Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт

З комп’ютерного практикуму № 6 з дисципліни
«Технології паралельних обчислень»

Виконала студентка ІП-13 Замковий Дмитро Володимирович
(шифр, прізвище, ім'я, по батькові)

Перевірив ст.викл. Дифучин Антон Юрійович
(прізвище, ім'я, по батькові)

Оцінка _____

Дата оцінювання _____

Київ 2024

Лабораторна робота 6

Завдання

1. Ознайомитись з методами блокуючого та неблокуючого обміну повідомленнями типу point-to-point (див. лекцію та документацію стандарту MPI).
2. Реалізувати алгоритм паралельного множення матриць з використанням розподілених обчислень в MPI з використанням методів блокуючого обміну повідомленнями (лістинг 1). **30 балів.**
3. Реалізувати алгоритм паралельного множення матриць з використанням розподілених обчислень в MPI з використанням методів неблокуючого обміну повідомленнями. **30 балів.**
4. Дослідити ефективність розподіленого обчислення алгоритму множення матриць при збільшенні розміру матриць та при збільшенні кількості вузлів, на яких здійснюється запуск програми. Порівняйте ефективність алгоритму при використанні блокуючих та неблокуючих методів обміну повідомленнями. **40 балів.**

Лістинги програми

Файл task2\Main.java

```
import mpi.*;
import static java.lang.System.exit;

class Main {
    private static final int MASTER = 0;
    private static final int FROM_MASTER = 1;
    private static final int FROM_WORKER = 2;
    private static final int NRA = 62;
    private static final int NCA = 15;
    private static final int NCB = 8;
    private static final int NRB = NCA;
    private static final int NRC = NRA;
    private static final int NCC = NCB;
    private static final int N_SEND = 4096;
    private static int N_CALC;

    public static void main(String[] args) {
        double[][] a = new double[NRA][NCA];
        double[][] b = new double[NRB][NCB];
        double[][] c = new double[NRC][NCC];

        MPI.Init(args);
        int size = MPI.COMM_WORLD.Size();
        int rank = MPI.COMM_WORLD.Rank();

        Main.N_CALC = Math.ceilDiv(NRC * NCC, size - 1);

        if (size < 2) {
            System.out.println("Need at least two MPI tasks. Quitting...");
            MPI.COMM_WORLD.Abort(1);
            exit(1);
        }

        if (rank == MASTER) {
            System.out.format("mpi_mm has started with %d tasks.%n", size - 1);
```

```
        for (int i = 0; i < NRA; i++) {
            for (int j = 0; j < NCA; j++) {
                a[i][j] = 10;
            }
        }
        for (int i = 0; i < NRB; i++) {
            for (int j = 0; j < NCB; j++) {
                b[i][j] = 10;
            }
        }

        for (int dest = 1; dest < size; dest++) {
            int len = Math.min(N_CALC, NRC * NCC - (dest - 1) * N_CALC);
            send(len, dest, FROM_MASTER);

            int offset = (dest - 1) * N_CALC;
            send(offset, dest, FROM_MASTER);

            send(a, dest, FROM_MASTER);
            send(b, dest, FROM_MASTER);
        }

        for (int source = 1; source < size; source++) {
            int offset = receive(source, FROM_WORKER);
