Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

**Комп’ютерний практикум № 7**

з дисципліни «Технології паралельних обчислень»

**Тема:** «Розробка паралельного алгоритму множення матриць з використанням MPI-методів колективного обміну повідомленнями («один-до-багатьох», «багато-до-багатьох», «багато-до-багатьох»)»

|  |  |
| --- | --- |
| **Виконав:**  студент групи ІТ-04  Чапча Святослав  Дата здачі \_\_\_\_\_\_\_\_  Захищено з балом \_\_\_\_\_\_\_ | **Перевірила:**  ас. кафедри ІПІ  Дифучина О.Ю. |

Київ 2023

**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.**

Лістинг програми:

**Main.java**

import mpi.MPI;  
import mpi.MPIException;  
  
public class Main {  
 public static int *MASTER* = 0;  
 public static int *matrixSize* = 2000;  
 public static void main(String[] args) {  
 double[][] matrixA = new double[*matrixSize*][*matrixSize*];  
 double[][] matrixB = new double[*matrixSize*][*matrixSize*];  
 double[][] result = new double[*matrixSize*][*matrixSize*];  
 long startTime = System.*currentTimeMillis*();  
 long endTime = System.*currentTimeMillis*();  
  
 MPI.*Init*(args);  
  
 int rank = MPI.*COMM\_WORLD*.Rank();  
 int size = MPI.*COMM\_WORLD*.Size();  
  
 if(rank == *MASTER*){  
 System.*out*.println("Started with " + size + " tasks");  
 matrixA = Methods.*generateMatrix*(*matrixSize*);  
 matrixB = Methods.*generateMatrix*(*matrixSize*);  
 startTime = System.*currentTimeMillis*();  
 }  
  
 int work = *matrixSize* / size;  
 int extra = *matrixSize* % size;  
  
 int[] sendcount = new int[size];  
 int[] displace = new int[size];  
  
 displace[0] = 0;  
 for(int i = 1; i < size; i++){  
 sendcount[i-1] = work;  
 if(i == size - 1){  
 sendcount[i-1] += extra;  
 }  
 displace[i] = sendcount[i-1] + displace[i-1];  
 }  
 sendcount[size-1] = work;  
  
 int currentWork = sendcount[rank];  
  
 double[][] blockA = new double[currentWork][*matrixSize*];  
 MPI.*COMM\_WORLD*.Scatterv(matrixA, 0, sendcount, displace, MPI.*OBJECT*, blockA, 0, currentWork, MPI.*OBJECT*, 0);  
 MPI.*COMM\_WORLD*.Bcast(matrixB, 0, matrixB.length, MPI.*OBJECT*, 0);  
  
 MPI.*COMM\_WORLD*.Barrier();  
 System.*out*.println("Row start: " + displace[rank] + " Row finish: " + (displace[rank]+sendcount[rank]) + " From task " + rank);  
 double[][] subResult = Methods.*multiply*(blockA, matrixB);  
  
 MPI.*COMM\_WORLD*.Gatherv(subResult, 0, subResult.length, MPI.*OBJECT*, result, 0, sendcount, displace, MPI.*OBJECT*, 0);  
  
 if(rank == *MASTER*){  
 endTime = System.*currentTimeMillis*();  
 //Methods.print(result);  
 System.*out*.println("Time: " +(endTime-startTime) + " ms");  
 }  
 MPI.*Finalize*();  
 }  
}

**Methods.java**

public class Methods {  
 public static double[][] generateMatrix(int size){  
 double[][] matrix = new double[size][size];  
 for (int i = 0; i < size; i++) {  
 double[] row = new double[size];  
 for (int j = 0; j < size; j++) {  
 row[j] = 1;  
 }  
 matrix[i] = row;  
 }  
 return matrix;  
 }  
  
 public static double[][] multiply(double[][] matrix1, double[][] matrix2){  
 double[][] result = new double[matrix1.length][matrix2[0].length];  
 for (int i = 0; i < matrix1.length; i++) {  
 for (int j = 0; j < matrix1[0].length; j++) {  
 double sum = 0;  
 for (int k = 0; k < matrix1[0].length; k++) {  
 sum += matrix1[i][k] \* matrix2[k][j];  
 }  
 result[i][j] = sum;  
 }  
 }  
 return result;  
 }  
   
 public static void print(double[][] matrix){  
 for(int i = 0 ;i < matrix.length; i++){  
 for(int j = 0; j < matrix[i].length; j++){  
 System.*out*.print(matrix[i][j] + " ");  
 }  
 System.*out*.println();  
 }  
 }  
}

Результат виконання програми:

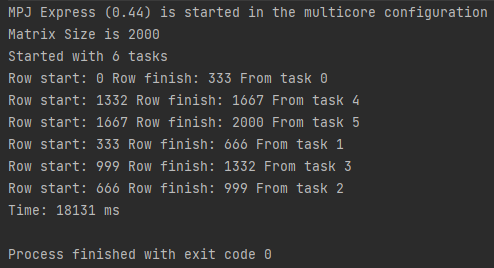


Рис. 1. Результат виконання програми

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-до-одного», «один-до-багатьох», «багато-до-одного», «багато-до-багатьох».**60 балів.**

