



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

Лабораторна робота №6
з дисципліни
Технології паралельних обчислень

Виконав:

студент групи ПІ-13:
Бабашев О. Д.

Перевірив:

ст.викл.
Дифучин А. Ю.

Київ 2024

Завдання до комп'ютерного практикуму 6

«Розробка паралельного алгоритму множення матриць з використанням MPI-методів обміну повідомленнями «один-до-одного» та дослідження його ефективності»

1. Ознайомитись з методами блокуючого та неблокуючого обміну повідомленнями типу point-to-point.
2. Реалізувати алгоритм паралельного множення матриць з використанням розподілених обчислень в MPI з використанням методів блокуючого обміну повідомленнями.
3. Реалізувати алгоритм паралельного множення матриць з використанням розподілених обчислень в MPI з використанням методів неблокуючого обміну повідомленнями.
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.

Хід роботи

Лістинг коду:

BlockingMultiplier.java

```
package lab6;

//import java.util.Random;
import mpi.MPI;

public class BlockingMultiplier {
    static final int A_ROWS = 1000;
    static final int A_COLS = 1000;
    static final int B_COLS = 1000;
    static final int MASTER = 0; /* id of main process */

    /* message types для комунікації між майстром та воркерами*/
    static final int FROM_MASTER = 1;
```

```

static final int FROM_WORKER = 2;

public static void main(String[] args) {

    int[][] A = new int[A_ROWS][A_COLS];
    int[][] B = new int[A_COLS][B_COLS];
    int[][] C = new int[A_ROWS][B_COLS];

    MPI.Init(args);
    int taskId = MPI.COMM_WORLD.Rank();
    int tasksNumber = MPI.COMM_WORLD.Size();
    int workersNumber = tasksNumber - 1;

    int[] rows = {0}, offset = {0};

    if (tasksNumber < 2) {
        System.out.println("At least 2 MPI tasks!");
        MPI.Finalize();
        System.exit(1);
    }

    if (taskId == MASTER) {

        //Random random = new Random();
        for (int i = 0; i < A_ROWS; i++) {
            for (int j = 0; j < A_COLS; j++) {
                A[i][j] = 10;
                //A[i][j] = random.nextInt(11);
            }
        }
    }
}

```

```

    }

    for (int i = 0; i < A_COLS; i++) {
        for (int j = 0; j < B_COLS; j++) {
            B[i][j] = 10;
            //B[i][j] = random.nextInt(11);
        }
    }

    int rowsForOneTask = A_ROWS / workersNumber;
    int rowsExtra = A_ROWS % workersNumber;

    long start = System.currentTimeMillis();
    // send data to tasks
    for (int destination = 1; destination <= workersNumber; destination++) {
        rows[0] = (destination <= rowsExtra) ? rowsForOneTask + 1 :
rowsForOneTask;

        MPI.COMM_WORLD.Send(offset, 0, 1, MPI.INT, destination,
FROM_MASTER);

        MPI.COMM_WORLD.Send(rows, 0, 1, MPI.INT, destination,
FROM_MASTER);

        MPI.COMM_WORLD.Send(A, offset[0], rows[0], MPI.OBJECT,
destination, FROM_MASTER);

        MPI.COMM_WORLD.Send(B, 0, A_COLS, MPI.OBJECT, destination,
FROM_MASTER);

        System.out.println("Sent " + rows[0] + " rows to worker " + destination);

        offset[0] += rows[0];
    }

