

Лістинг програми ПРГ1

Director.java

```
package com.vodotiiets.directors;

import com.vodotiiets.controllers.TaskController;
import com.vodotiiets.primitives.Matrix;
import com.vodotiiets.primitives.Vector;
import com.vodotiiets.workers.TaskWorker;

/**
 * Created by Denys Vodotiiets.
 */
public class Director {

    private static int  $N = 2500$ ;
    private static int  $P = 1$ ;
    private static int  $H = N / P$ ;

    private static int maxValue = 5;
    private static long startTime;
    private static long endTime;

    private static Matrix MA = new Matrix( $N$ );
    private static Matrix MO = new Matrix( $N$ );
    private static Matrix MK = new Matrix( $N$ );

    private static Vector B = new Vector( $N$ );
    private static Vector C = new Vector( $N$ );
    private static Vector Z = new Vector( $N$ );

    public static void main(String[] args) {

        if( $P > N$ ) {
             $P = N$ ;
             $H = 1$ ;
        }

        Thread[] tasks = new Thread[ $P$ ];
        TaskController monitor = new TaskController();

        for (int i = 0; i <  $P$ ; i++) {
            tasks[i] = new Thread(new TaskWorker(i, monitor));
        }

        startTime = System.currentTimeMillis();
        for (int i = 0; i <  $P$ ; i++) {
```

```

        tasks[i].start();
    }
}

public static int getN() {
    return N;
}

public static int getP() {
    return P;
}

public static int getH() {
    return H;
}

public static int getMaxValue() {
    return maxValue;
}

public static void setEndTime(long endTime) {
    Director.endTime = endTime;
}

public static long getTime() {
    return endTime - startTime;
}

public static Matrix getMA() {
    return MA;
}

public static Matrix getMO() {
    return MO;
}

public static Matrix getMK() {
    return MK;
}

public static Vector getB() {
    return B;
}

public static Vector getC() {
    return C;
}

public static Vector getZ() {
    return Z;
}

```

```
}  
}
```

TaskController.java

```
package com.vodotiiets.controllers;  
  
import com.vodotiiets.directors.Director;  
import com.vodotiiets.primitives.Matrix;  
  
/**  
 * Created by Denys Vodotiiets.  
 */  
public class TaskController {  
  
    private int inputCount;  
    private int preparationCount;  
    private int calculationCount;  
  
    private int a = Integer.MIN_VALUE;  
    private int s;  
    private Matrix MR = new Matrix(Director.getN());  
  
    public synchronized void waitInput() {  
        while (inputCount < 2) {  
            try {  
                wait();  
            } catch (InterruptedException e) {  
                System.out.println("Error has occurred in " + getClass().getSimpleName()  
                    + "when running waitInput()!" + e.getMessage());  
            }  
        }  
    }  
  
    public synchronized void waitCalculation() {  
        while (calculationCount < Director.getP() - 1) {  
            try {  
                wait();  
            } catch (InterruptedException e) {  
                System.out.println("Error has occurred in " + getClass().getSimpleName()  
                    + "when running waitCalculation()!" + e.getMessage());  
            }  
        }  
    }  
  
    public synchronized void waitPreparation() {  
        while (preparationCount < Director.getP()) {  
            try {  
                wait();  
            } catch (InterruptedException e) {
```

```

        System.out.println("Error has occurred in " + getClass().getSimpleName()
            + "when running waitPreparation()" + e.getMessage());
    }
}

public synchronized void inputMR() {
    MR.generate(Director.getMaxValue());
}

public synchronized void setA(int a) {
    if(a > this.a) {
        this.a = a;
    }
}

public synchronized void setS(int s) {
    this.s += s;
}

public synchronized Matrix getMR() {
    return new Matrix(MR);
}

public synchronized int getA() {
    return a;
}

public synchronized int getS() {
    return s;
}

public synchronized void signalInputDone() {
    inputCount++;
    if(inputCount == 2) {
        notifyAll();
    }
}

public synchronized void signalCalculationDone() {
    calculationCount++;
    if (calculationCount == Director.getP() - 1) {
        notifyAll();
    }
}

public synchronized void signalPreparationDone() {
    preparationCount++;
    if (preparationCount == Director.getP()) {
        notifyAll();
    }
}

