Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп'ютерних технологій, автоматики та метрології



**Лабораторна робота №4**

з дисципліни “Паралельні та розподілені обчислення”

на тему “МНОЖЕННЯ МАТРИЦЬ НА КІЛЬЦЕВІЙ СТРУКТУРІ ЗАСОБАМИ MPI”

Варіант 2

Виконав:

ст. гр. КІ-303

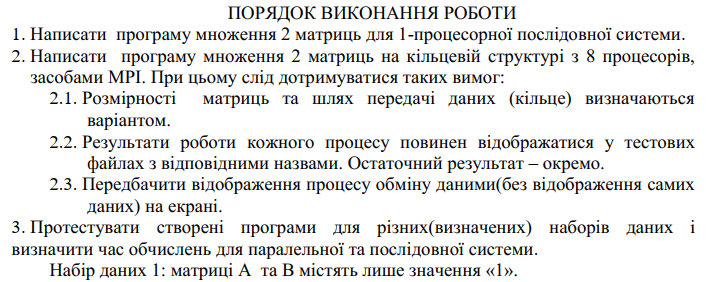
Білецький М.М.

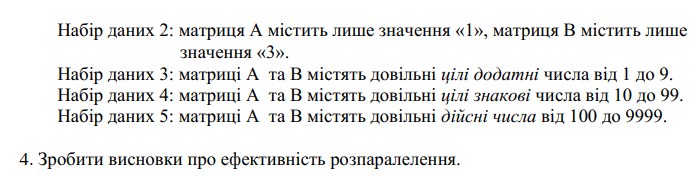
Перевірила:

Ногаль М. В.

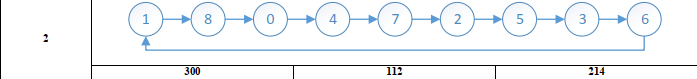
Львів-2024

МЕТА РОБОТИ. Засвоєння основних методів та алгоритмів моделювання  
паралельних обчислювальних процесів, принципів побудови відповідних структур, набуття початкових практичних навиків проектування таких засобів та ознайомлення з бібліотекою MPI.





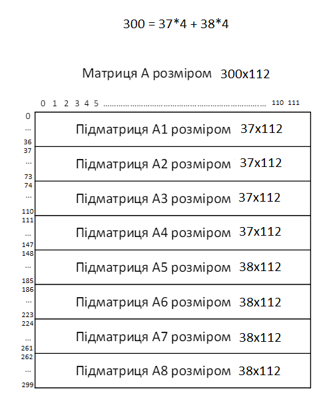
**Варіант: 2**

****

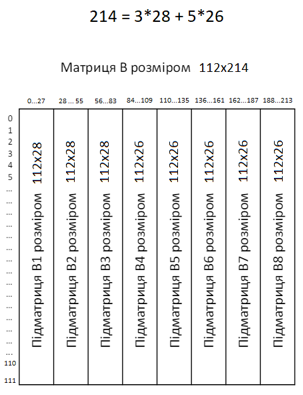
**ХІД РОБОТИ**

*Склав схему роботи кільцевої структури відповідно до варіанту:*

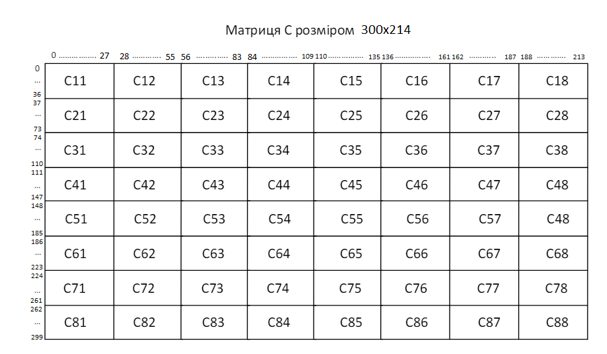
N1 = 300, N2 = 112, N3 = 214C[300][214] =А[300][112] \* B[112][214]  
Визначення розмірів підматриць:  
Для матриці А поділ відбувається по рядках. Маємо:  
300/8=37(4-остача), тобто 300=4\*37+4\*38.  
Таким чином 4 підматриці А будуть мати по 37 рядків і 4 підматриці А будуть  
мати по 38 рядків. Кількість стовпців- 112.  
Для матриці B поділ відбувається по стовпцях. Маємо:  
214/8=26(6-остача), тобто 214=3\*28+5\*26.  
Таким чином 3 підматриці B будуть мати по 28 стовпців і 5 підматриць B  
будуть мати по 26 стовпців. Кількість рядків - 112.

