) -> {
    if (servidoresConectados.isEmpty()) {
        JOptionPane.showMessageDialog(null, "No hay servidores
disponibles para realizar el cálculo.", "Error",
JOptionPane.ERROR_MESSAGE);
        return;
    }

    hiloParalelo = new ThreadMultParalelo(this);
    hiloParalelo.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();
});

//Calculo Sec-Conc-Par
btnCalculoSCP.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;
    }
    if(nServers == 0){
        JOptionPane.showMessageDialog(null, "No hay servidores
disponibles", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    limpiar();
    hiloSecuencial = new ThreadMultSecuencial();
    hiloSecuencial.start();
    hiloConcurrente = new ThreadMultConcurrente();
    hiloConcurrente.start();
    hiloParalelo = new ThreadMultParalelo(this);
    hiloParalelo.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);
    });

    btnResultadoPar.addActionListener((ActionEvent e) -> {
        Dialog ventana = new Dialog(true, resultPar, "Resultado
algoritmo paralelo");
        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);
    });

    btnAgregarServidor.addActionListener((ActionEvent e) -> {
        if(nServers < 3){

```

```

        String ipServidor = JOptionPane.showInputDialog(null,
        "Ingrese la IP del servidor:", "Agregar Servidor",
        JOptionPane.PLAIN_MESSAGE);
        if (ipServidor == null || ipServidor.isEmpty()) {
            JOptionPane.showMessageDialog(null, "No se ingresó
una IP válida.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        } else if (servidoresConectados.contains(ipServidor)) {
            JOptionPane.showMessageDialog(null, "Ese servidor ya
se esta usando", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        try {
            String serverName = "rmi://" + ipServidor +
            ":1099/MatrixMultiplier";

            // Intenta localizar el servidor y probar la conexión
            MatrixMultiplierInterface multiplier =
            (MatrixMultiplierInterface) Naming.lookup(serverName);
            System.out.println("isReady: " +
            multiplier.isReady());
            if (multiplier.isReady()) {
                servidoresConectados.add(ipServidor);
                nServers++;
                ipServidores[nServers-1].setText("IP: " +
            ipServidor);

                serverIcons[nServers-1].setVisible(true);
                disponibleIcons[nServers-1].setVisible(true);
                ipServidores[nServers-1].setVisible(true);
                JOptionPane.showMessageDialog(null, "Servidor
conectado correctamente.", "Éxito", JOptionPane.INFORMATION_MESSAGE);
            }
        } catch (HeadlessException | MalformedURLException |
        NotBoundException | RemoteException ex) {
            JOptionPane.showMessageDialog(null, "No se pudo
conectar al servidor: " + ipServidor, "Error",
            JOptionPane.ERROR_MESSAGE);
        }
        } else {
            JOptionPane.showMessageDialog(null, "Ya no se pueden
agregar más servidores", "Advertencia", JOptionPane.WARNING_MESSAGE);
        }

    });

    btnRemoverServidor.addActionListener((ActionEvent e) -> {
        if (nServers > 0) {

```

```

        // Crear una lista desplegable con las direcciones IP de
los servidores
        String[] opciones =
servidoresConectados.toArray(String[]::new);
        String servidorAEliminar = (String)
JOptionPane.showInputDialog(
            null,
            "Selecciona el servidor a eliminar:",
            "Eliminar Servidor",
            JOptionPane.QUESTION_MESSAGE,
            null,
            opciones,
            opciones[0]
        );

        // Verificar que el usuario seleccionó un servidor
        if (servidorAEliminar != null) {
            // Confirmar la eliminación
            int confirm = JOptionPane.showConfirmDialog(
                null,
                "¿Estás seguro de que deseas eliminar el servidor
" + servidorAEliminar + "?",
                "Confirmar eliminación",
                JOptionPane.YES_NO_OPTION,
                JOptionPane.QUESTION_MESSAGE
            );

            if (confirm == JOptionPane.YES_OPTION) {
                // Encontrar el índice del servidor seleccionado
                int index =
servidoresConectados.indexOf(servidorAEliminar);

                // Eliminar el servidor de la lista
                servidoresConectados.remove(index);

                // Actualizar la interfaz gráfica
                for (int i = 0; i < servidoresConectados.size();
i++) {
                    ipServidores[i].setText(servidoresConectados.
get(i));

                    serverIcons[i].setVisible(true);
                    disponibleIcons[i].setVisible(true);
                    ipServidores[i].setVisible(true);
                }

                // Ocultar elementos sobrantes si quedan menos
servidores
                for (int i = servidoresConectados.size(); i <
ipServidores.length; i++) {

```



```

        ipServidores[i].setVisible(false);
        serverIcons[i].setVisible(false);
        disponibleIcons[i].setVisible(false);
    }

    nServers--;
}
}
} else {
    JOptionPane.showMessageDialog(null, "No hay servidores
para eliminar.", "Advertencia", JOptionPane.WARNING_MESSAGE);
}
});
}

private void enviarMatrizBServidores(int[][] matriz, List<String>
servidoresConectados) {
    if (servidoresConectados.isEmpty()) {
        System.err.println("No hay servidores conectados para enviar
la matriz.");
        return;
    }

    for (String servidor : servidoresConectados) {
        try {
            String serverName = "rmi://" + servidor +
":1099/MatrixMultiplier";
            MatrixMultiplierInterface multiplier =
(MatrixMultiplierInterface) Naming.lookup(serverName);