```

```
}
```

TaskWorker.java

```
package com.vodotiiets.workers;

import com.vodotiiets.ParallelTasks.MaxTask;
import com.vodotiiets.ParallelTasks.VectorMultipleTask;
import com.vodotiiets.controllers.TaskController;
import com.vodotiiets.directors.Director;
import com.vodotiiets.primitives.Matrix;

import java.util.concurrent.ForkJoinPool;

/**
 * Created by Denys Vodotiiets.
 */
public class TaskWorker implements Runnable {
    private int tid;
    private TaskController monitor;

    private Matrix MRcopy;
    private int aCopy;
    private int sCopy;

    public TaskWorker(int tid, TaskController monitor) {
        this.monitor = monitor;
        this.tid = tid;
    }

    @Override
    public void run() {
        if (tid == 0) {
            monitor.inputMR();
            Director.getMA().generate(Director.getMaxValue());
            Director.getB().generate(Director.getMaxValue());

            monitor.signalInputDone();
        }

        if (tid == Director.getP() - 1) {
            Director.getC().generate(Director.getMaxValue());
            Director.getMO().generate(Director.getMaxValue());
            Director.getMK().generate(Director.getMaxValue());
            Director.getZ().generate(Director.getMaxValue());

            monitor.signalInputDone();
        }
    }
}
```

```

monitor.waitInput();

int startIndex = tid * Director.getH();
int endIndex = (tid != Director.getP() - 1) ? (tid + 1) * Director.getH() : Director.getN();

MaxTask maxTask = new MaxTask(Director.getZ().getPart(startIndex, endIndex));
int a = new ForkJoinPool().invoke(maxTask);
monitor.setA(a);

VectorMultipleTask vectorMultipleTask = new
VectorMultipleTask(Director.getB().getPart(startIndex, endIndex),
                    Director.getC().getPart(startIndex, endIndex));
int s = new ForkJoinPool(Director.getP()).invoke(vectorMultipleTask);

monitor.setS(s);

monitor.signalPreparationDone();
monitor.waitPreparation();

MRcopy = monitor.getMR();
aCopy = monitor.getA();
sCopy = monitor.getS();

for (int i = 0; i < Director.getN(); i++) {
    for (int j = startIndex; j < endIndex; j++) {
        int sum = 0;
        for (int k = 0; k < Director.getN(); k++) {
            sum += MRcopy.get(i, k) * Director.getMK().get(k, j);
        }

        int value = sCopy * Director.getMO().get(i, j) + aCopy * sum;
        Director.getMA().set(i, j, value);
    }
}

if (tid == 0) {
    monitor.waitCalculation();
} else {
    monitor.signalCalculationDone();
}

if (tid == 0) {
    Director.setEndTime(System.currentTimeMillis());
    System.out.println("All threads ended calculations. Result time(ms): " + Director.getTime());

    if(Director.getMA().getDimension() < 10) {
        System.out.println("Result MA:\n" + Director.getMA());
    } else {
        System.out.println("Result was calculated. Matrix is too large");
    }
}
}

```

```
}  
}
```

MaxTask.java

```
package com.vodotiiets.ParallelTasks;  
  
import com.vodotiiets.directors.Director;  
import com.vodotiiets.primitives.Vector;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.RecursiveTask;  
  
/**  
 * Created by Denys Vodotiiets.  
 */  
public class MaxTask extends RecursiveTask<Integer> {  
    private Vector vector;  
  
    public MaxTask(Vector vector) {  
        this.vector = vector;  
    }  
  
    @Override  
    protected Integer compute() {  
        if(vector.getDimension() > Director.getH() / Director.getP()) {  
            List<MaxTask> subtasks = createSubtasks();  
  
            for(MaxTask subtask : subtasks) {  
                subtask.fork();  
            }  
  
            int result = Integer.MIN_VALUE;  
            for(MaxTask subtask : subtasks) {  
                int temp = subtask.join();  
                if(temp > result) {  
                    result = temp;  
                }  
            }  
  
            return result;  
        } else {  
            return vector.getMaxElement(0, vector.getDimension());  
        }  
    }  
  
    private List<MaxTask> createSubtasks() {  
        List<MaxTask> subtasks = new ArrayList<>();
```

```

        MaxTask subtask1 = new MaxTask(this.vector.getPart(0, this.vector.getDimension()/2));
        MaxTask subtask2 = new MaxTask(this.vector.getPart(this.vector.getDimension()/2 + 1,
this.vector.getDimension()));

        subtasks.add(subtask1);
        subtasks.add(subtask2);

        return subtasks;
    }
}