****

*Рис. 1. Схема матриці A*

******

*Рис. 2. Схема матриці B*

******

*Рис. 3. Схема матриці C*

ПОСЛІДОВНІСТЬ ОБЧИСЛЕННЯ МАТРИЦІ C



КРОК 1:



КРОК 2:



КРОК 3:



КРОК 4:



КРОК 5:



КРОК 6:



КРОК 7:



КРОК 8:



***Код послідовної програми:***

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <time.h>

#include <windows.h>

#define size1 300

#define size2 112

#define size3 214

double\*\* CreateMatrix(int m, int n)

{

double\*\* a = (double\*\*)malloc(m \* sizeof(double\*));

for (int i = 0; i < m; i++)

a[i] = (double\*)malloc(n \* sizeof(double));

return a;

}

void DeleteMatrix(double\*\* a, int m, int n)

{

for (int i = 0; i < m; i++)

free(a[i]);

free(a);

}

void FillMatrix(double\*\* a, int m, int n, int mode)

{

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

{

if (mode >= 0)

{

a[i][j] = mode;

}

else if (mode == -1)

{

a[i][j] = 1 + (rand() % 9);

}

else if (mode == -2)

{

a[i][j] = (10 + (rand() % 90)) \* (rand() % 2 == 0 ? 1 : -1);

}

else

{

a[i][j] = (100 + (rand() % 9900)) + (rand() / (double)RAND\_MAX);

}

}

}

void PrintMatrixToFile(FILE\* f, char\* title, double\*\* a, int m, int n)

{

fprintf(f, title);

fprintf(f, "\n");

for (int i = 0; i < m; i++)

{

for (int j = 0; j < n; j++)

fprintf(f, "%.0f\t", a[i][j]);

fprintf(f, "\n");

}

fprintf(f, "\n");

}

// c = a \* b, a - [m][n], b - [n][k], c - [m][k]

double\*\* MultMatrix(double\*\* c, double\*\* a, double\*\* b, int m, int n, int k)

{

for (int i = 0; i < m; i++)

for (int j = 0; j < k; j++)

{

c[i][j] = 0;

for (int l = 0; l < n; l++)

c[i][j] = c[i][j] + a[i][l] \* b[l][j];

}

return c;

}

int main(int argc, char\* argv[])

{

srand((unsigned int)time(NULL));

LARGE\_INTEGER frequency, start, end;

double\*\* A = CreateMatrix(size1, size2);

double\*\* B = CreateMatrix(size2, size3);

double\*\* C = CreateMatrix(size1, size3);

FILE\* file = fopen("Results.txt", "w");

if (file == NULL)

{

printf("\nError opening file. Check the path and permissions !\n");

return 1;

}

static int modes[2];

printf("Available modes:\n");

printf("0+ fill with 'mode' value\n");

printf("-1 fill with random ints from 1 to 9\n");

printf("-2 fill with random ints from (-)10 to (-)99\n");

printf("-3 fill with random doubles from 100 to 9999\n");

printf("Select A matrix mode:\n");

fflush(stdout);

scanf("%d", &modes[0]);

printf("Select B matrix mode:\n");

fflush(stdout);

scanf("%d", &modes[1]);

// random input matrix A

FillMatrix(A, size1, size2, modes[0]);

PrintMatrixToFile(file, (char\*)"Matrix A = ", A, size1, size2);

// random input matrix B

FillMatrix(B, size2, size3, modes[1]);

