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 №7 з дисципліни
«Технології паралельних обчислень»

«Розробка паралельного алгоритму множення матриць з використанням MPI-методів колективного обміну повідомленнями («один-до-багатьох», «багато-до одного», «багатодо-багатьох») та дослідження його ефективності»

Виконав(ла) *ІП-11 Прищепа В. С.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Дифучин А. Ю.* _____
(прізвище, ім'я, по батькові)

Київ 2024

Завдання

1. Ознайомитись з методами колективного обміну повідомленнями типу «один-до-багатьох», «багато-до-одного», «багато-до-багатьох» (див. лекцію та документацію стандарту MPI).

2. Реалізувати алгоритм паралельного множення матриць з використанням розподілених обчислень в MPI з використанням методів колективного обміну повідомленнями. 40 балів.

3. 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 методів обміну повідомленнями «один-доодного», «один-до-багатьох», «багато-до-одного», «багато-до-багатьох». 60 балів.

Виконання

Лістинг

CollectiveMain.java

```
package org.example;

import mpi.*;

import java.util.Random;

public class CollectiveMain {
    private static final int MASTER = 0;
    private static final int MATRIX_SIZE = 5;

    public static void main(String[] args) {
        int[] A = new int[MATRIX_SIZE * MATRIX_SIZE];
        int[] B = new int[MATRIX_SIZE * MATRIX_SIZE];

        MPI.Init(args);

        int me = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();

        int[] sendcounts = new int[size];
        int[] displs = new int[size];
```

```
        if (size < 1) {
            MPI.COMM_WORLD.Abort(1);
            throw new RuntimeException("Need at least one MPI task. Quitting...\n");
        }

        long startTime = 0;
        if (me == MASTER) {
            System.out.printf("MPI has started with %d tasks.\n", size);

            Random random = new Random();
            for (int i = 0; i < MATRIX_SIZE; i++) {
                for (int j = 0; j < MATRIX_SIZE; j++) {
                    A[MATRIX_SIZE * i + j] = random.nextInt(10);
                }
            }
            for (int i = 0; i < MATRIX_SIZE; i++) {
                for (int j = 0; j < MATRIX_SIZE; j++) {
                    B[MATRIX_SIZE * i + j] = random.nextInt(10);
                }
            }
            //printMatrix(A, "A");
            //printMatrix(B, "B");

            startTime = System.currentTimeMillis();
        }

        int avgRow = MATRIX_SIZE / size;
        for (int i = 0; i < size; i++) {
            sendcounts[i] = (i < MATRIX_SIZE % size) ? (avgRow + 1) * MATRIX_SIZE :
avgRow * MATRIX_SIZE;
        }

        displs[0] = 0;
        for (int i = 1; i < size; i++) {
            displs[i] = displs[i-1] + sendcounts[i-1];
        }

        int[] recvA = new int[sendcounts[me]];
        MPI.COMM_WORLD.Scatterv(A, 0, sendcounts, displs, MPI.INT, recvA, 0,
sendcounts[me], MPI.INT, MASTER);
        MPI.COMM_WORLD.Bcast(B, 0, MATRIX_SIZE * MATRIX_SIZE, MPI.INT,
MASTER);

        int numberOfRows = recvA.length / MATRIX_SIZE;
        int[] C = new int[numberOfRows * MATRIX_SIZE];
```

```
        // Perform matrix multiplication
        for (int k = 0; k < MATRIX_SIZE; k++) {
            for (int i = 0; i < numberOfRows; i++) {
                C[MATRIX_SIZE * i + k] = 0;
                for (int j = 0; j < MATRIX_SIZE; j++) {
                    C[MATRIX_SIZE * i + k] += recvA[MATRIX_SIZE * i + j] *
B[MATRIX_SIZE * j + k];
                }
            }
        }

        int[] recvC = null;
        if (me == MASTER) {
            recvC = new int[MATRIX_SIZE * MATRIX_SIZE];
        }
        MPI.COMM_WORLD.Gatherv(C, 0, C.length, MPI.INT, recvC, 0, sendcounts,
displs, MPI.INT, MASTER);

        if (me == MASTER) {
            System.out.printf("Execution time for matrix %dx%d and %d workers:
%dms\n", MATRIX_SIZE, MATRIX_SIZE, size, System.currentTimeMillis() -
startTime);
            //printMatrix(recvC, "Result");
        }

        MPI.Finalize();
    }

    private static void printMatrix(int[] matrix, String name) {
        System.out.printf("Matrix %s:\n", name);
        for (int i = 0; i < MATRIX_SIZE; i++) {
            System.out.println();
            for (int j = 0; j < MATRIX_SIZE; j++)
                System.out.printf("%6d ", matrix[MATRIX_SIZE * i + j]);
        }
        System.out.println("\n" + "*".repeat(10));
    }
}
```

Результат

```
MPJ Express (0.44) is started in the multicore configuration
MPI has started with 2 tasks.
Matrix A:
     3      0      1      6      3
     0      7      6      5      0
     4      5      5      1      5
     5      9      6      2      8
     3      8      7      7      5
**********
Matrix B:
     9      0      5      6      5
     7      9      7      6      7
     3      7      4      2      7
     7      1      1      9      1
     4      4      0      0      7
**********
Execution time for matrix 5x5 and 2 workers: 3ms
Matrix Result:
    84     25     25     74     49
   102    110     78     99     96
   113    101     76     73    126
   172    157    114    114    188
   173    148    106    143    162
**********
```

Головний процес ініціалізує матриці A та B і їх розсилає. Матрицю A він розсилає в усі процеси з допомогою Scatterv, щоб правильно розділити матрицю. Матрицю B він розсилає в усі процеси з допомогою Bcast. Потім кожен процес виконує множення своєї локальної частини матриці A на матрицю B. Після цього головний процес збирає результати множень з процесів з допомогою Gatherv.

Тепер дослідимо ефективність розподіленого обчислення алгоритму множення матриць при збільшенні розміру матриць та при збільшенні кількості вузлів, на яких здійснюється запуск програми. Порівняймо ефективність алгоритму при використанні методів обміну повідомленнями «один-до-одного», «один-до-багатьох», «багато-до-одного», «багато-до-багатьох».

Розмірність	1 поток		2 потоки		4 потоки		8 потоків	
	1 до 1	К	1 до 1	К	1 до 1	К	1 до 1	К
500x500	266	275	151	149	87	82	66	77
1000x1000	2324	2074	1123	1082	618	557	407	382
2000x2000	29453	23252	14690	14433	7568	7720	4515	4538
2500x2500	73069	69134	36987	36741	18811	18636	10670	10871

Бачимо, що прискорення для колективних методів немає. Це пов’язано з тим, що вони все ще використовують методи «один до одного», але це приховано від користувача. Проте використання колективних методів значно спрощує написання програм.

Висновок: Під час виконання комп’ютерного практикуму я закріпив про MPI та його колективні методи, зі зв’язком «один до багатьох», «багато до одного» та «багато до багатьох». Реалізував з допомогою MPJ Express множення матриць з допомогою блокуючих колективних методів та дослідив алгоритм цього множення, збільшуючи кількість вузлів та розмір матриці, і порівнюючи його з блокуючими методами «один до одного».