Час виконання колективних методів

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Naive | 4 | | | 6 | | | 8 | | |
| time | diff | eff | time | diff | eff | time | diff | eff |
| 500 | 0.151 | 0.122 | 1.238 | 0.309 | 0.122 | 1.238 | 0.309 | 0.148 | 1.020 | 0.127 |
| 1000 | 1.317 | 0.96 | 1.372 | 0.343 | 0.927 | 1.421 | 0.237 | 0.836 | 1.575 | 0.197 |
| 2000 | 41.09 | 13.134 | 3.128 | 0.782 | 18.131 | 2.267 | 0.378 | 10.064 | 4.083 | 0.51 |
| 3000 | 223.373 | 62.894 | 3.552 | 0.888 | 53.296 | 4.191 | 0.698 | 49.535 | 4.509 | 0.564 |

Графік прискорення колективних методів:

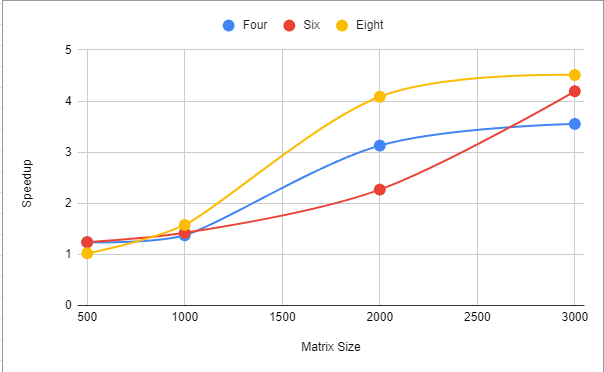


Рис. 2. Графік прискорення колективних методів

Графік прискорення блокуючих методів з шостого практикуму:

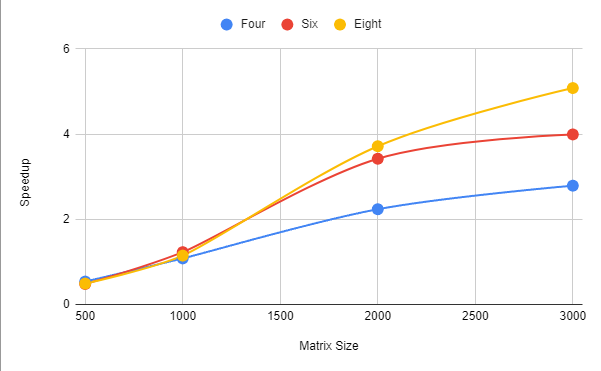


Рис. 3. Графік прискорення блокуючих методів

Графік ефективності колективних методів:

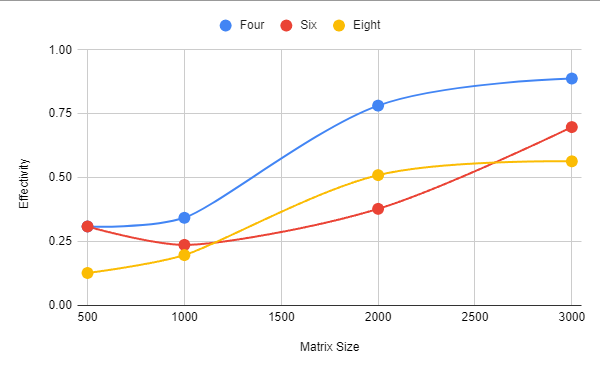


Рис. 3. Графік ефективності колективних методів

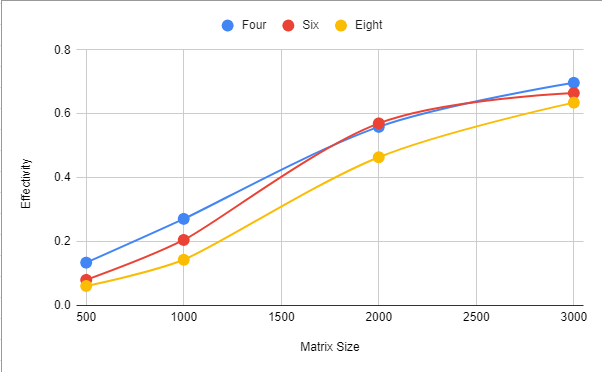
Графік ефективності блокуючих методів з шостого практикуму:  


Рис. 4. Графік ефективності блокуючих методів

Програмний код до всіх завдань наведений в репозиторії:

<https://github.com/whitetark/multithreading/tree/main/lab07>

**Висновок**

У ході виконання моєї лабораторної роботи я познайомився з методами alltoall, gather, scatter обміну повідомленнями типу collective та дослідив їх ефективність при збільшенні розміру матриць та при збільшенні кількості вузлів.