PrintMatrixToFile(file, (char\*)"Matrix B = ", B, size2, size3);

QueryPerformanceFrequency(&frequency);

QueryPerformanceCounter(&start);

MultMatrix(C, A, B, size1, size2, size3);

QueryPerformanceCounter(&end);

double elapsedTime = static\_cast<double>(end.QuadPart - start.QuadPart) /

frequency.QuadPart \* 1000;

printf("\nElapsed Time: %f milliseconds\n\n", elapsedTime);

PrintMatrixToFile(file, (char\*)"Matrix C = ", C, size1, size3);

DeleteMatrix(A, size1, size2);

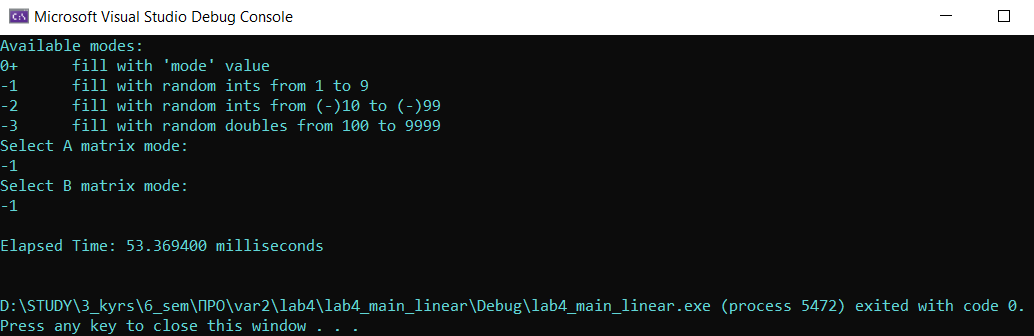
DeleteMatrix(B, size2, size3);

DeleteMatrix(C, size1, size3);

fclose(file);

return 0;

}



***Код програми на основі MPI:***

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "mpi.h"

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <time.h>

#include <windows.h>

#define size1 300

#define size2 112

#define size3 214

const int ROW1 = 37;

const int ROW2 = 38;

const int ROW1COUNT = 4;

const int COLUMN1 = 28;

const int COLUMN2 = 26;

const int COLUMN1COUNT = 3;

const int MAX\_COLUMN\_SIZE = (COLUMN1 > COLUMN2) ? COLUMN1 : COLUMN2;

const int ANOTHER\_COLUMN\_SIZE\_INDEX = COLUMN1 \* COLUMN1COUNT;

double\*\* CreateMatrix(int m, int n)

{

double\*\* a = (double\*\*)malloc(m \* sizeof(double\*));

for (int i = 0; i < m; i++)

a[i] = (double\*)malloc(n \* sizeof(double));

return a;

}

void DeleteMatrix(double\*\* a, int m, int n)

{

for (int i = 0; i < m; i++)

free(a[i]);

free(a);

}

void FillMatrixA(double\*\* a, int m, int n, int mode)

{

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

{

if (mode == 1)

{

a[i][j] = 1;

}

else if (mode == 2)

{

a[i][j] = 1;

}

else if (mode == 3)

{

a[i][j] = 1 + (rand() % 9);

}

else if (mode == 4)

{

a[i][j] = (10 + (rand() % 90)) \* (rand() % 2 == 0 ? 1 : -1);

}

else

{

a[i][j] = (100 + (rand() % 9900)) + (rand() / (double)RAND\_MAX);

}

}

}

void FillMatrixB(double\*\* a, int m, int n, int mode)

{

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

{

if (mode == 1)

{

a[i][j] = 1;

}

else if (mode == 2)

{

a[i][j] = 3;

}

else if (mode == 3)

{

a[i][j] = 1 + (rand() % 9);

}

else if (mode == 4)

{

a[i][j] = (10 + (rand() % 90)) \* (rand() % 2 == 0 ? 1 : -1);

}

else

{

a[i][j] = (100 + (rand() % 9900)) + (rand() / (double)RAND\_MAX);

