

*Рисунок 3.2 – Структурна схема взаємодії задач для ПРГ*

**Лістинг ПРГ2**

/\*

\* PRG2.cpp

\* A = sotr(B \* MO + C \* MX \* MZ)

\* Cherednichenko Svyatoslav Sergiyovich, IO-01

\* 12.04.2013

\*/

#include "stdafx.h"

#include <iostream>

#include <mpi.h>

#include <time.h>

using namespace std;

const int P = 5;

int N = 400;

int H = N/P;

//Ввід вектору

int\* Input\_V(int n){

int \* result = new int[N];

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

result[i] = n;

return result;

}

//Вивід вектору

void Output\_V(int\* v){

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

cout << v[i] << " ";

cout << endl;

}

//Сума 2-х векторів

void SumV(int\* a, int\* b, int\* res, int m, int m1){

for (int i = m; i < m1; i++)

res[i] = a[i-m] + b[i-m];

}

//Добуток 2-х матриць

int\*\* MulM(int\*\* a, int\*\* b, int m, int m1){

int \*\* result = Input\_M(0, m1-m);

for (int i = m; i < m1; i++)

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

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

result[i-m][j] = result[i-m][j] + a[k][j]\*b[i-m][k];

return result;

}

//Добуток вектору на матрицю

int\* MulVM(int\* a, int\*\* b, int m, int m1){

int \* result = Input\_V(0, m1-m);

for (int i = m; i < m1; i++)

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

result[i-m] = result[i-m] + a[i] \* b[i-m][j];

return result;

}

//сортування

void Sort(int \*mas, int task){

int k;

int r;

while(1){

k=0;

for (int i=(task-1)\*H; i<(task\*H)-1; i++){

if (mas[i]<mas[i+1]){

r=mas[i];

mas[i]=mas[i+1];

mas[i+1]=r;

k++;

}

}

if (k==0) break;

}

}

int \*Merge(int \*m1, int \*m2, int l1, int l2){

int \*ret = new int[l1+l2];

int n = 0;

while (l1 && l2){

if (\*m1 < \*m2){

ret[n] = \*m1;

m1++; l1--;}

else {

ret[n] = \*m2;

m2++; l2--;}

n++;}

if (l1 == 0){

for (int i=0; i<l2; i++){

ret[n++] = \*m2++;}}

else {

for (int i=0; i<l1; i++){

ret[n++] = \*m1++;}}

return ret;}

int\* Merge(int\* left, int\* right){

int a = 0, b = 0;

int rS = sizeof (left)/sizeof(int);

int lS = sizeof (right)/sizeof(int);

int size = rS + lS;

int\* Merged = new int[size];

for (int i = 0; i < size; i++){

if (b < rS && a < lS)

if (left[a] > right[b] && b < rS)

Merged[i] = right[b++];

else

Merged[i] = left[a++];

else

if (b < rS)

Merged[i] = right[b++];

else

Merged[i] = left[a++];

}

return Merged;

}

//сортування злиттям

static void MergeSort(int\* vector, int l, int r){

int vS = sizeof (vector)/sizeof(int);

if (vS == 1)

return;

int mid = (r - l) / 2;

int\* Merged = new int[r - l];

int\* array1 = new int[mid];

int\* array2 = new int[mid];

for (int i = 0; i < r - l; i++){

if (i < mid)

array1[i] = vector[i + l];

else

array2[i - mid] = vector[i + l];

}

Merged = Merge(array1, array2);

for (int i = l; i < r; i++)

vector [i] = Merged[i-l];

}

//Ввід вектору всі елементи якого = n, розміру size

int \*alloc\_V(int size,int number) {

int \*data = (int \*)malloc(size\*sizeof(int));

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

data[i] = number;

return data;

}

int \*alloc\_V(int size) {

int \*data = (int \*)malloc(size\*sizeof(int));

return data;

}

int \*\*alloc\_M(int rows, int cols) {

int \*data = (int \*)malloc(rows\*cols\*sizeof(int));

int \*\*array= (int \*\*)malloc(rows\*sizeof(int\*));

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

array[i] = &(data[cols\*i]);

return array;

}

//Ввід Матриці всі елементи якої = n, розмірності rows x cols

int \*\*alloc\_M(int rows, int cols, int number) {

int \*data = (int \*)malloc(rows\*cols\*sizeof(int));

int \*\*array= (int \*\*)malloc(rows\*sizeof(int\*));

for (int i=0; i<rows; i++){

array[i] = &(data[cols\*i]);