            // Enviar la matriz B
            multiplier.recibirMatrizB(matriz);

            System.out.println("Matriz B enviada al servidor: " +
servidor);
        } catch (MalformedURLException | NotBoundException |
RemoteException e) {
            System.err.println("Error al enviar la matriz B al
servidor " + servidor + ": " + e.getMessage());
        }
    }
}

private void enviarPartesMatrizA(int[][] matriz, List<String>
servidoresConectados) {
    if (servidoresConectados.isEmpty()) {
        System.err.println("No hay servidores conectados para enviar
las partes de la matriz A.");
        return;
    }
}

```

```

    }

    int totalRows = matriz.length;
    int blockSize = (int) Math.ceil((double) totalRows /
servidoresConectados.size());

    for (int i = 0; i < servidoresConectados.size(); i++) {
        int startRow = i * blockSize;
        int endRow = Math.min(startRow + blockSize, totalRows);
        int[][] subMatrizA = new int[endRow -
startRow][matriz[0].length];

        // Copiar las filas correspondientes
        for (int r = startRow; r < endRow; r++) {
            subMatrizA[r - startRow] = matriz[r];
        }

        try {
            String serverName = "rmi://" +
servidoresConectados.get(i) + ":1099/MatrixMultiplier";
            MatrixMultiplierInterface multiplier =
(MatrixMultiplierInterface) Naming.lookup(serverName);

            // Enviar la parte correspondiente de la matriz A
            multiplier.recibirParteMatrizA(subMatrizA);

            System.out.println("Parte de la matriz A enviada al
servidor: " + servidoresConectados.get(i));
        } catch (MalformedURLException | NotBoundException |
RemoteException e) {
            System.err.println("Error al enviar la parte de la matriz
A al servidor " + servidoresConectados.get(i) + ": " + e.getMessage());
        }
    }
}

/*private void verificarServidores() {
    ScheduledExecutorService scheduler = (ScheduledExecutorService)
Executors.newScheduledThreadPool(1);
    Runnable task = () -> {
        for (String ip : new ArrayList<>(servidoresConectados)) {
            try {
                String serverName = "rmi://" + ip +
":1099/MatrixMultiplier";
                MatrixMultiplierInterface multiplier =
(MatrixMultiplierInterface) Naming.lookup(serverName);
                if (multiplier.isReady()) {

```

```

                disponibleIcons[servidoresConectados.indexOf(ip)]
                .setIcon(new
                ImageIcon(Proyecto2.this.getClass().getResource("/images/yes.png")));
            }
        } catch (MalformedURLException | NotBoundException |
        RemoteException e) {
            disponibleIcons[servidoresConectados.indexOf(ip)].set
            Icon(new
            ImageIcon(Proyecto2.this.getClass().getResource("/images/no.png")));
            servidoresConectados.remove(ip);
        }
    }
};
scheduler.scheduleAtFixedRate(task, 0, 5, TimeUnit.SECONDS);
}*/

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();
    btnCalculoParalelo = new JButton();
    btnCalculoSyc = new JButton();
    btnCalculoSCP = new JButton();

    tituloSecuencial = new JLabel();
    tituloConcurrente = new JLabel();
    tituloParalelo = 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();

        imgHiloPar = new JLabel();
        labelsHiloPar = new JLabel();
        labelResPar = new JLabel();
        btnResultadoPar = new JButton();

        servidoresDisponibles = new JPanel();
        btnAgregarServidor = new JButton();
        btnRemoverServidor = new JButton();

        for (int i = 0; i < 3 ; i++ ) {
            serverIcons[i] = new JLabel();
            disponibleIcons[i] = new JLabel();
            ipServidores[i] = new JLabel();
        }

        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);

btnCalculoParalelo.setText("Calculo paralelo");
add(btnCalculoParalelo);

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

btnCalculoSCP.setText("Calculo secuencial, concurrente y
paralelo");
add(btnCalculoSCP);

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

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

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

```

servidoresDisponibles.setLayout(null); // Layout personalizado
servidoresDisponibles.setBorder(BorderFactory.createTitledBorder(
"Servidores disponibles: " + nServers));
add(servidoresDisponibles);

btnAgregarServidor.setIcon(new
ImageIcon(getClass().getResource("/images/add.png")));
servidoresDisponibles.add(btnAgregarServidor);
btnRemoverServidor.setIcon(new
ImageIcon(getClass().getResource("/images/minus.png")));
servidoresDisponibles.add(btnRemoverServidor);

    //--- 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]);
        }

        //--- Paralelo ---//
        imgHiloPar.setIcon(new
ImageIcon(getClass().getResource("/images/thread1.png")));
        add(imgHiloPar);

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

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

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

        for (int i = 0; i < 3 ; i++ ) {