```

```

// receive results from tasks
for (int source = 1; source <= workersNumber; source++) {

    MPI.COMM_WORLD.Recv(offset, 0, 1, MPI.INT, source,
FROM_WORKER);

    MPI.COMM_WORLD.Recv(rows, 0, 1, MPI.INT, source,
FROM_WORKER);

    MPI.COMM_WORLD.Recv(C, offset[0], rows[0], MPI.OBJECT,
source, FROM_WORKER);

    System.out.println("Worker " + source + " ready");
}

long end = System.currentTimeMillis();

System.out.println("Blocking time: " + (end - start));

int SequentialMatrix[][] = new int[A_ROWS][B_COLS];
SequentialMultiplier.multiply(A, B, SequentialMatrix);
MatrixHelper.compareMatrices(C, SequentialMatrix);

} else { // worker task
    // receive data

    MPI.COMM_WORLD.Recv(offset, 0, 1, MPI.INT, MASTER,
FROM_MASTER);

    MPI.COMM_WORLD.Recv(rows, 0, 1, MPI.INT, MASTER,
FROM_MASTER);

    MPI.COMM_WORLD.Recv(A, 0, rows[0], MPI.OBJECT, MASTER,
FROM_MASTER);

    MPI.COMM_WORLD.Recv(B, 0, A_COLS, MPI.OBJECT, MASTER,
FROM_MASTER);

    for (int k = 0; k < B_COLS; k++) {

```

```

        for (int i = 0; i < rows[0]; i++) {
            for (int j = 0; j < A_COLS; j++) {
                C[i][k] += A[i][j] * B[j][k];
            }
        }
    }

    // send result
    MPI.COMM_WORLD.Send(offset, 0, 1, MPI.INT, MASTER,
FROM_WORKER);

    MPI.COMM_WORLD.Send(rows, 0, 1, MPI.INT, MASTER,
FROM_WORKER);

    MPI.COMM_WORLD.Send(C, 0, rows[0], MPI.OBJECT, MASTER,
FROM_WORKER);

}

MPI.Finalize();
}
}

```

MatrixHelper.java

```
package lab6;
```

```

public class MatrixHelper {

    public static void printMatrix(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            System.out.println();
            for (int j = 0; j < matrix[0].length; j++) {
                System.out.print(matrix[i][j] + " ");
            }
        }
    }
}

```

```

    }
}
}

```

```

public static void compareMatrices(int[][] a, int[][] b) {
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a[0].length; j++) {
            if (a[i][j] != b[i][j]) {
                System.out.println("\nNOT Correct");
                return;
            }
        }
    }
    System.out.println("\nCorrect");
}

```

```

}

```

NonBlockingMultiplier.java

```

package lab6;

```

```

//import java.util.Random;

```

```

import mpi.MPI;

```

```

import mpi.Request;

```

```

public class NonBlockingMultiplier {
    static final int A_ROWS = 1000;
    static final int A_COLS = 1000;
    static final int B_COLS = 1000;
    static final int MASTER = 0; /* id of main process */

```

```
/* message types для комунікації між майстром та воркерами*/
```

```
static final int FROM_MASTER = 1;
```

```
static final int FROM_WORKER = 2;
```

```
public static void main(String[] args) {
```

```
    int[][] a = new int[A_ROWS][A_COLS];
```

```
    int[][] b = new int[A_COLS][B_COLS];
```

```
    int[][] c = new int[A_ROWS][B_COLS];
```

```
    MPI.Init(args);
```

```
    int taskId = MPI.COMM_WORLD.Rank();
```

```
    int tasksNumber = MPI.COMM_WORLD.Size();
```

```
    int workersNumber = tasksNumber - 1;
```

```
    int[] rows = {0}, offset = {0};
```

```
    if (tasksNumber < 2) {
```

```
        System.out.println("At least 2 tasks!");
```

```
        MPI.Finalize();
```

```
        System.exit(1);
```

```
    }
```

```
    if (taskId == MASTER) {
```

```
        //Random random = new Random();
```

```
        for (int i = 0; i < A_ROWS; i++) {
```

```
            for (int j = 0; j < A_COLS; j++) {
```

```
                a[i][j] = 10;
```



```

        //a[i][j] = random.nextInt(11);
    }
}

for (int i = 0; i < A_COLS; i++) {
    for (int j = 0; j < B_COLS; j++) {
        b[i][j] = 10;
        //b[i][j] = random.nextInt(11);
    }
}

int rowsForOneTask = A_ROWS / workersNumber;
int rowsExtra = A_ROWS % workersNumber;

long start = System.currentTimeMillis();

// send data to worker tasks
for (int destination = 1; destination <= workersNumber; destination++) {
    rows[0] = (destination <= rowsExtra) ? rowsForOneTask + 1 :
rowsForOneTask;

    MPI.COMM_WORLD.Isend(b, 0, A_COLS, MPI.OBJECT, destination,
FROM_MASTER);

    MPI.COMM_WORLD.Isend(offset, 0, 1, MPI.INT, destination,
FROM_MASTER);

    MPI.COMM_WORLD.Isend(rows, 0, 1, MPI.INT, destination,
FROM_MASTER);

    MPI.COMM_WORLD.Isend(a, offset[0], rows[0], MPI.OBJECT,
destination, FROM_MASTER);

    System.out.println("Sent " + rows[0] + " rows to worker " + destination);

    offset[0] += rows[0];