            int len = receive(source, FROM_WORKER);
            double[][] cBlock = receive(source, NRC, NCC, FROM_WORKER);

            for (int i = 0; i < len; i++) {
                int row = Math.floorDiv((offset + i), NCC);
                int col = (offset + i) % NCC;
                c[row][col] = cBlock[row][col];
            }
        }

        printMatrix(a, "Matrix A");
        printMatrix(b, "Matrix B");
        printMatrix(c, "Matrix C");
    } else {
        int len = receive(MASTER, FROM_MASTER);
        int offset = receive(MASTER, FROM_MASTER);
        a = receive(MASTER, NRA, NCA, FROM_MASTER);
        b = receive(MASTER, NRB, NCB, FROM_MASTER);

        for (int i = 0; i < len; i++) {
            int row = (offset + i) / NCC;
            int col = (offset + i) % NCC;
            c[row][col] = 0;

            for (int j = 0; j < NCA; j++) {
                c[row][col] += a[row][j] * b[j][col];
            }
        }

        send(offset, MASTER, FROM_WORKER);
        send(len, MASTER, FROM_WORKER);
        send(c, MASTER, FROM_WORKER);
    }
    MPI.Finalize();
}

private static void send(double[][] matrix, int dest, int tag) {
    int rows = matrix.length;
    int cols = matrix[0].length;
```

```
        double[] tmp = new double[rows * cols];

        for (int i = 0; i < rows; i++) {
            System.arraycopy(matrix[i], 0, tmp, i * cols, cols);
        }

        int nCountSend = Math.ceilDiv(rows * cols, N_SEND);
        for (int i = 0; i < nCountSend; i++) {
            int nSend = Math.min(N_SEND, rows * cols - i * N_SEND);
            MPI.COMM_WORLD.Send(tmp, i * N_SEND, nSend, MPI.DOUBLE, dest, tag);
        }
    }

    private static double[][] receive(int source, int rows, int cols, int tag) {
        double[][] matrix = new double[rows][cols];
        double[] tmp = new double[rows * cols];
        int nCountRecv = Math.ceilDiv(rows * cols, N_SEND);

        for (int i = 0; i < nCountRecv; i++) {
            int nRecv = Math.min(N_SEND, rows * cols - i * N_SEND);
            MPI.COMM_WORLD.Recv(tmp, i * N_SEND, nRecv, MPI.DOUBLE, source,
tag);
        }

        for (int i = 0; i < rows; i++) {
            System.arraycopy(tmp, i * cols, matrix[i], 0, cols);
        }

        return matrix;
    }

    private static void send(int value, int dest, int tag) {
        int[] tmp = new int[1];
        tmp[0] = value;
        MPI.COMM_WORLD.Send(tmp, 0, 1, MPI.INT, dest, tag);
    }

    private static int receive(int source, int tag) {
        int[] tmp = new int[1];
        MPI.COMM_WORLD.Recv(tmp, 0, 1, MPI.INT, source, tag);
        return tmp[0];
    }

    private static void printMatrix(double[][] matrix, String name) {
        System.out.print(" \n\n\n**********\n");
        System.out.println(name);
        System.out.println("**********");
        for (double[] row : matrix) {
            System.out.println();
            for (double elem : row) {
                System.out.format("%6.2f ", elem);
            }
        }
        System.out.println("\n**********");
    }
}
```