```

```

        serverIcons[i].setIcon(new
ImageIcon(getClass().getResource("/images/server.png")));
        servidoresDisponibles.add(serverIcons[i]);
        disponibleIcons[i].setIcon(new
ImageIcon(getClass().getResource("/images/yes.png")));
        servidoresDisponibles.add(disponibleIcons[i]);
        ipServidores[i].setText("");
        ipServidores[i].setHorizontalAlignment(SwingConstants.CENTER)
;

        servidoresDisponibles.add(ipServidores[i]);
        serverIcons[i].setOpaque(false);
        disponibleIcons[i].setOpaque(false);
        serverIcons[i].setVisible(false);
        disponibleIcons[i].setVisible(false);
        ipServidores[i].setVisible(false);
    }
    //set component bounds

    labelTitulo.setBounds(655, 10, 290, 35);

    imgM1.setBounds(558, 70, 64, 64);
    imgM2.setBounds(838, 70, 64, 64);

    labelM1.setBounds(638, 70, 121, 60);
    labelM2.setBounds(918, 70, 121, 60);

    btnGenerarM1.setBounds(563, 140, 195, 28);
    btnGenerarM2.setBounds(843, 140, 195, 28);

    btnCalculoSecuencial.setBounds(563, 180, 475, 23);
    btnCalculoConcurrente.setBounds(563, 210, 475, 23);
    btnCalculoParalelo.setBounds(563, 240, 475, 23);
    btnCalculoSyc.setBounds(563, 270, 475, 23);
    btnCalculoSCP.setBounds(563, 300, 475, 23);

    tituloSecuencial.setBounds(140, 330, 260, 20);
    tituloConcurrente.setBounds(670, 330, 260, 20);
    tituloParalelo.setBounds(1200, 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(660, 370, 180, 22);
    btnRemoverHilo.setBounds(850, 370, 30, 30);
    btnAgregarHilo.setBounds(890, 370, 30, 30);
    labelResConc.setBounds(625, 740, 350, 22);

```



```

        btnResultadoConc.setBounds(620, 780, 350, 50);

        for (int i = 0; i < 3; i++) {
            serverIcons[i].setBounds(100 + i * 100, 20, 30, 30);
            disponibleIcons[i].setBounds(100 + i * 100, 35, 20, 20);
            ipServidores[i].setBounds(65 + i * 100, 60, 100, 20);
        }

        for (int i = 0; i < maxHilos/2 ; i++ ) {
            pbsConc[i*2].setBounds(545, 420+i*30, 145, 25);
            labelsHiloConc[i*2].setBounds(695, 420+i*30, 100, 25);
        }
        for (int i = 0; i < maxHilos/2 ; i++ ) {
            pbsConc[i*2+1].setBounds(805, 420+i*30, 145, 25);
            labelsHiloConc[i*2+1].setBounds(955, 420+i*30, 100, 25);
        }

        imgHiloPar.setBounds(1266, 370, 128, 128);
        labelsHiloPar.setBounds(1205, 500, 250, 30);
        labelResPar.setBounds(1155, 740, 350, 22);
        btnResultadoPar.setBounds(1155, 780, 350, 50);

        servidoresDisponibles.setBounds(1130, 570, 400, 120);
        btnAgregarServidor.setBounds(50, 80, 30, 30);
        btnRemoverServidor.setBounds(10, 80, 30, 30);

    }

    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));
        }

        labelsHiloPar.setText("Estado hilo paralelo");
        labelsHiloPar.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 void setResPar(int[][] res){
    resultPar = res;
}

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");
    }
}

public class ThreadMultParalelo extends Thread {
    private int[][] resultadoFinal; // Variable para almacenar la
matriz final
    private final Object lock = new Object(); // Para sincronización
Proyecto2 proyecto2;
    long start, time;

    public ThreadMultParalelo(Proyecto2 proyecto2){
        this.proyecto2 = proyecto2;
    }
}

```

```

@Override
public void run() {
    try {
        int totalRows = rowsM1; // Número total de filas en la
matriz A

        int cols = colsM2; // Número de columnas en la matriz B
        resultadoFinal = new int[totalRows][cols]; // Inicializa
la matriz

        labelsHiloPar.setText("Procesando...");
        labelsHiloPar.setBackground(new Color(0, 255, 255));

        start = System.nanoTime();

        // Iniciar la multiplicación en cada servidor
        for (int i = 0; i < servidoresConectados.size(); i++) {
            String servidor = servidoresConectados.get(i);
            String serverName = "rmi://" + servidor +
":1099/MatrixMultiplier";

            try {
                MatrixMultiplierInterface multiplier =
(MatrixMultiplierInterface) Naming.lookup(serverName);

                // Iniciar la multiplicación en el servidor
                multiplier.multiplyPart();
                System.out.println("Cálculo iniciado en el
servidor " + servidor);

            } catch (MalformedURLException | NotBoundException |
RemoteException ex) {
                System.err.println("Error al iniciar cálculo en
el servidor " + servidor + ": " + ex.getMessage());
            }
        }

        // Marca el tiempo total y notifica que el cálculo ha
finalizado

        time = System.nanoTime() - start;

        labelsHiloPar.setText("Finalizado");
        labelsHiloPar.setBackground(new java.awt.Color(0, 200,
0));

        labelResPar.setText("Resultado después de " + (double)
time / 1_000_000 + "ms");

        // Recuperar los resultados parciales de cada servidor
        for (int i = 0; i < servidoresConectados.size(); i++) {
            String servidor = servidoresConectados.get(i);