```

```

    }

    // receive results from tasks
    for (int source = 1; source <= workersNumber; source++) {

        MPI.COMM_WORLD.Irecv(offset, 0, 1, MPI.INT, source,
FROM_WORKER).Wait();

        MPI.COMM_WORLD.Irecv(rows, 0, 1, MPI.INT, source,
FROM_WORKER).Wait();

        MPI.COMM_WORLD.Irecv(c, offset[0], rows[0], MPI.OBJECT,
source, FROM_WORKER).Wait();

        System.out.println("Worker " + source + " ready");
    }

    long end = System.currentTimeMillis();

    System.out.println("\nNonBlocking time: " + (end - start));

    int[][] SequentialMatrix = new int[A_ROWS][B_COLS];
    SequentialMultiplier.multiply(a, b, SequentialMatrix);
    MatrixHelper.compareMatrices(c, SequentialMatrix);

} else { // worker task

    Request reqB = MPI.COMM_WORLD.Irecv(b, 0, A_COLS,
MPI.OBJECT, MASTER, FROM_MASTER);

    MPI.COMM_WORLD.Irecv(offset, 0, 1, MPI.INT, MASTER,
FROM_MASTER).Wait();

    MPI.COMM_WORLD.Irecv(rows, 0, 1, MPI.INT, MASTER,
FROM_MASTER).Wait();

```

```

        MPI.COMM_WORLD.Isend(offset, 0, 1, MPI.INT, MASTER,
FROM_WORKER);

        MPI.COMM_WORLD.Isend(rows, 0, 1, MPI.INT, MASTER,
FROM_WORKER);

        MPI.COMM_WORLD.Irecv(a, 0, rows[0], MPI.OBJECT, MASTER,
FROM_MASTER).Wait();

```

```

    reqB.Wait();

```

```

    for (int k = 0; k < B_COLS; k++) {
        for (int i = 0; i < rows[0]; i++) {
            for (int j = 0; j < A_COLS; j++) {
                c[i][k] += a[i][j] * b[j][k];
            }
        }
    }

```

```

        MPI.COMM_WORLD.Isend(c, 0, rows[0], MPI.OBJECT, MASTER,
FROM_WORKER);
    }

```

```

    MPI.Finalize();
}
}

```

SequentialMultiplier.java

```

package lab6;

```

```

//import java.util.Random;

```

```

public class SequentialMultiplier {
    static final int A_ROWS = 1000;

```

```

static final int A_COLS = 1000;
static final int B_COLS = 1000;

public static void multiply(int[][] a, int[][] b, int[][] c) {
    for (int i = 0; i < A_ROWS; i++) {

        for (int j = 0; j < B_COLS; j++) {

            for (int k = 0; k < A_COLS; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}

```

```

public static void main(String[] args) {

    int[][] A = new int[A_ROWS][A_COLS];

    int[][] B = new int[A_COLS][B_COLS];
    int[][] C = new int[A_ROWS][B_COLS];

    //Random random = new Random();
    for (int i = 0; i < A_ROWS; i++) {
        for (int j = 0; j < A_COLS; j++) {
            A[i][j] = 10;

            //A[i][j] = random.nextInt(11);
        }
    }
}