}

}

}

void PrintMatrixToFile(FILE \* f, char\* title, double\*\* a, int m, int n)

{

fprintf(f, title);

fprintf(f, "\n");

for (int i = 0; i < m; i++)

{

for (int j = 0; j < n; j++)

fprintf(f, "%.0f\t", a[i][j]);

fprintf(f, "\n");

}

fprintf(f, "\n");

}

// c = a \* b, a - [m][n], b - [n][k], c - [m][k]

double\*\* MultMatrix(double\*\* c, double\*\* a, double\*\* b, int m, int n, int k)

{

for (int i = 0; i < m; i++)

for (int j = 0; j < k; j++)

{

c[i][j] = 0;

for (int l = 0; l < n; l++)

c[i][j] = c[i][j] + a[i][l] \* b[l][j];

}

return c;

}

int main(int argc, char\* argv[])

{

srand((unsigned int)time(NULL));

MPI\_Init(NULL, NULL);

LARGE\_INTEGER frequency, start, end;

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if (size != 9)

{

printf("This program requires 9 processes !\n");

fflush(stdout);

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

int i, j, k;

int shift;

int out;

char SubMatrix[32];

char fileName[32];

if (rank == 0)

{

double\*\* A = CreateMatrix(size1, size2);

double\*\* B = CreateMatrix(size2, size3);

double\*\* C = CreateMatrix(size1, size3);

sprintf(fileName, "Results\_0.txt");

FILE\* file = fopen(fileName, "w");

if (file == NULL)

{

printf("\nError opening file. Check the path and permissions !\n");

return 1;

}

printf("Output the results to the screen and to the file? 1 - yes, else -no : ");

fflush(stdout);

scanf("%d", &out);

for (i = 1; i <= 8; i++)

MPI\_Send(&out, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

static int mode;

printf("Available modes:\n");

printf("1 fill A with 1, B with 1\n");

printf("2 fill A with 1, B with 3\n");

printf("3 fill with random ints from 1 to 9\n");

printf("4 fill with random ints from (-)10 to (-)99\n");

printf("5 fill with random doubles from 100 to 9999\n");

printf("Select mode:\n");

fflush(stdout);

scanf("%d", &mode);

// random input matrix A

FillMatrixA(A, size1, size2, mode);

PrintMatrixToFile(file, (char\*)"Matrix A = ", A, size1, size2);

// random input matrix B

FillMatrixB(B, size2, size3, mode);

PrintMatrixToFile(file, (char\*)"Matrix B = ", B, size2, size3);

QueryPerformanceFrequency(&frequency);

QueryPerformanceCounter(&start);

// розсилання підматриць А і В

int A\_rows;

j = 0;

int B\_columns;

int B\_submatrix;

shift = 0;

for (i = 1; i <= 8; i++)

{

if (i <= ROW1COUNT)

A\_rows = ROW1;

else

A\_rows = ROW2;

if (out == 1)

{

printf("I am 0 process. Send SubMatrix A%d in size %d x 112 to %d process\n", i, A\_rows, i);

fflush(stdout);

}

MPI\_Send(&A\_rows, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

for (k = 0; k < A\_rows; k++, j++)

MPI\_Send(A[j], size2, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD);

if (i <= COLUMN1COUNT)

B\_columns = COLUMN1;

else

B\_columns = COLUMN2;

B\_submatrix = i;

if (out == 1)

{

printf("I am 0 process. Send SubMatrix B%d in size 112 x %d to %d process\n", B\_submatrix, B\_columns, i);

fflush(stdout);

}

MPI\_Send(&B\_submatrix, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

MPI\_Send(&B\_columns, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

for (k = 0; k < size2; k++)

MPI\_Ssend((B[k] + shift), B\_columns, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD);

shift += B\_columns;

}

// отримання частини матриці С після першого кроку

j = 0;

shift = 0;

for (i = 1; i <= 8; i++)

{

MPI\_Recv(&A\_rows, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&B\_columns, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

for (k = 0; k < A\_rows; k++, j++)