```

```

        String serverName = "rmi://" + servidor +
":1099/MatrixMultiplier";

        try {
            MatrixMultiplierInterface multiplier =
(MatrixMultiplierInterface) Naming.lookup(serverName);

            // Recuperar el resultado parcial
            int[][] partialResult =
multiplier.enviarMatrizRes();

            // Agregar el resultado parcial a la matriz final
            synchronized (lock) {
                for (int r = 0; r < partialResult.length;
r++) {
                    for (int c = 0; c < cols; c++) {
                        resultadoFinal[r + (i *
partialResult.length)][c] += partialResult[r][c];
                    }
                }
            }

            System.out.println("Resultado parcial recibido
del servidor " + servidor);

        } catch (MalformedURLException | NotBoundException |
RemoteException ex) {
            System.err.println("Error al recuperar resultado
del servidor " + servidor + ": " + ex.getMessage());
        }

        proyecto2.setResPar(resultadoFinal);

    } catch (HeadlessException ex) {
        JOptionPane.showMessageDialog(null, "Error durante el
cálculo paralelo.", "Error", JOptionPane.ERROR_MESSAGE);
    }
}
}
}

```

MatrixMultiplierInterface.java:

```
package Server;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface MatrixMultiplierInterface extends Remote {
    void multiplyPart() throws RemoteException;
    void recibirMatrizB(int[][] matriz) throws RemoteException;
    void recibirParteMatrizA(int[][] matriz) throws RemoteException;
    int[][] enviarMatrizRes() throws RemoteException;
    boolean isReady() throws RemoteException;
}
```

MatrixMultiplierServer.java:

```
package Server;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.locks.ReentrantLock;

public class MatrixMultiplierServer extends UnicastRemoteObject
implements MatrixMultiplierInterface {

    private final ExecutorService executorService;
    private int[][] m1;
    private int[][] m2;
    private int[][] parteMatrizA;
    private int[][] res;
    private final ReentrantLock lock; // Para sincronización en acceso a
    la matriz

    protected MatrixMultiplierServer() throws RemoteException {
        super();
        // Crea un pool de hilos con un número fijo de hilos igual al
        número de núcleos de la CPU
        int numThreads = 20;
        executorService = Executors.newFixedThreadPool(numThreads);
        lock = new ReentrantLock();
    }

    @Override
    public void multiplyPart() throws RemoteException {
```

```

        if (parteMatrizA == null || m2 == null) {
            throw new RemoteException("No se han recibido las matrices
necesarias para realizar la multiplicación.");
        }

        int rows = parteMatrizA.length; // Número de filas de la parte de
la matriz A
        int cols = m2[0].length; // Número de columnas de la matriz B
        int commonDim = parteMatrizA[0].length; // Dimensión común entre
A y B
        res = new int[rows][cols]; // Matriz de resultado

        // Divide las filas en subtarear
        int blockSize = (int) Math.ceil((double) rows / 10);
        int numTasks = (int) Math.ceil((double) rows / blockSize);

        for (int i = 0; i < numTasks; i++) {
            final int blockStart = i * blockSize;
            final int blockEnd = Math.min(blockStart + blockSize, rows);

            executorService.submit(() -> {
                for (int r = blockStart; r < blockEnd; r++) {
                    for (int k = 0; k < commonDim; k++) {
                        int temp = parteMatrizA[r][k];
                        for (int c = 0; c < cols; c++) {
                            res[r][c] += temp * m2[k][c];
                        }
                    }
                }
            });
        }

        System.out.println("Multiplicación completada en este
servidor.");
    }

    @Override
    public boolean isReady() throws RemoteException {
        System.out.println("Conexión establecida.");
        return true;
    }

    @Override
    public void recibirParteMatrizA(int[][] matriz) throws
RemoteException {
        lock.lock();
        try {
            this.parteMatrizA = matriz;
        }
    }

```

```

        System.out.println("Parte de la matriz A recibida con " +
matriz.length + " filas y " + matriz[0].length + " columnas.");
    } finally {
        lock.unlock();
    }
}

@Override
public void recibirMatrizB(int[][] matriz) throws RemoteException {
    lock.lock(); // Bloquea el acceso concurrente
    try {
        this.m2 = matriz; // Almacena la matriz recibida
        System.out.println("Matriz B recibida con " + matriz.length +
" filas y " + matriz[0].length + " columnas.");
    } finally {
        lock.unlock(); // Libera el bloqueo
    }
}

public int[][] getMatrizA() throws RemoteException {
    lock.lock();
    try {
        if (m1 == null) {
            throw new RemoteException("No hay ninguna matriz
almacenada.");
        }
        return m1;
    } finally {
        lock.unlock();
    }
}

public int[][] getMatrizB() throws RemoteException {
    lock.lock();
    try {
        if (m2 == null) {
            throw new RemoteException("No hay ninguna matriz
almacenada.");
        }
        return m2;
    } finally {
        lock.unlock();
    }
}

@Override
public int[][] enviarMatrizRes() throws RemoteException {
    return res;
}