```

```
for (int i = 0; i < A_COLS; i++) {  
    for (int j = 0; j < B_COLS; j++) {  
        B[i][j] = 10;  
        //B[i][j] = random.nextInt(11);  
    }  
}
```

```
long start = System.currentTimeMillis();  
multiply(A, B, C);  
long end = System.currentTimeMillis();
```

```
System.out.println("\nMatrix A:\n");  
MatrixHelper.printMatrix(A);
```

```
System.out.println("\nMatrix B:\n");  
MatrixHelper.printMatrix(B);
```

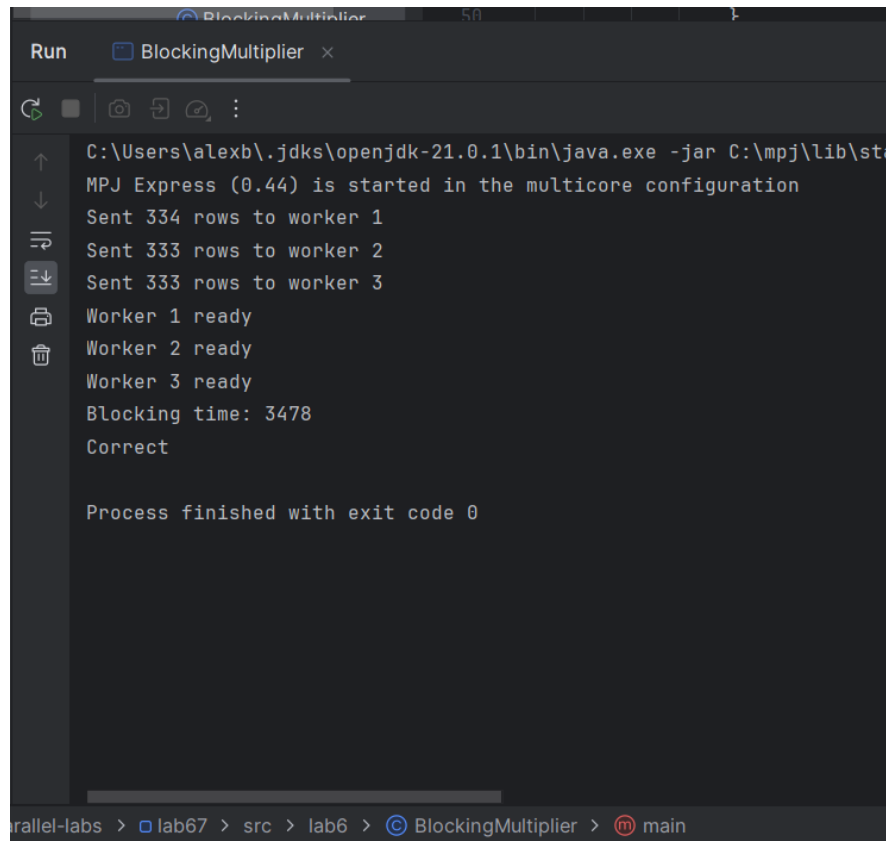
```
System.out.println("\nMatrix C:\n");  
MatrixHelper.printMatrix(C);
```

```
System.out.println("\nSequential time: " + (end - start));  
}  
  
}
```

Результати виконання коду:

Для блокуючої передачі повідомлень використовуємо два основні методи Send та Recv. Методи send та recv є фундаментальними для паралельного програмування та обміну повідомленнями між процесами або потоками. Вони забезпечують ефективну передачу даних і дозволяють реалізувати складні системи, які потребують взаємодії різних компонентів.

Протестовано роботу методу на 4 процессах.



```
Run BlockingMultiplier x
C:\Users\alexh\.jdk\openjdk-21.0.1\bin\java.exe -jar C:\mpj\lib\sta
MPJ Express (0.44) is started in the multicore configuration
Sent 334 rows to worker 1
Sent 333 rows to worker 2
Sent 333 rows to worker 3
Worker 1 ready
Worker 2 ready
Worker 3 ready
Blocking time: 3478
Correct

Process finished with exit code 0
```

Рисунок 2.1 – Тестування методу з блокуючим обміном повідомленнями

Після виконання паралельного множення матриць блокуючим методом маємо коректний результат.