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

array[i][j]=number;

}

return array;

}

//Задача №1

int task1(){

double t1, t2;

t1 = MPI\_Wtime();

cout << "task 1 started " << endl;

//Ввід вхідних данних

int \* B = alloc\_V(N,1);

int \* C = alloc\_V(N,1);

int \* A = Input\_V(0);

//Пересилка В,С в Т2

MPI\_Send(&(B[0]), N, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

MPI\_Send(&(C[0]), N, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

int\*\* MO = alloc\_M(N,N);

int\*\* MX = alloc\_M(N,N);

int\*\* MZ = alloc\_M(N,N);

//Прийом MO,MХ,MZ від Т2

MPI\_Recv(&(MO[0][0]), N\*N, MPI\_INT, 1, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Recv(&(MX[0][0]), N\*N, MPI\_INT, 1, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Recv(&(MZ[0][0]), N\*N, MPI\_INT, 1, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

//Виконання обчислень

SumV(MulVM(B, MO, 0, H), MulVM(C, MulM(MX, MZ, 0, H), 0, H), A, 0, H);

Sort(A, 0);

//Прийом резудьтатів обчислень від Т2

MPI\_Recv(&(A[H]), 2\*H-H, MPI\_INT, 1, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MergeSort(A, 0, 2\*H);

//Прийом резудьтатів обчислень від Т2

MPI\_Recv(&(A[2\*H]), N-2\*H, MPI\_INT, 1, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MergeSort(A, 0, N);

//Вивід результату

if(N<11){

Output\_V(A);

}

cout << "task 1 ended " << endl;

t2 = MPI\_Wtime();

printf( "Elapsed time is %f\n", t2 - t1 );

return 0;

}

//Задача №2

int task2(){

cout << "task 2 started " << endl;

//Ввід вхідних данних

int\*\* MO = alloc\_M(N,N,1);

int\*\* MX = alloc\_M(N,N,1);

int\*\* MZ = alloc\_M(N,N,1);

int \* A = Input\_V(0);

int \* B = alloc\_V(N);

int \* C = alloc\_V(N);

//Прийом В,С від Т1

MPI\_Recv(&(B[0]), N, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Recv(&(C[0]), N, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

//Пересилка MO,MX,MZ в Т1

MPI\_Send(&(MO[0][0]), N\*N, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

MPI\_Send(&(MX[0][0]), N\*N, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

MPI\_Send(&(MZ[0][0]), N\*N, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

//Пересилка MO,MX,MZ в Т3

MPI\_Send(&(MO[0][0]), N\*N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD);

MPI\_Send(&(MX[0][0]), N\*N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD);

MPI\_Send(&(MZ[0][0]), N\*N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD);

//Пересилка В,С в Т3

MPI\_Send(&(B[0]), N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD);

MPI\_Send(&(C[0]), N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD);

//Виконання обчисленнь

SumV(MulVM(B, MO, H, 2\*H), MulVM(C, MulM(MX, MZ, H, 2\*H), H, 2\*H), A, H, 2\*H);

Sort(A, 1);

//Пересилка резудьтатів обчислень в Т1

MPI\_Send(&(A[H]), 2\*H-H, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

MPI\_Recv(&(A[2\*H]), N-2\*H, MPI\_INT, 2, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Send(&(A[2\*H]), N-2\*H, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

cout << "task 2 ended " << endl;

return 0;

}

//Задача №3

int task3(){

cout << "task 3 started " << endl;

int\*\* MO = alloc\_M(N,N);

int\*\* MX = alloc\_M(N,N);

int\*\* MZ = alloc\_M(N,N);

int \* A = Input\_V(0);

int \* B = alloc\_V(N);

int \* C = alloc\_V(N);

//Прийом MO,MX,MZ від Т2

MPI\_Recv(&(MO[0][0]), N\*N, MPI\_INT, 1, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Recv(&(MX[0][0]), N\*N, MPI\_INT, 1, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Recv(&(MZ[0][0]), N\*N, MPI\_INT, 1, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

//Прийом В,С від Т2

MPI\_Recv(&(B[0]), N, MPI\_INT, 1, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Recv(&(C[0]), N, MPI\_INT, 1, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

//Пересилка В,С,MO,MX,MZ в Т4,Т5

for(int i = 1; i <= 2;i++){

MPI\_Send(&(MO[0][0]), N\*N, MPI\_INT, 2 + i, 0, MPI\_COMM\_WORLD);