Файл task3\Main.java

```
import mpi.*;
import static java.lang.System.exit;

class Main {
```

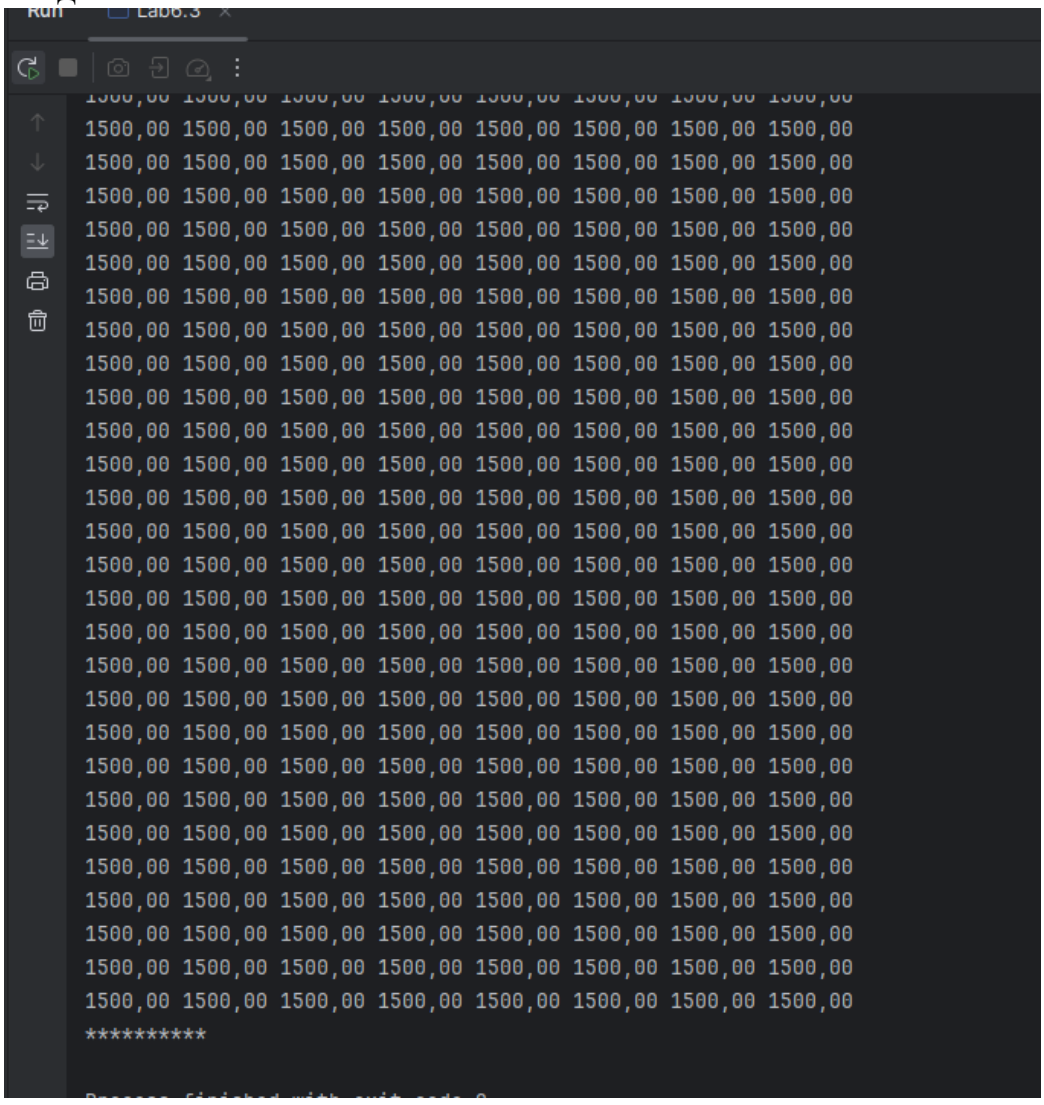
```java
    private static final int MASTER = 0;
    private static final int FROM_MASTER = 1;
    private static final int FROM_WORKER = 2;
    private static final int NRA = 62;
    private static final int NCA = 15;
    private static final int NCB = 8;
    private static final int NRB = NCA;
    private static final int NRC = NRA;
    private static final int NCC = NCB;
    private static final int N_SEND = 4096;
    private static int N_CALC;

    public static void main(String[] args) {
        double[][] a = new double[NRA][NCA];
        double[][] b = new double[NRB][NCB];
        double[][] c = new double[NRC][NCC];

        MPI.Init(args);
        int size = MPI.COMM_WORLD.Size();
        int rank = MPI.COMM_WORLD.Rank();

        Main.N_CALC = Math.ceilDiv(NRC * NCC, size - 1);

        if (size < 2) {
            System.out.println("Need at least two MPI tasks. Quitting...");
            MPI.COMM_WORLD.Abort(1);
            exit(1);
        }

        if (rank == MASTER) {
            System.out.format("mpi_mm has started with %d tasks.%n", size - 1);

            for (int i = 0; i < NRA; i++) {
                for (int j = 0; j < NCA; j++) {
                    a[i][j] = 10;
                }
            }
            for (int i = 0; i < NRB; i++) {
                for (int j = 0; j < NCB; j++) {
                    b[i][j] = 10;
                }
            }

            for (int dest = 1; dest < size; dest++) {
                int len = Math.min(N_CALC, NRC * NCC - (dest - 1) * N_CALC);
                send(len, dest, FROM_MASTER);
                int offset = (dest - 1) * N_CALC;
                send(offset, dest, FROM_MASTER);
                send(a, dest, FROM_MASTER);
                send(b, dest, FROM_MASTER);
            }

            for (int source = 1; source < size; source++) {
                int offset = receive(source, FROM_WORKER);
                int len = receive(source, FROM_WORKER);
                double[][] cBlock = receive(source, NRC, NCC, FROM_WORKER);

                for (int i = 0; i < len; i++) {
                    int row = Math.floorDiv((offset + i), NCC);
                    int col = (offset + i) % NCC;