Для неблокуючої передачі повідомлень використовуємо інші два методи isend та irectv. Методи isend і irectv є асинхронними варіантами традиційних методів send і recv, які використовуються для передачі повідомлень у паралельному програмуванні. Асинхронні методи дозволяють процесам або потокам продовжувати виконання інших завдань, не чекаючи завершення операцій відправки або прийому даних. Це може підвищити продуктивність, особливо у випадках, коли необхідно обробляти великі обсяги даних або підтримувати високу пропускну здатність мережі.

Протестовано роботу методу на 4 процессах.

```

Run NonBlockingMultiplier x
C:\Users\alexh\jdk\openjdk-21.0.1\bin\java.exe -jar
MPJ Express (0.44) is started in the multicore configu
Sent 334 rows to worker 1
Sent 333 rows to worker 2
Sent 333 rows to worker 3
Worker 1 ready
Worker 2 ready
Worker 3 ready
NonBlocking time: 2558
Correct

Process finished with exit code 0
parallel-labs > lab67 > src > lab6 > BlockingMultiplier > main

```

Рисунок 2.2 – Тестування методу з неблокуючим обміном повідомленнями

Після виконання паралельного множення матриць неблокуючим методом маємо коректний результат.

Аналіз

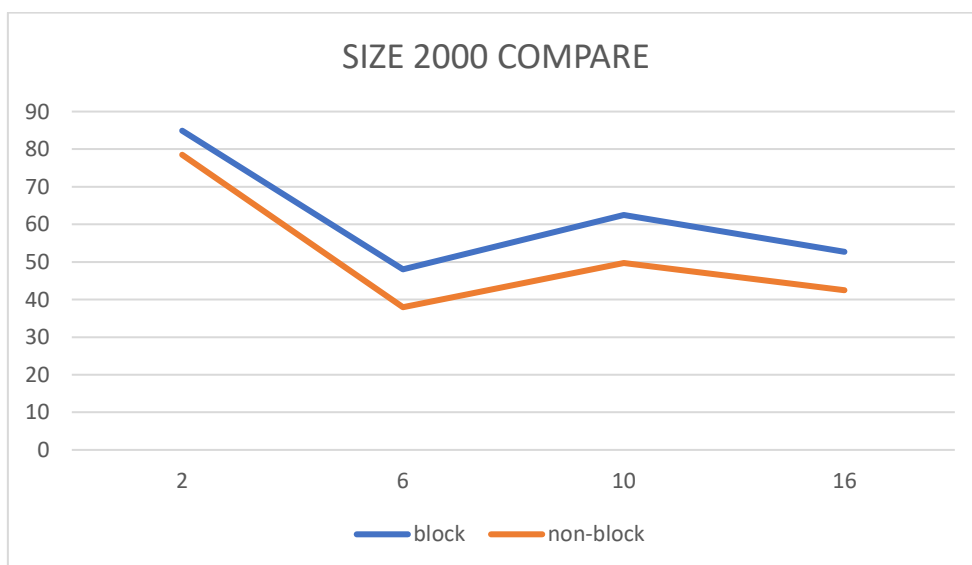
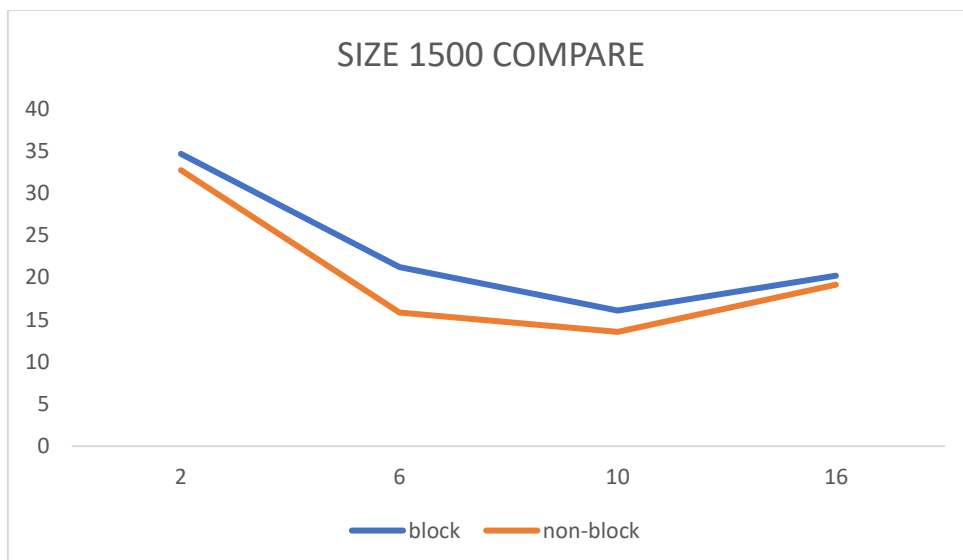
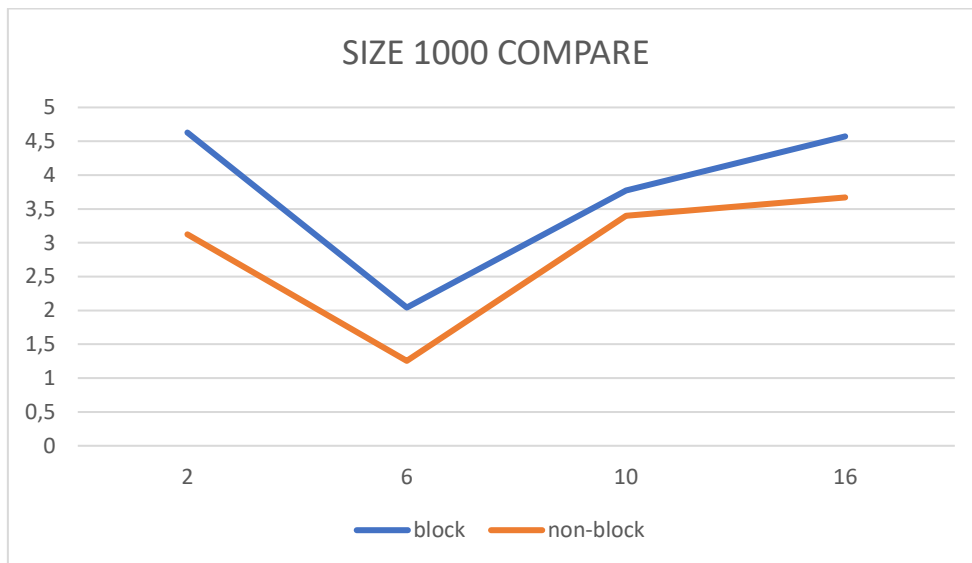
Тестування з блокуючими методами.