MPI\_Recv((C[j] + shift), B\_columns, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

shift += B\_columns;

if (out == 1)

{

printf("I am 0 process. Received SubMatrix C%d%d from %d process\n", i, i, i);

fflush(stdout);

}

}

// буфер для отримання смуги матриці В від 1 процесу

double\*\* b\_temp = CreateMatrix(size2, MAX\_COLUMN\_SIZE);

for (i = 2; i <= 8; i++)

{

// отримання смуги матриці В від 8-го процесу

MPI\_Recv(&B\_submatrix, 1, MPI\_INT, 8, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&B\_columns, 1, MPI\_INT, 8, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

for (j = 0; j < size2; j++)

MPI\_Recv(b\_temp[j], B\_columns, MPI\_DOUBLE, 8, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

if (out == 1)

{

printf("I am 0 process. Received SubMatrix B%d from %d process\n", B\_submatrix, 1);

fflush(stdout);

printf("I am 0 process. Send SubMatrix B%d to %d process\n",

B\_submatrix, 2);

fflush(stdout);

}

// пересилання смуги матриці 4-му процесу

MPI\_Send(&B\_submatrix, 1, MPI\_INT, 4, 0, MPI\_COMM\_WORLD);

MPI\_Send(&B\_columns, 1, MPI\_INT, 4, 0, MPI\_COMM\_WORLD);

for (j = 0; j < size2; j++)

MPI\_Send(b\_temp[j], B\_columns, MPI\_DOUBLE, 4, 0, MPI\_COMM\_WORLD);

// отримання частин матриці С після і-го кроку

j = 0;

for (int ii = 1; ii <= 8; ii++)

{

MPI\_Recv(&A\_rows, 1, MPI\_INT, ii, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&B\_columns, 1, MPI\_INT, ii, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&B\_submatrix, 1, MPI\_INT, ii, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

if (B\_submatrix <= 3)

shift = (B\_submatrix - 1) \* B\_columns;

else

shift = ANOTHER\_COLUMN\_SIZE\_INDEX + (B\_submatrix - 4) \* B\_columns;

for (k = 0; k < A\_rows; k++, j++)

MPI\_Recv((C[j] + shift), B\_columns, MPI\_DOUBLE, ii, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

if (out == 1)

{

printf("I am 0 process. Received SubMatrix C%d%d from %d process\n", ii, B\_submatrix, ii);

fflush(stdout);

}

}

}

QueryPerformanceCounter(&end);

double elapsedTime = static\_cast<double>(end.QuadPart - start.QuadPart) / frequency.QuadPart \* 1000;

printf("\nElapsed Time: %f milliseconds\n\n", elapsedTime);

PrintMatrixToFile(file, (char\*)"Matrix C = ", C, size1, size3);

DeleteMatrix(A, size1, size2);

DeleteMatrix(B, size2, size3);

DeleteMatrix(C, size1, size3);

DeleteMatrix(b\_temp, size2, MAX\_COLUMN\_SIZE);

fclose(file);

}

else

{

sprintf(fileName, "Results\_%d.txt", rank);

FILE\* file = fopen(fileName, "w");

if (file == NULL)

{

printf("Error opening file. Check the path and permissions !\n");

return 1;

}

MPI\_Recv(&out, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

// отримання поточним процесом смуги матриці А

int A\_rows;

MPI\_Recv(&A\_rows, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

double\*\* a = CreateMatrix(A\_rows, size2);

for (i = 0; i < A\_rows; i++)

MPI\_Recv(a[i], size2, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

if (out == 1)

{

printf("I am %d process. Received SubMatrix A%d in size %d x 112 from %d process\n", rank, rank, A\_rows, 0);

fflush(stdout);

sprintf(SubMatrix, "SubMatrix A%d = ", rank);

PrintMatrixToFile(file, SubMatrix, a, A\_rows, size2);

}

// отримання поточним процесом смуги матриці B

double\*\* b = CreateMatrix(size2, MAX\_COLUMN\_SIZE);

int B\_columns, B\_submatrix;