```

VectorMultiple.java

```

package com.vodotiiets.ParallelTasks;

import com.vodotiiets.directors.Director;
import com.vodotiiets.primitives.Vector;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.RecursiveTask;

/**
 * Created by Denys Vodotiiets.
 */
public class VectorMultipleTask extends RecursiveTask<Integer> {

    private Vector vector1;
    private Vector vector2;
    private int dimension;

    public VectorMultipleTask(Vector vector1, Vector vector2) {
        this.vector1 = vector1;
        this.vector2 = vector2;

        dimension = vector1.getDimension();
    }

    @Override
    protected Integer compute() {
        if(dimension > Director.getH() / Director.getP()) {
            List<VectorMultipleTask> subtasks = createSubtasks();

            for(VectorMultipleTask subtask : subtasks) {
                subtask.fork();
            }

            int result = 0;
            for(VectorMultipleTask subtask : subtasks) {

```



```

        result += subtask.join();
    }

    return result;
} else {
    int result = 0;
    for (int i = 0; i < dimension; i++) {
        result += vector1.get(i) * vector2.get(i);
    }
    return result;
}
}

private List<VectorMultipleTask> createSubtasks() {
    List<VectorMultipleTask> subtasks = new ArrayList<>();

    VectorMultipleTask subtask1 = new VectorMultipleTask(vector1.getPart(0, vector1.getDimension()
/ 2),
        vector2.getPart(0, vector2.getDimension() / 2));
    VectorMultipleTask subtask2 = new VectorMultipleTask(vector1.getPart(vector1.getDimension() / 2
+ 1,
        vector1.getDimension()), vector2.getPart(vector2.getDimension() / 2 + 1,
vector2.getDimension()));

    subtasks.add(subtask1);
    subtasks.add(subtask2);

    return subtasks;
}
}

```

Vector.java

```
package com.vodotiiets.primitives;
```

```
import java.util.Random;
```

```
/**
```

```
 * Created by Denys Vodotiiets.
```

```
 */
```

```
public class Vector {
```

```
    private int dimension;
```

```
    private int[] array;
```

```
    public Vector(int dimension) {
```

```
        this.dimension = dimension;
```

```
        array = new int[dimension];
```

```
    }
```

```
    public Vector(int[] array) {
```

```
        this.array = array;
```

```
        this.dimension = array.length;
```

```

}

public void generate(int maxValue) {
    Random generator = new Random();

    for (int i = 0; i < dimension; i++) {
        array[i] = generator.nextInt(maxValue);
    }
}

public int getMaxElement(int startIndex, int endIndex) {
    if (startIndex < 0 || endIndex > dimension || startIndex >= endIndex)
        throw new IllegalStateException();

    int result = Integer.MIN_VALUE;
    for (int i = startIndex ; i < endIndex; i++) {
        if(array[i] > result) {
            result = array[i];
        }
    }
    return result;
}

public Vector getPart(int startIndex, int endIndex) {
    if(endIndex <= startIndex || endIndex > dimension || startIndex < 0)
        throw new IllegalStateException();

    int length = endIndex - startIndex;
    int[] resultArray = new int[length];
    System.arraycopy(this.array, startIndex, resultArray, 0, length);

    return new Vector(resultArray);
}

public int getDimension() {
    return dimension;
}

public int get(int i) {
    return array[i];
}

@Override
public String toString() {
    String result = "";

    for (int i = 0; i < dimension; i++) {
        result += array[i] + "\t";
    }

    return result;
}