```

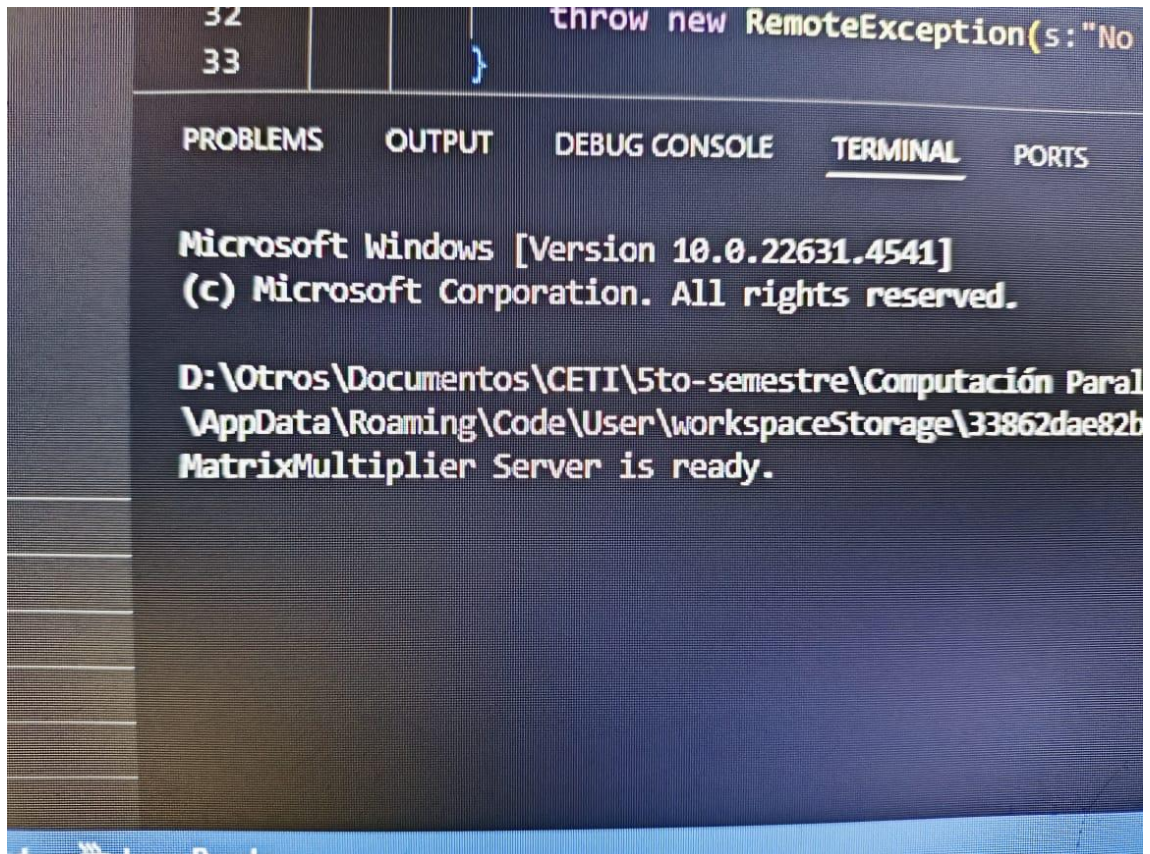


```

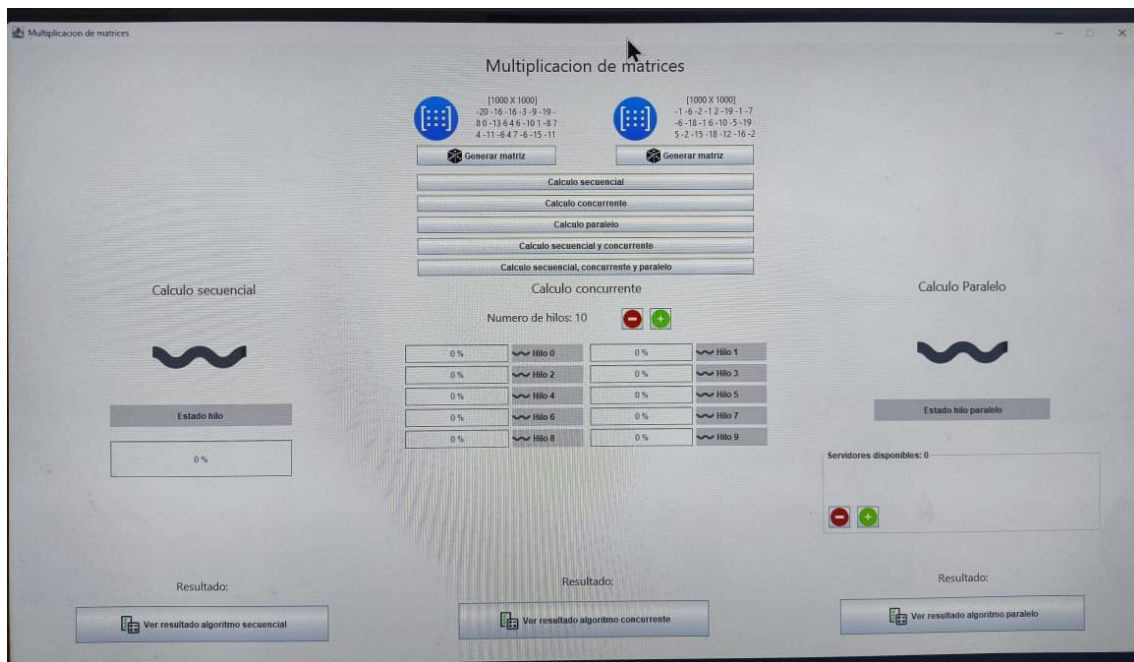
public static void main(String[] args) {
    try {
        MatrixMultiplierServer multiplier = new
MatrixMultiplierServer();
        LocateRegistry.createRegistry(1099);
        Naming.rebind("MatrixMultiplier", multiplier);
        System.out.println("MatrixMultiplier Server is ready.");
    } catch (MalformedURLException | RemoteException e) {
        System.out.println("Error en el servidor: " +
e.getMessage());
    }
}
}

```

Al ejecutarse el servidor sólo se tiene un mensaje diciendo que está listo:



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



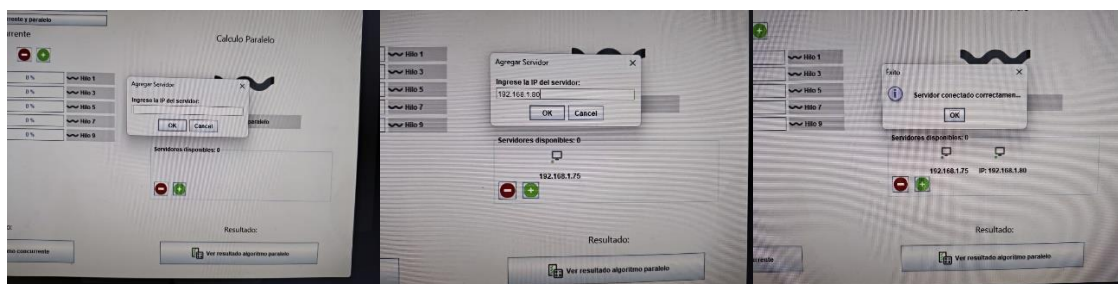
La interfaz muestra la interacción y el control de la aplicación, número de hilos, estado de los hilos, estado en tiempo real del procesamiento (secuencial, concurrente o paralelo), características del problema a resolver.

Se resuelve el mismo problema en igualdad de condiciones con las mismas características en los 3 casos (secuencial, concurrente y paralelo).


La información de los tiempos en los que se han tardado los 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, así como la tarea en paralelo sea más efectivo que el concurrente y el secuencial.


Para empezar, se tienen que agregar los servidores, ya que, al momento de generar las matrices, se envían a los servidores conectados y se necesitan los servidores que se vayan a utilizar ya conectados para este paso.




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

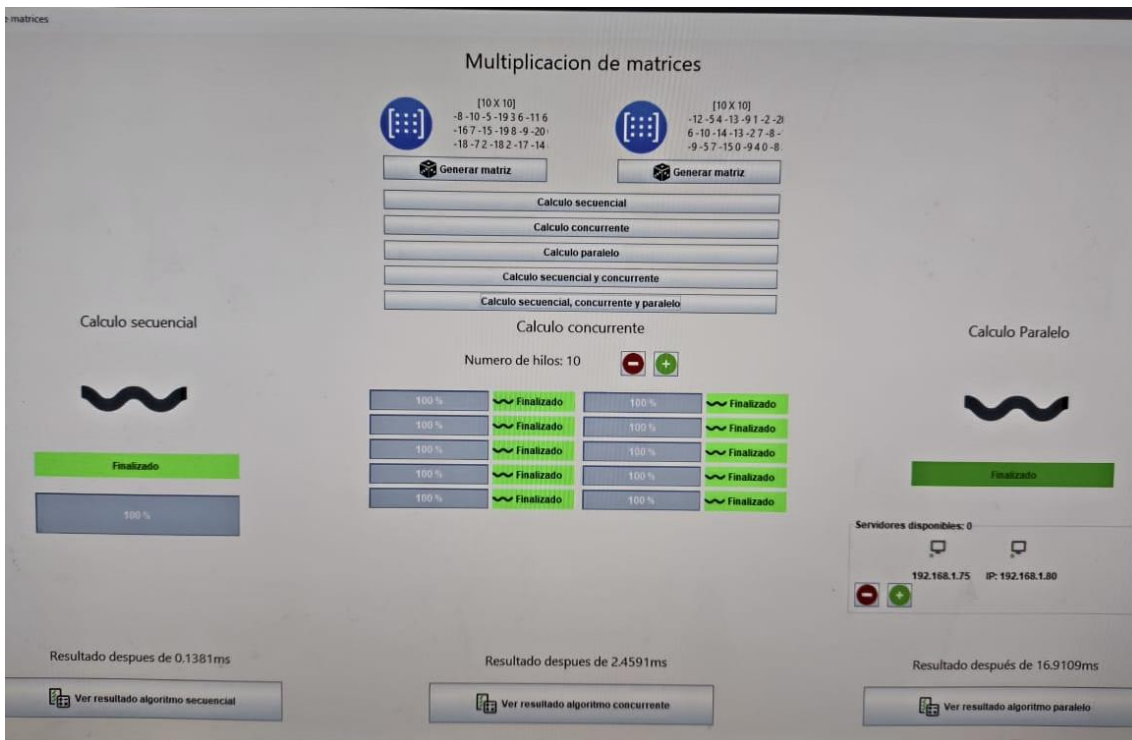
 **Generar matriz**



[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.