MPI\_Recv(&B\_submatrix, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&B\_columns, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

for (i = 0; i < size2; i++)

MPI\_Recv(b[i], B\_columns, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

if (out == 1)

{

printf("I am %d process. Received SubMatrix B%d in size 112 x %d from %d process\n", rank, B\_submatrix, B\_columns, 0);

fflush(stdout);

sprintf(SubMatrix, "SubMatrix B%d = ", B\_submatrix);

PrintMatrixToFile(file, SubMatrix, b, size2, B\_columns);

}

double\*\* c = CreateMatrix(A\_rows, MAX\_COLUMN\_SIZE);

// множення смуги матриці А на смугу матриці В

MultMatrix(c, a, b, A\_rows, size2, B\_columns);

if (out == 1)

{

printf("I am %d process. Multiplied SubMatrix A%d by B%d = SubMatrix C %d %d \n", rank, rank, B\_submatrix, rank, B\_submatrix);

fflush(stdout);

sprintf(SubMatrix, "SubMatrix C%d%d = A%d x B%d ", rank, B\_submatrix, rank, B\_submatrix);

PrintMatrixToFile(file, SubMatrix, c, A\_rows, B\_columns);

printf("I am %d process. Send SubMatrix C%d%d to %d process\n", rank, rank, B\_submatrix, 0);

fflush(stdout);

}

// надсилання частини матриці С після першого кроку

MPI\_Send(&A\_rows, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

MPI\_Send(&B\_columns, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

for (i = 0; i < A\_rows; i++)

MPI\_Ssend(c[i], B\_columns, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD);

int from, to;

switch (rank)

{

case 1: from = 6; to = 8; break;

case 2: from = 7; to = 5; break;

case 3: from = 5; to = 6; break;

case 4: from = 0; to = 7; break;

case 5: from = 2; to = 3; break;

case 6: from = 3; to = 1; break;

case 7: from = 4; to = 2; break;

case 8: from = 1; to = 0; break;

}

for (i = 2; i <= 8; i++)

{

// пересилання смуги матриці В від поточного процесу до процесу з рангом to

if (out == 1)

{

printf("I am %d process. Send SubMatrix B%d to %d process\n",

rank, B\_submatrix, to);

fflush(stdout);

}

MPI\_Send(&B\_submatrix, 1, MPI\_INT, to, 0, MPI\_COMM\_WORLD);

MPI\_Send(&B\_columns, 1, MPI\_INT, to, 0, MPI\_COMM\_WORLD);

for (j = 0; j < size2; j++)

MPI\_Send(b[j], B\_columns, MPI\_DOUBLE, to, 0, MPI\_COMM\_WORLD);

// отримання смуги матриці В від процесу з рангом from поточним процесом

MPI\_Recv(&B\_submatrix, 1, MPI\_INT, from, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&B\_columns, 1, MPI\_INT, from, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

for (j = 0; j < size2; j++)

MPI\_Recv(b[j], B\_columns, MPI\_DOUBLE, from, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

if (out == 1)

{

printf("I am %d process. Received SubMatrix B%d from %d process\n", rank, B\_submatrix, from);

fflush(stdout);

sprintf(SubMatrix, "SubMatrix B%d = ", B\_submatrix);

PrintMatrixToFile(file, SubMatrix, b, size2, B\_columns);

}

// множення смуги матриці А на смугу матриці В

MultMatrix(c, a, b, A\_rows, size2, B\_columns);

if (out == 1)

{

printf("I am %d process. Multiplied SubMatrix A%d by B%d = SubMatrix C %d %d \n", rank, rank, B\_submatrix, rank, B\_submatrix);

fflush(stdout);

sprintf(SubMatrix, "SubMatrix C%d%d = A%d x B%d ", rank,

B\_submatrix, rank, B\_submatrix);

PrintMatrixToFile(file, SubMatrix, c, A\_rows, B\_columns);