```

```
}  
}
```

Matrix.java

```
package com.vodotiiets.primitives;  
  
import java.util.Random;  
  
/**  
 * Created by Denys Vodotiiets.  
 */  
public class Matrix {  
    private int dimension;  
    private int[][] array;  
  
    public Matrix(int dimension) {  
        this.dimension = dimension;  
        array = new int[dimension][dimension];  
    }  
  
    public Matrix(Matrix other) {  
        this.dimension = other.dimension;  
        array = new int [this.dimension][this.dimension];  
        for (int i = 0; i < this.dimension; i++) {  
            for(int j = 0; j < this.dimension; j++) {  
                this.array[i][j] = other.array[i][j];  
            }  
        }  
    }  
  
    public void generate(int maxValue) {  
        Random generator = new Random();  
        for (int i = 0; i < dimension; i++){  
            for (int j = 0; j < dimension; j++){  
                array[i][j] = generator.nextInt(maxValue);  
            }  
        }  
    }  
  
    public int get(int i, int j) {  
        return array[i][j];  
    }  
  
    public void set(int i, int j, int value) {  
        array[i][j] = value;  
    }  
  
    public int getDimension() {  
        return dimension;  
    }  
}
```

```
}

@Override
public String toString() {
    String result = "";
    for (int i = 0; i < dimension; i++){
        for (int j = 0; j < dimension; j++){
            result += array[i][j] + "\t";
        }
        result += "\n";
    }
    return result;
}
}
```

Лістинг програми ПРГ2

```

#include <iostream>
#include <limits.h>
#include <time.h>
#include "mpi.h"

using namespace std;

const int N = 100;

void inputMatrix(int matrix[N][N]);
void outputMatrix(int matrix[N][N]);
void inputVector(int vector[N]);
void outputVector(int vector[N]);

void ckeckSize(int r, int s);

int maxNumber(int vector[N], int start, int end);

void sendMatrixPart(int matrix[N][N], int start, int end, int dest, int tag);
void recvMatrixPart(int matrix[N][N], int start, int end, int source, int tag, MPI_Status
status);

int main(int argc, char* argv[])
{
    long start = clock();
    MPI_Init(&argc, &argv);

    int rank, size;
    int Z[N], B[N], C[N];
    int MA[N][N], MO[N][N], MR[N][N], MK[N][N];
    int a, s;

    MPI_Status status;
    int msgTag = 0;

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    ckeckSize(rank, size);

    int P = size;
    int H = N / P;

    //input P/2 + 1
    if (rank == P / 2)
    {
        inputVector(B);
        inputMatrix(MR);

        //send to 1
        MPI_Send(B, N, MPI_INT, 0, msgTag, MPI_COMM_WORLD);
        MPI_Send(MR, N * N, MPI_INT, 0, msgTag, MPI_COMM_WORLD);

        //send to P/2 + 2
        MPI_Send(B, N, MPI_INT, P / 2 + 1, msgTag, MPI_COMM_WORLD);
    }
}