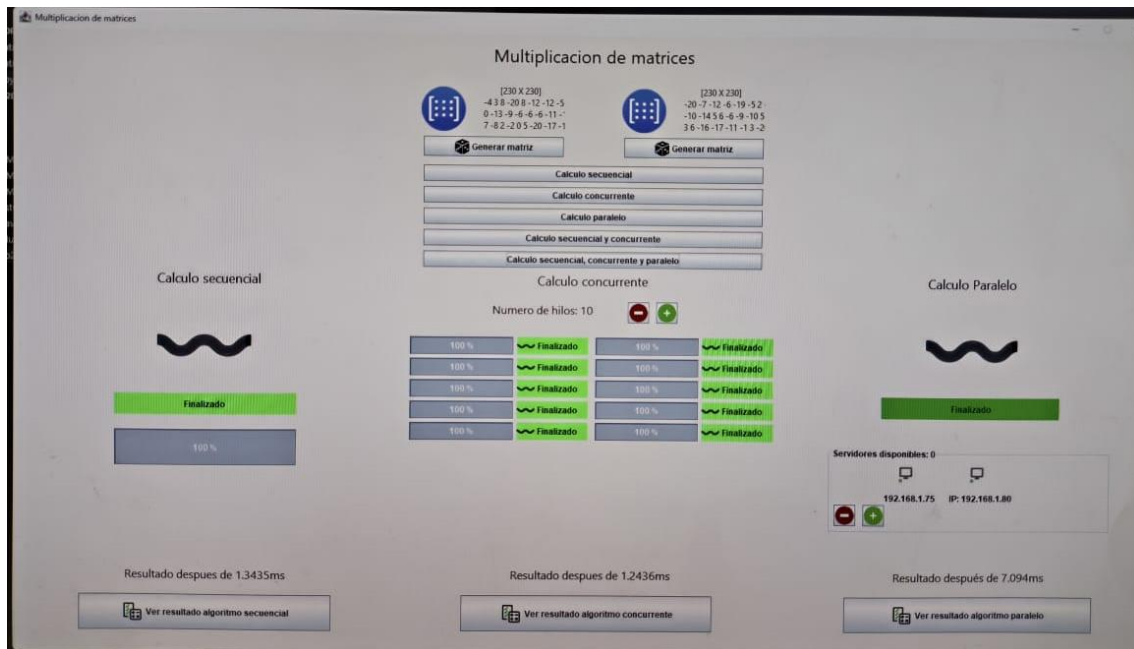
The screenshot shows the 'Multiplicacion de matrices' application interface. It displays three calculation methods for a 10x10 matrix:

- Calculo secuencial:** Shows a wavy line icon, a green 'Finalizado' bar, and a 100% progress bar. The result is shown after 0.1381ms.
- Calculo concurrente:** Shows a wavy line icon, a green 'Finalizado' bar, and a 100% progress bar. The result is shown after 2.4591ms.
- Calculo Paralelo:** Shows a wavy line icon, a green 'Finalizado' bar, and a 100% progress bar. The result is shown after 16.9109ms.

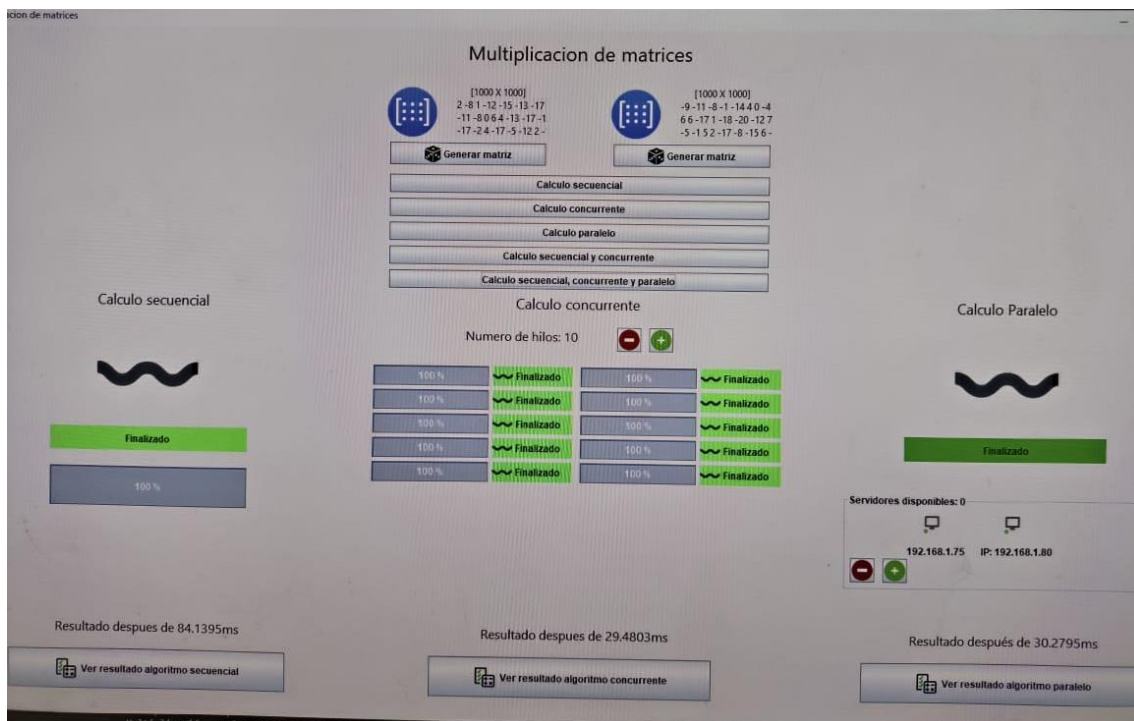
The interface also includes buttons for 'Generar matriz', 'Ver resultado algoritmo secuencial', 'Ver resultado algoritmo concurrente', and 'Ver resultado algoritmo paralelo'. The number of threads is set to 10.

Como se puede ver, la ejecución en secuencial es más rápida que la concurrente y la paralela. 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, y en el caso de la ejecución en paralelo, la comunicación hace que tarde más tiempo, pero este resultado va a ir cambiando a medida que la matriz se hace más grande.