                    c[row][col] = cBlock[row][col];
                }
            }
```

```
            Main.printMatrix(a, "Matrix A");
            Main.printMatrix(b, "Matrix B");
            Main.printMatrix(c, "Matrix C");
        } else {
            int len = receive(MASTER, FROM_MASTER);
            int offset = receive(MASTER, FROM_MASTER);
            a = receive(MASTER, NRA, NCA, FROM_MASTER);
            b = receive(MASTER, NRB, NCB, FROM_MASTER);

            for (int i = 0; i < len; i++) {
                int row = (offset + i) / NCC;
                int col = (offset + i) % NCC;
                c[row][col] = 0;
                for (int j = 0; j < NCA; j++) {
                    c[row][col] += a[row][j] * b[j][col];
                }
            }

            send(offset, MASTER, FROM_WORKER);
            send(len, MASTER, FROM_WORKER);
            send(c, MASTER, FROM_WORKER);
        }
        MPI.Finalize();
    }

    private static void send(double[][] matrix, int dest, int tag) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        double[] tmp = new double[rows * cols];
        for (int i = 0; i < rows; i++) {
            System.arraycopy(matrix[i], 0, tmp, i * cols, cols);
        }

        int nCountSend = Math.ceilDiv(rows * cols, N_SEND);
        for (int i = 0; i < nCountSend; i++) {
            int nSend = Math.min(N_SEND, rows * cols - i * N_SEND);
            MPI.COMM_WORLD.Isend(tmp, i * N_SEND, nSend, MPI.DOUBLE, dest, tag);
        }
    }

    private static double[][] receive(int source, int rows, int cols, int tag) {
        double[][] matrix = new double[rows][cols];
        double[] tmp = new double[rows * cols];

        int nCountRecv = Math.ceilDiv(rows * cols, N_SEND);
        Request[] r = new Request[nCountRecv];
        for (int i = 0; i < nCountRecv; i++) {
            int nRecv = Math.min(N_SEND, rows * cols - i * N_SEND);
            r[i] = MPI.COMM_WORLD.Irecv(tmp, i * N_SEND, nRecv, MPI.DOUBLE,
source, tag);
        }
        Request.Waitall(r);

        for (int i = 0; i < rows; i++) {
            System.arraycopy(tmp, i * cols, matrix[i], 0, cols);
        }

        return matrix;
    }

    private static void send(int value, int dest, int tag) {
        int[] tmp = new int[1];
        tmp[0] = value;
        MPI.COMM_WORLD.Isend(tmp, 0, 1, MPI.INT, dest, tag);
```

