**ДОДАТКИ**

**Додаток А**



*Рисунок 1.2 – Структурна схема взаємодії задач для ПРГ1*

**Лістинг ПРГ1**

/\*

\* PRG1.cpp

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

\* Cherednichenko Svyatoslav Sergiyovich, IO-01

\* 10.04.2013

\*/

#include "stdafx.h"

#include <iostream>

#include <omp.h>

using namespace std;

const int P = 4;

int N = 0;

int H = 0;

//Ввід вектору всі елементи якого = n

int\* Input\_V(int n){

int \* result = new int[N];

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

result[i] = n;

return result;

}

//Ввід Матриці всі елементи якої = n

int\*\* Input\_M(int n){

int \*\* result = new int\*[N];

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

result[i] = new int[N];

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

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

result[i][j] = n;

return result;

}

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

int\* Input\_V(int n, int size){

int \* result = new int[size];

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

result[i] = n;

return result;

}

//Ввід Матриці всі елементи якої = n, розміру size

int\*\* Input\_M(int n, int size){

int \*\* result = new int\*[size];

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

result[i] = new int[N];

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

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

result[i][j] = n;

return result;

}

//Вивід вектору

void Output\_V(int\* v){

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

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

cout << endl;

}

//Копія Матриці

int\*\* Copy\_M(int\*\* MX){

int \*\* result = new int\*[N];

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

result[i] = new int[N];

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

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

result[i][j] = MX[i][j];

return result;

}

//Копія вектору

int\* Copy\_V(int\* V){

int\* result = new int[N];

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

result[i] = V[i];

return result;

}

//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];

}

int\*\* MZ;

int\*\* MO;

//Спільні песурси

int\*\* MX;

int\* C;

int\* B;

//резудьтат

int\* A;

//Задача №1

int task1(){

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

//синхронізація по вводу початкових даних

#pragma omp barrier

int task = 0;

int\*\* MX1;

int\* B1;

int\* C1;

//Критична секція: копіювання спільного ресурсу

#pragma omp critical

{

B1 = Copy\_V(B);

C1 = Copy\_V(C);

MX1 = Copy\_M(MX);

}

//Виконання обчислень

SumV(MulVM(B1, MO, 0, H), MulVM(C1, MulM(MX1, MZ, 0, H), 0, H), A, 0, H);

Sort(A, task);

#pragma omp barrier

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

#pragma omp barrier

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

return 0;

}

int task2(){

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

//Ввід початкових данних

B = Input\_V(1);

C = Input\_V(1);

A = Input\_V(0);

//синхронізація по вводу початкових даних

#pragma omp barrier

int task = 1;

int\*\* MX2;

int\* B2;

int\* C2;

//Критична секція: копіювання спільного ресурсу

#pragma omp critical

{

B2 = Copy\_V(B);

C2 = Copy\_V(C);

MX2 = Copy\_M(MX);

}

//Виконання обчислень

SumV(MulVM(B2, MO, H, 2\*H), MulVM(C2, MulM(MX2, MZ, H, 2\*H), H, 2\*H), A, H, 2\*H);

Sort(A, task);

#pragma omp barrier

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

#pragma omp barrier

MergeSort(A, 0, N);

//Вивід результату

if(N <= 10){

cout << endl;

Output\_V(A);

}

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

return 0;

}

int task3(){

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

//Ввід початкових данних

MO = Input\_M(1);

MX = Input\_M(1);

//синхронізація по вводу початкових даних

#pragma omp barrier

int task = 2;

int\*\* MX3;

int\* B3;

int\* C3;

//Критична секція: копіювання спільного ресурсу

#pragma omp critical

{

B3 = Copy\_V(B);

C3 = Copy\_V(C);

MX3 = Copy\_M(MX);

}

//Виконання обчислень

SumV(MulVM(B3, MO, 2\*H, 3\*H), MulVM(C3, MulM(MX3, MZ, 2\*H, 3\*H), 2\*H, 3\*H), A, 2\*H, 3\*H);

Sort(A, task);

#pragma omp barrier

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

#pragma omp barrier

return 0;

}

int task4(){

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

//Ввід початкових данних

MZ = Input\_M(1);

//синхронізація по вводу початкових даних

#pragma omp barrier

int task = 3;

int\*\* MX4;

int\* B4;

int\* C4;

//Критична секція: копіювання спільного ресурсу

#pragma omp critical

{

B4 = Copy\_V(B);

C4 = Copy\_V(C);

MX4 = Copy\_M(MX);

}

SumV(MulVM(B4, MO, 3\*H, N), MulVM(C4, MulM(MX4, MZ, 3\*H, N), 3\*H, N), A, 3\*H, N);

Sort(A, task);

#pragma omp barrier

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

#pragma omp barrier

return 0;

}

void work(){

double ctime1 = omp\_get\_wtime();

#pragma omp parallel num\_threads(4)

{

int i = omp\_get\_thread\_num();

if(i == 0)

{

task1();

}

if(i == 1)

{

task2();

}

if(i == 2)

{

task3();

}

if(i == 3)

{

task4();

}

}

double ctime2 = omp\_get\_wtime();

cout << "work time: " << (ctime2 - ctime1) << endl;

}

int \_tmain(int argc, \_TCHAR\* argv[]){

cout << "Enter N: ";

cin >> N;

cout << endl;

H = N/P;

work();

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

cin >> N;

return 0;

}