Más o menos al multiplicar matrices de 230 x 230 cada matriz, las ejecuciones en secuencial y concurrente se resuelven en la misma cantidad de tiempo, pero para el caso del paralelo, aún no es la suficiente cantidad de procesamiento para terminar antes. A partir de este punto, la ejecución concurrente será más rápida que la secuencial y poco a poco será cada vez menos rápido que la ejecución paralela.



Y más o menos al momento de multiplicar matrices de 1000 x 1000 la ejecución concurrente y paralela son bastante similares en cuanto a tiempo, la ejecución secuencial ya es mucho más tardada que las otras dos.



The screenshot displays a software application titled "Multiplicacion de matrices" (Matrix Multiplication). The interface is divided into three main sections: sequential, concurrent, and parallel execution.

Sequential Execution (Calculo secuencial): This section shows a single wavy line icon and a progress bar that is 100% complete. The result is displayed as "Resultado despues de 658.0072ms".

Concurrent Execution (Calculo concurrente): This section shows a wavy line icon and a progress bar that is 100% complete. The result is displayed as "Resultado despues de 193.2653ms".

Parallel Execution (Calculo Paralelo): This section shows a wavy line icon and a progress bar that is 100% complete. The result is displayed as "Resultado despues de 172.3832ms".

The application also includes a central control panel with buttons for "Generar matriz" (Generate matrix), "Calculo secuencial" (Sequential calculation), "Calculo concurrente" (Concurrent calculation), "Calculo paralelo" (Parallel calculation), and "Calculo secuencial y concurrente" (Sequential and concurrent calculation). The "Calculo concurrente" button is currently selected.

At the bottom, there are three buttons labeled "Ver resultado algoritmo secuencial", "Ver resultado algoritmo concurrente", and "Ver resultado algoritmo paralelo".

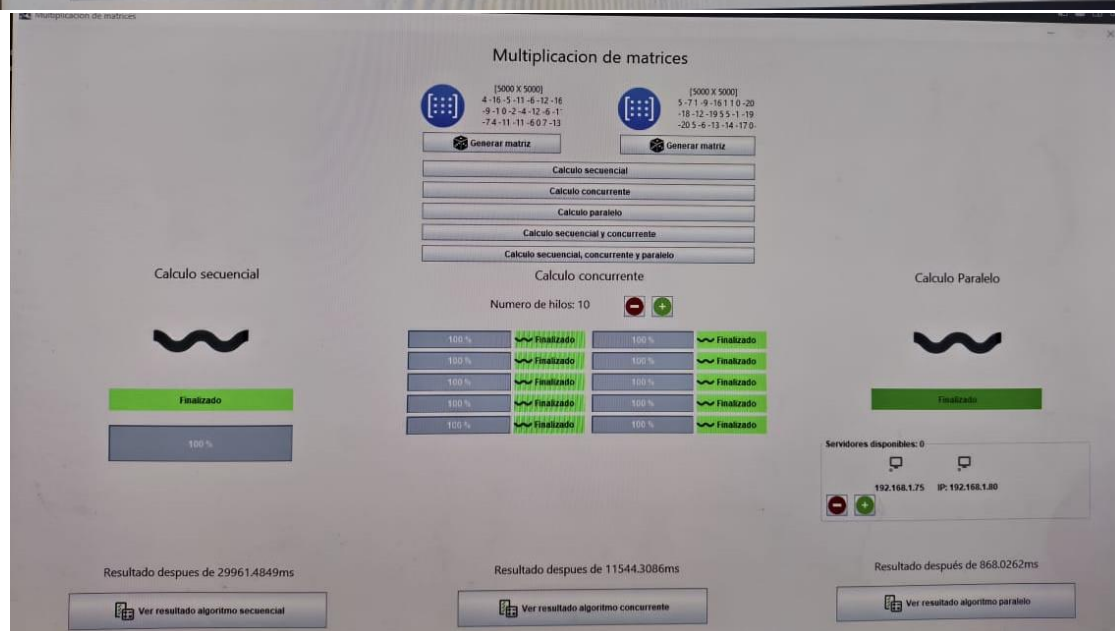


Tabla de resultados

	Caso1 (10x10)	Caso2 (100x100)	Caso3 (230x230)	Caso4 (1000x1000)	Caso5 (5000x5000)
Secuencial	138.1us	155us	1.34ms	84.14ms	29.96seg
Concurrente	669ms	680us	1.24ms	29.48ms	11.54seg
Paralelo	16.91ms	5.755ms	7.09ms	30.27ms	868.02ms

Conclusiones:

La computación distribuida y el cálculo paralelo son herramientas clave para resolver problemas intensivos en recursos, como la multiplicación de matrices, de manera eficiente. Delegar tareas a servidores mediante Java RMI permite optimizar el uso del hardware disponible y reducir tiempos de procesamiento, mientras que la separación entre cálculo y recuperación de resultados mejora la gestión de latencia y la resiliencia ante fallos.

Entre los principales desafíos destacan la sincronización en operaciones concurrentes y el manejo de errores en la conexión con servidores, aspectos que se superan mediante controles de concurrencia y un manejo robusto de excepciones. La presentación de resultados a través de una interfaz gráfica mejora la claridad y accesibilidad de la información, brindando una experiencia de usuario más efectiva.

Estas soluciones demuestran el potencial de la programación distribuida para abordar tareas complejas de manera escalable y eficiente, sentando las bases para desarrollos más robustos y adaptables en el futuro.