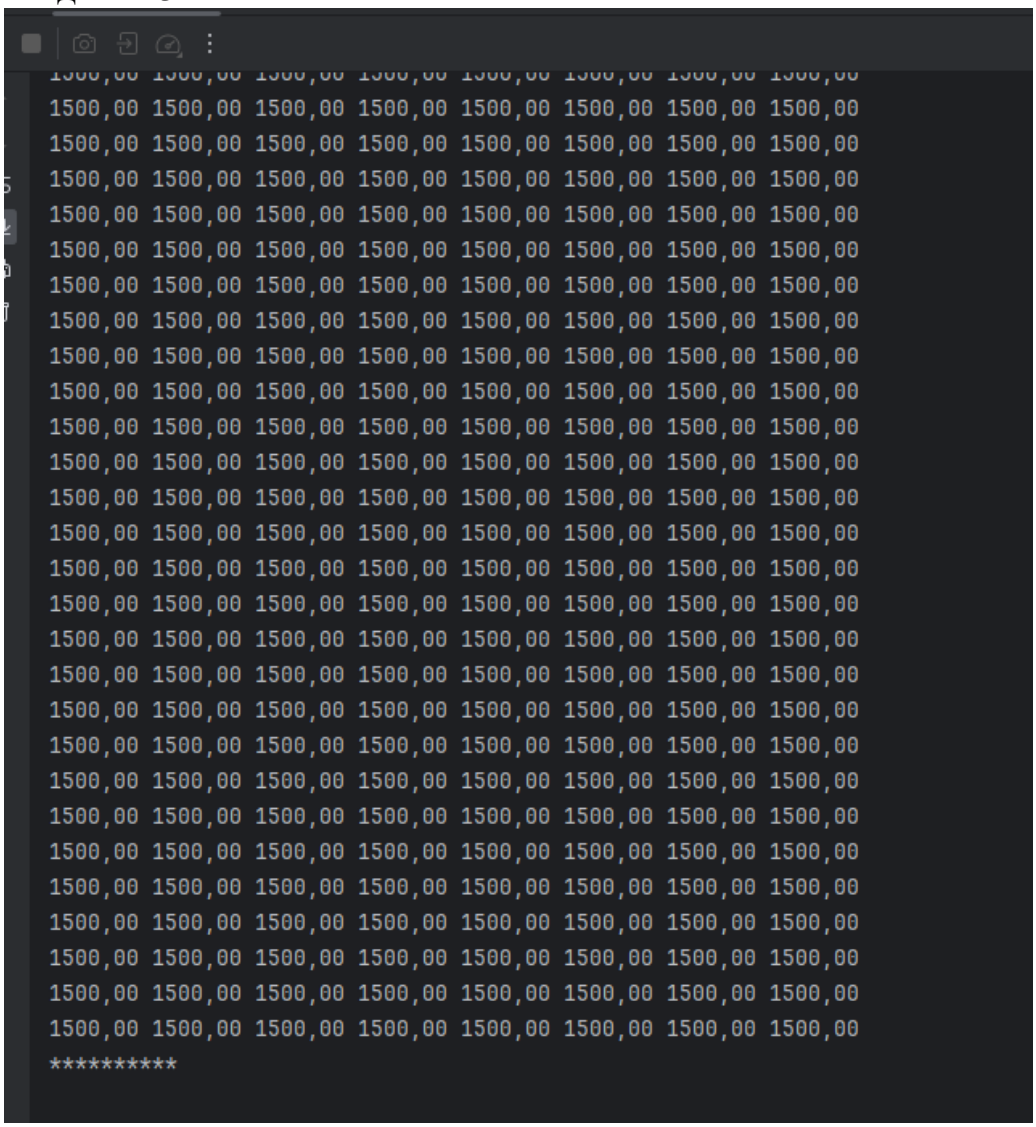
```
    }

    private static int receive(int source, int tag) {
        int[] tmp = new int[1];
        Request r = MPI.COMM_WORLD.Irecv(tmp, 0, 1, MPI.INT, source, tag);
        r.Wait();
        return tmp[0];
    }

    private static void printMatrix(double[][] matrix, String name) {
        System.out.print(" \n\n\n**********\n");
        System.out.println(name);
        System.out.println("**********");
        for (double[] row : matrix) {
            System.out.println();
            for (double elem : row) {
                System.out.format("%6.2f ", elem);
            }
        }
        System.out.println("\n**********");
    }
}
```