```

```

MPI_Send(MR, N * N, MPI_INT, P / 2 + 1, msgTag, MPI_COMM_WORLD);

//receive from P/2 + 2
MPI_Recv(C, N, MPI_INT, P / 2 + 1, msgTag, MPI_COMM_WORLD, &status);
MPI_Recv(Z, N, MPI_INT, P / 2 + 1, msgTag, MPI_COMM_WORLD, &status);
MPI_Recv(MO, N * N, MPI_INT, P / 2 + 1, msgTag, MPI_COMM_WORLD, &status);
MPI_Recv(MK, N * N, MPI_INT, P / 2 + 1, msgTag, MPI_COMM_WORLD, &status);
}
//input P/2
else if (rank == P / 2 - 1)
{
    inputVector(C);
    inputVector(Z);
    inputMatrix(MO);
    inputMatrix(MK);

    //send to P/2 - 1
    MPI_Send(C, N, MPI_INT, P / 2 - 2, msgTag, MPI_COMM_WORLD);
    MPI_Send(Z, N, MPI_INT, P / 2 - 2, msgTag, MPI_COMM_WORLD);
    MPI_Send(MO, N * N, MPI_INT, P / 2 - 2, msgTag, MPI_COMM_WORLD);
    MPI_Send(MK, N * N, MPI_INT, P / 2 - 2, msgTag, MPI_COMM_WORLD);

    //send to P
    MPI_Send(C, N, MPI_INT, P - 1, msgTag, MPI_COMM_WORLD);
    MPI_Send(Z, N, MPI_INT, P - 1, msgTag, MPI_COMM_WORLD);
    MPI_Send(MO, N * N, MPI_INT, P - 1, msgTag, MPI_COMM_WORLD);
    MPI_Send(MK, N * N, MPI_INT, P - 1, msgTag, MPI_COMM_WORLD);

    //receive from P/2 - 1
    MPI_Recv(B, N, MPI_INT, P / 2 - 2, msgTag, MPI_COMM_WORLD, &status);
    MPI_Recv(MR, N * N, MPI_INT, P / 2 - 2, msgTag, MPI_COMM_WORLD, &status);
}
//input 1
else if (rank == 0)
{
    //receive from P/2 + 1
    MPI_Recv(B, N, MPI_INT, P / 2, msgTag, MPI_COMM_WORLD, &status);
    MPI_Recv(MR, N * N, MPI_INT, P / 2, msgTag, MPI_COMM_WORLD, &status);

    //send to 2
    MPI_Send(B, N, MPI_INT, 1, msgTag, MPI_COMM_WORLD);
    MPI_Send(MR, N * N, MPI_INT, 1, msgTag, MPI_COMM_WORLD);

    //receive from 2
    MPI_Recv(C, N, MPI_INT, 1, msgTag, MPI_COMM_WORLD, &status);
    MPI_Recv(Z, N, MPI_INT, 1, msgTag, MPI_COMM_WORLD, &status);
    MPI_Recv(MO, N * N, MPI_INT, 1, msgTag, MPI_COMM_WORLD, &status);
    MPI_Recv(MK, N * N, MPI_INT, 1, msgTag, MPI_COMM_WORLD, &status);
}
//input P
else if (rank == P - 1)
{
    //receive from P/2
    MPI_Recv(C, N, MPI_INT, P / 2 - 1, msgTag, MPI_COMM_WORLD, &status);
    MPI_Recv(Z, N, MPI_INT, P / 2 - 1, msgTag, MPI_COMM_WORLD, &status);
    MPI_Recv(MO, N * N, MPI_INT, P / 2 - 1, msgTag, MPI_COMM_WORLD, &status);
    MPI_Recv(MK, N * N, MPI_INT, P / 2 - 1, msgTag, MPI_COMM_WORLD, &status);

    //send to P - 1
    MPI_Send(C, N, MPI_INT, P - 2, msgTag, MPI_COMM_WORLD);
    MPI_Send(Z, N, MPI_INT, P - 2, msgTag, MPI_COMM_WORLD);
    MPI_Send(MO, N * N, MPI_INT, P - 2, msgTag, MPI_COMM_WORLD);
    MPI_Send(MK, N * N, MPI_INT, P - 2, msgTag, MPI_COMM_WORLD);
}

```

```

        //receive from P - 1
        MPI_Recv(B, N, MPI_INT, P - 2, msgTag, MPI_COMM_WORLD, &status);
        MPI_Recv(MR, N * N, MPI_INT, P - 2, msgTag, MPI_COMM_WORLD, &status);
    }
    else
    {
        //receive from rank - 1
        MPI_Recv(B, N, MPI_INT, rank - 1, msgTag, MPI_COMM_WORLD, &status);
        MPI_Recv(MR, N * N, MPI_INT, rank - 1, msgTag, MPI_COMM_WORLD, &status);

        //send to rank + 1
        MPI_Send(B, N, MPI_INT, rank + 1, msgTag, MPI_COMM_WORLD);
        MPI_Send(MR, N * N, MPI_INT, rank + 1, msgTag, MPI_COMM_WORLD);

        //receive from rank + 1
        MPI_Recv(C, N, MPI_INT, rank + 1, msgTag, MPI_COMM_WORLD, &status);
        MPI_Recv(Z, N, MPI_INT, rank + 1, msgTag, MPI_COMM_WORLD, &status);
        MPI_Recv(MO, N * N, MPI_INT, rank + 1, msgTag, MPI_COMM_WORLD, &status);
        MPI_Recv(MK, N * N, MPI_INT, rank + 1, msgTag, MPI_COMM_WORLD, &status);

        //send to rank - 1
        MPI_Send(C, N, MPI_INT, rank - 1, msgTag, MPI_COMM_WORLD);
        MPI_Send(Z, N, MPI_INT, rank - 1, msgTag, MPI_COMM_WORLD);
        MPI_Send(MO, N * N, MPI_INT, rank - 1, msgTag, MPI_COMM_WORLD);
        MPI_Send(MK, N * N, MPI_INT, rank - 1, msgTag, MPI_COMM_WORLD);
    }

    int startIndex = rank * H;
    int endIndex = (rank + 1) * H;
    if (rank == P - 1) {
        endIndex = N;
    }

    //Find max element
    int a_i = maxNumber(Z, startIndex, endIndex);

    MPI_Allreduce(&a_i, &a, 1, MPI_INT, MPI_MAX, MPI_COMM_WORLD);

    //calc B * C

    int s_i = 0;

    for (int i = startIndex; i < endIndex; i++)
    {
        s_i += B[i] * C[i];
    }

    MPI_Allreduce(&s_i, &s, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);

    //Calc MA_H
    for (int i = startIndex; i < endIndex; i++)
    {
        for (int j = 0; j < N; j++)
        {
            int sum = 0;

            for (int k = 0; k < N; k++)
            {
                sum += MR[i][k] * MK[k][j];
            }

            int resultValue = MO[i][j] * s + a * sum;
            MA[i][j] = resultValue;
        }
    }