MPI\_Send(&(MX[0][0]), N\*N, MPI\_INT, 2 + i, 0, MPI\_COMM\_WORLD);

MPI\_Send(&(MZ[0][0]), N\*N, MPI\_INT, 2 + i, 0, MPI\_COMM\_WORLD);

MPI\_Send(&(B[0]), N, MPI\_INT, 2 + i, 0, MPI\_COMM\_WORLD);

MPI\_Send(&(C[0]), N, MPI\_INT, 2 + i, 0, MPI\_COMM\_WORLD);

}

//Виконання обчисленнь

SumV(MulVM(B, MO, 2\*H, 3\*H), MulVM(C, MulM(MX, MZ, 2\*H, 3\*H), 2\*H, 3\*H), A, 2\*H, 3\*H);

Sort(A, 2);

//Прийом резудьтатів обчислень від Т4

MPI\_Recv(&(A[3\*H]), 4\*H-3\*H, MPI\_INT, 3, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MergeSort(A, 3\*H, 4\*H);

//Прийом резудьтатів обчислень від Т5

MPI\_Recv(&(A[4\*H]), N-4\*H, MPI\_INT, 4, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MergeSort(A, 3\*H, N);

//Пересилка резудьтатів обчислень в Т2

MPI\_Send(&(A[2\*H]), N-2\*H, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

cout << "task 3 ended " << endl;

return 0;

}

//Задача №4

int task4(){

cout << "task 4 started " << endl;

int\*\* MO = alloc\_M(N,N);

int\*\* MX = alloc\_M(N,N);

int\*\* MZ = alloc\_M(N,N);

int \* A = Input\_V(0);

int \* B = alloc\_V(N);

int \* C = alloc\_V(N);

//Прийом MO,MX,MZ від Т3

MPI\_Recv(&(MO[0][0]), N\*N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Recv(&(MX[0][0]), N\*N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Recv(&(MZ[0][0]), N\*N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

//Прийом В,С від Т3

MPI\_Recv(&(B[0]), N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Recv(&(C[0]), N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

//Виконання обчисленнь

SumV(MulVM(B, MO, 3\*H, 4\*H), MulVM(C, MulM(MX, MZ, 3\*H, 4\*H), 3\*H, 4\*H), A, 3\*H, 4\*H);

Sort(A, 3);

//Пересилка резудьтатів обчислень в Т3

MPI\_Send(&(A[3\*H]), 4\*H-3\*H, MPI\_INT, 2, 0, MPI\_COMM\_WORLD);

cout << "task 4 ended " << endl;

return 0;

}

//Задача №5

int task5(){

cout << "task 5 started " << endl;

int\*\* MO = alloc\_M(N,N);

int\*\* MX = alloc\_M(N,N);

int\*\* MZ = alloc\_M(N,N);

int \* A = Input\_V(0);

int \* B = alloc\_V(N);

int \* C = alloc\_V(N);

//Прийом MO,MX,MZ від Т3

MPI\_Recv(&(MO[0][0]), N\*N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Recv(&(MX[0][0]), N\*N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Recv(&(MZ[0][0]), N\*N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

//Прийом В,С від Т3

MPI\_Recv(&(B[0]), N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

MPI\_Recv(&(C[0]), N, MPI\_INT, 2, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

//Виконання обчисленнь

SumV(MulVM(B, MO, 4\*H, N), MulVM(C, MulM(MX, MZ, 4\*H, N), 4\*H, N), A, 4\*H, N);

Sort(A, 4);

//Пересилка резудьтатів обчислень в Т2

MPI\_Send(&(A[4\*H]), (N-4\*H), MPI\_INT, 2, 0, MPI\_COMM\_WORLD);

cout << "task 5 ended " << endl;

return 0;

}

void work(int argc, char \*argv[]){

/\*\*

\*\* створюється група процесів, в яку поміщаються всі процеси програми,

\*\* і створюється область зв'язку, що описується наперед визначним комунікатором MPI\_COMM\_WORLD

\*\*/

MPI\_Init(&argc, &argv);

int processID;

/\*\*

\*\* Функція повертає номер процесу, що викликав цю функцію

\*\*/

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

if (processID == 0){

task1();

}

if (processID == 1){

task2();

}

if (processID == 2){

task3();

}

if (processID == 3){

task4();

}

if (processID == 4){

task5();

}

MPI\_Finalize();

}

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

{

work(argc, argv);

cout << "enter any key to exit the program...";

cin >> N;

}