size np	2	6	10	16
500	1.015	0.463	0.439	0.671
1000	4.629	2.043	3.771	4.571
1500	34.679	21.233	16.084	20.224
2000	84.908	48.037	62.464	52.680

Тестування з неблокуючими методами.

	2	6	10	16
500	0.688	0.313	0.417	0.541
1000	3.123	1.255	3.399	3.670
1500	32.743	15.868	13.558	19.146
2000	78.507	37.957	49.718	42.469

Графіки залежності швидкості виконання від кількості процесів на різних розмірах матриць.



Узагальнюючи результати наших тестувань, можемо зазначити, що неблокуючий метод працює швидше. Неблокуючі методи працюють швидше і

ефективніше, оскільки вони дозволяють виконувати інші завдання, не чекаючи завершення операцій відправки або прийому даних. Це забезпечує кращу утилізацію ресурсів, зменшення затримок і підвищення загальної продуктивності паралельних програм.

Висновок

У цьому комп'ютерному практикумі ми ознайомились з роботою MPI, використовуючи Java-бібліотеку MPJExpress для реалізації задач передачі повідомлень двома способами: з блокуючою передачею та з неблокуючою передачею. Після реалізації паралельних алгоритмів множення матриць ми провели дослідження ефективності використаних методів та зробили висновки з отриманих результатів.