```

```

    }

    if (rank == P / 2)
    {
        recvMatrixPart(MA, 0, rank * H, 0, msgTag, status);

        int previousStart = (P / 2 + 2) * H;

        recvMatrixPart(MA, (rank + 1) * H, N, P / 2 + 1, msgTag, status);

        long end = clock();
        cout << "Calculation ended. Time: " << end - start << endl;

        cout << "Result MA:" << endl;
        outputMatrix(MA);
    }
    else
    {
        if (rank == P - 1 || rank == P / 2 - 1)
        {
            sendMatrixPart(MA, startIndex, endIndex, rank - 1, msgTag);
        }
        else if (rank > P / 2 && rank < P - 1)
        {
            int previousStart = (rank + 1) * H;
            recvMatrixPart(MA, previousStart, N, rank + 1, msgTag, status);
            sendMatrixPart(MA, startIndex, N, rank - 1, msgTag);
        }
        else if (rank > 0 && rank < P / 2 - 1)
        {
            int previousStart = (rank + 1) * H;
            int previousEnd = (P / 2) * H;
            recvMatrixPart(MA, previousStart, previousEnd, rank + 1, msgTag, status);
            sendMatrixPart(MA, startIndex, previousEnd, rank - 1, msgTag);
        }
        else if (rank == 0)
        {
            recvMatrixPart(MA, endIndex, (P / 2) * H, rank + 1, msgTag, status);
            sendMatrixPart(MA, startIndex, (P / 2) * H, P / 2, msgTag);
        }
    }

    MPI_Finalize();
    return 0;
}

void inputMatrix(int matrix[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            matrix[i][j] = 1;
        }
    }
}

void outputMatrix(int matrix[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)

```



```

        {
            printf("%10d ", matrix[i][j]);
            //cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

void outputVector(int vector[N])
{
    for (int i = 0; i < N; i++)
    {
        printf("%10d ", vector[i]);
        //cout << vector[i] << " ";
    }
    cout << endl;
}

void inputVector(int vector[N])
{
    for (int i = 0; i < N; i++)
    {
        vector[i] = 1;
    }
}

void ckeckSize(int rank, int size)
{
    if (size < 4 )
    {
        if (rank == 0)
        {
            cout << "For correct program work the count of threads must be more than 3."
<<
            "Please make sure that you input correct data!" << endl <<
            "Your threads' size is " << size << endl;
        }
        MPI_Finalize();
        exit(-1);
    }

    if (N % 4 != 0)
    {
        if (rank == 0)
        {
            cout << "The dimension of the arrays must be a multiple of four." <<
            "Please make sure that you input correct data!" << endl <<
            "Current dimension is " << N << endl;
        }

        MPI_Finalize();
        exit(-1);
    }
}

int maxNumber(int vector[N], int start, int end)
{
    int result = INT_MIN;
    for (int i = start; i < end; i++)
    {
        if (vector[i] > result)
        {
            result = vector[i];
        }
    }
}

```

```

        }
    }

    return result;
}

void sendMatrixPart(int matrix[N][N], int start, int end, int dest, int tag)
{
    for (int i = start; i < end; i++)
    {
        MPI_Send(matrix[i], N, MPI_INT, dest, tag, MPI_COMM_WORLD);
    }
}

void recvMatrixPart(int matrix[N][N], int start, int end, int source, int tag, MPI_Status status)
{
    for (int i = start; i < end; i++)
    {
        MPI_Recv(matrix[i], N, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
    }
}

```