Скріншоти

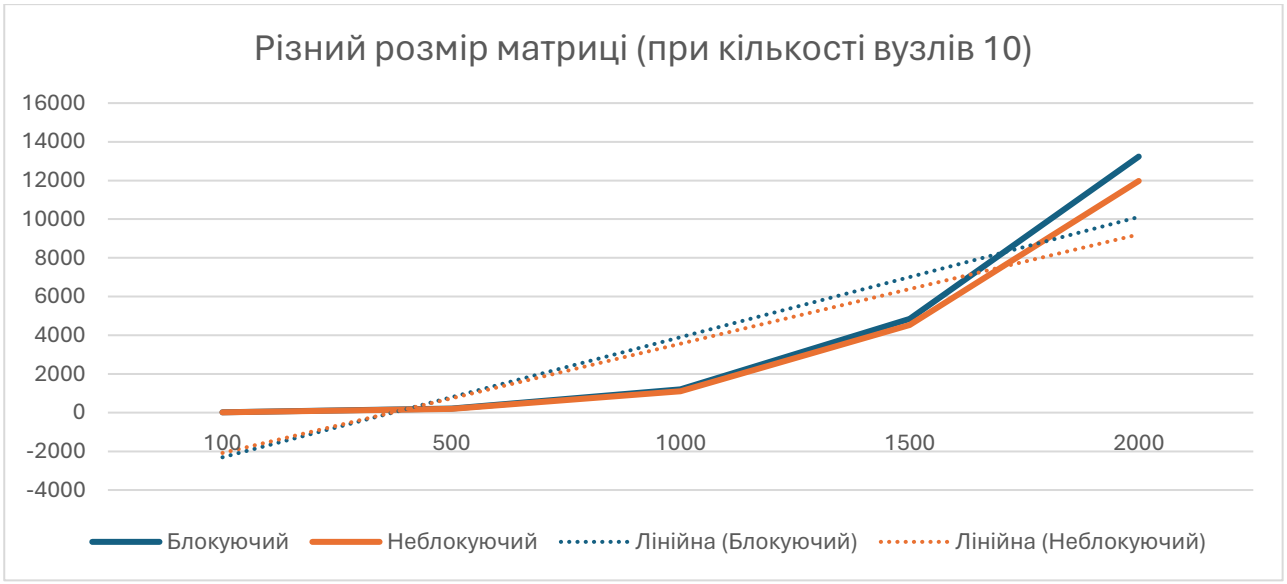
Завдання 2

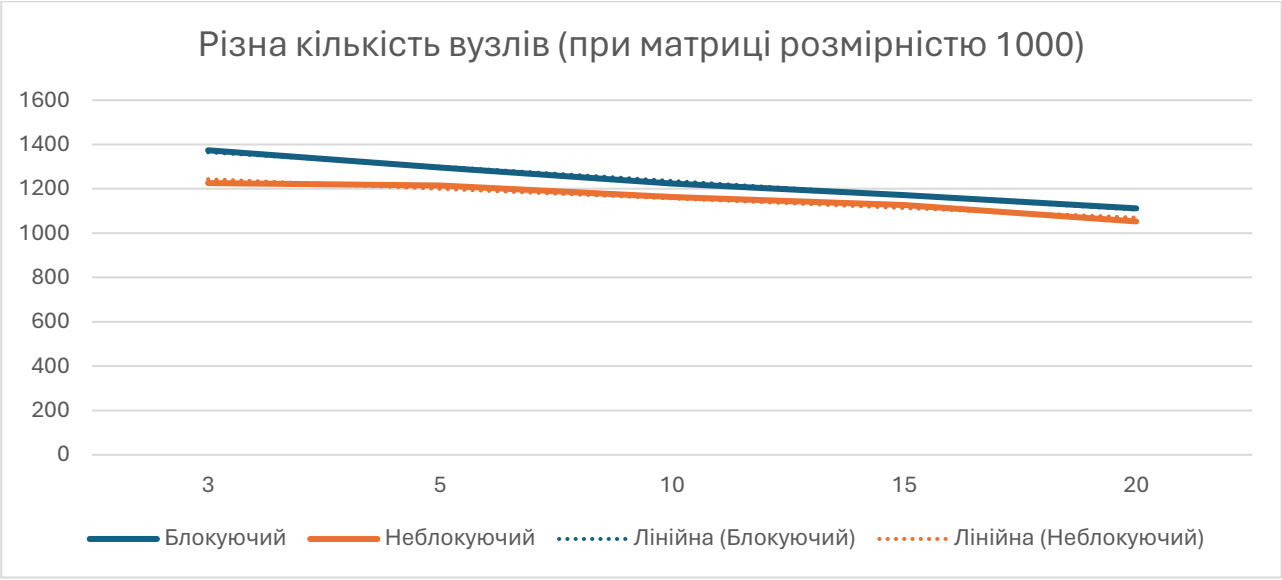


Завдання 3



Інші результати





Висновки

В ході даної лабораторної роботи було розроблено та досліджено програму на MPI з використанням блокуючих та неблокуючих методів обміну повідомлень. З результатів можемо зробити висновки, що неблокуючий метод повідомлень працює швидше, оскільки не очікую кожне прийняття повідомлення.