}

// надсилання частини матриці С після і-го кроку

MPI\_Send(&A\_rows, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

MPI\_Send(&B\_columns, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

MPI\_Send(&B\_submatrix, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

for (j = 0; j < A\_rows; j++)

MPI\_Send(c[j], B\_columns, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD);

if (out == 1)

{

printf("I am %d process. Sent SubMatrix C%d%d to %d process\n", rank, rank, B\_submatrix, 0);

fflush(stdout);

}

}

DeleteMatrix(a, A\_rows, size2);

DeleteMatrix(b, size2, MAX\_COLUMN\_SIZE);

DeleteMatrix(c, A\_rows, MAX\_COLUMN\_SIZE);

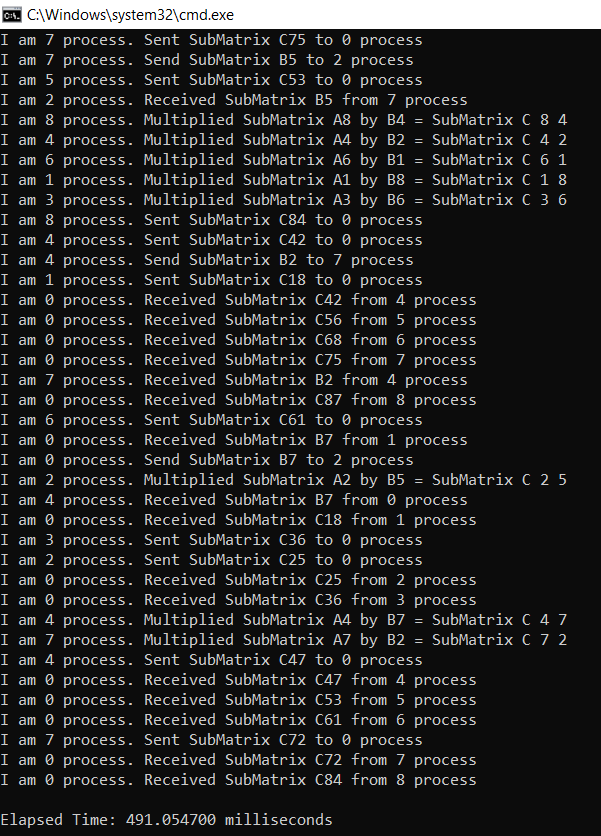
fclose(file);

}

MPI\_Finalize();

return 0;

}



Тестування обох програм на різних наборах даних та відповідні графіки залежності часу виконання програми від типу даних

|  |  |  |
| --- | --- | --- |
| № набору | Час виконання, мс | |
| Послідовна програма | MPI програма |
| 1 | 53.833100 | 447.919400 |
| 2 | 53.670500 | 441.431800 |
| 3 | 54.004700 | 449.539900 |
| 4 | 53.671600 | 470.134200 |
| 5 | 54.006500 | 503.244500 |

Збільшимо розмірності матриць у 10 раз, тобто C[3000][2140] =А[3000][1120] \* B[1120][2140] і протестуємо програми ще раз.

|  |  |  |
| --- | --- | --- |
| № набору | Час виконання, мс | |
| Послідовна програма | MPI програма |
| 1 | 158757,361800 | 3557,625600 |
| 2 | 158196,481800 | 3584,964800 |
| 3 | 158464,559400 | 3634,609100 |
| 4 | 157293,923700 | 3942,978800 |
| 5 | 158456,482400 | 3903,341200 |

**Висновок:** під час виконання даної лабораторної роботи я засвоїв основні методи та алгоритми моделювання паралельних обчислювальних процесів, принципи побудови відповідних структур, набув початкових практичних навиків проектування таких засобів та ознайомився з бібліотекою MPI, а також написав програму, яка виконує паралельне множення двох матриць на кільцевій структурі. Помітив, що під час обчислення матриць з малими розмірностями лінійна програма працювала швидше ніж програма з використанням mpi. Але тільки збільшивши розмірність в 10x mpi-програма почала значно швидше обчислювати матриці.