НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА АВТОМАТИЗАЦІЇ СИСТЕМ ОБРОБКИ ІНФОРМАЦІЇ ТА УПРАВЛІННЯ

Лабораторна робота №4

з дисципліни **«**Паралельне програмування**»**

Виконав:

студент 3 курсу

ФІОТ гр. ІП-31

Кобилинський Д.А.

Перевірив:

Корочкін О. В.

Київ – 2016 р.

**Тема:** Програмування для комп’ютерних систем зі спільною пам’яттю. Бібліотека OpenMP.

**Технічне завдання:** Розробити програму для розв’язання ПКС із СП (структура на рис. 1) математичної задачі: A = sort (d\*B + e\*Z(MO\*MK));

**Бібліотека: OpenMP.**

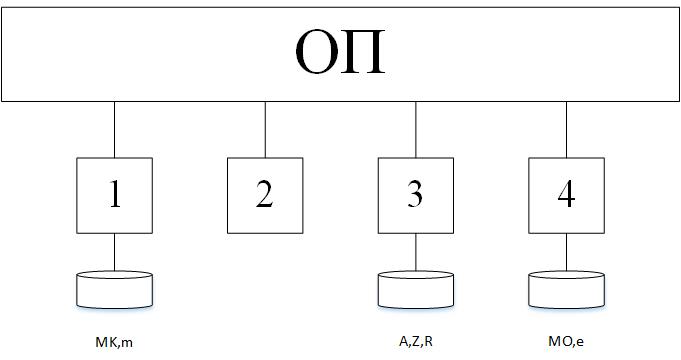


Рис.1 Структурна схема ПКС

**Виконання роботи**

**Етап 1.** Побудова паралельного алгоритму

2. A = merge(Buf);

Спільний ресурс: m, e, MO

**Етап 2. Розробка алогоритмів роботи кожного процесу**

i = (1 .. 4), tID – номер поточної задачі, j = i/tID.

**Задача Т1**

|  |  |  |
| --- | --- | --- |
| **№** | **Дія** | **ТС/КД** |
| 1. | Ввід MK,m |  |
| 2. | Надіслати сигнал Т2..n про завершення вводу |  |
| 3. | Очікувати сигнал про завершення вводу в Т2..4 |  |
| 4. | Копіювати , e1 = e, MO1 = MO | КД |
| 5. | Обчислити Buf[1] = |  |
| 6. | Надіслати сигнал T2..n про завершення обчислень |  |
| 7. | Очікувати сигнал про завершення обчислень Buf[i] інших задачах |  |

**Задача Т2**

|  |  |  |
| --- | --- | --- |
| **№** | **Дія** | **ТС/КД** |
| 1. | Надіслати сигнал Т1,3,4 про завершення вводу |  |
| 2. | Очікувати сигнал про завершення вводу в Т1,3,4 |  |
| 3. | Копіювати , e2 = e, MO2 = MO | КД |
| 4. | Обчислити Buf[2] = |  |
| 5. | Надіслати сигнал T1,3,4 про завершення обчислень |  |
| 6. | Очікувати сигнал про завершення обчислень Buf в інших задачах |  |

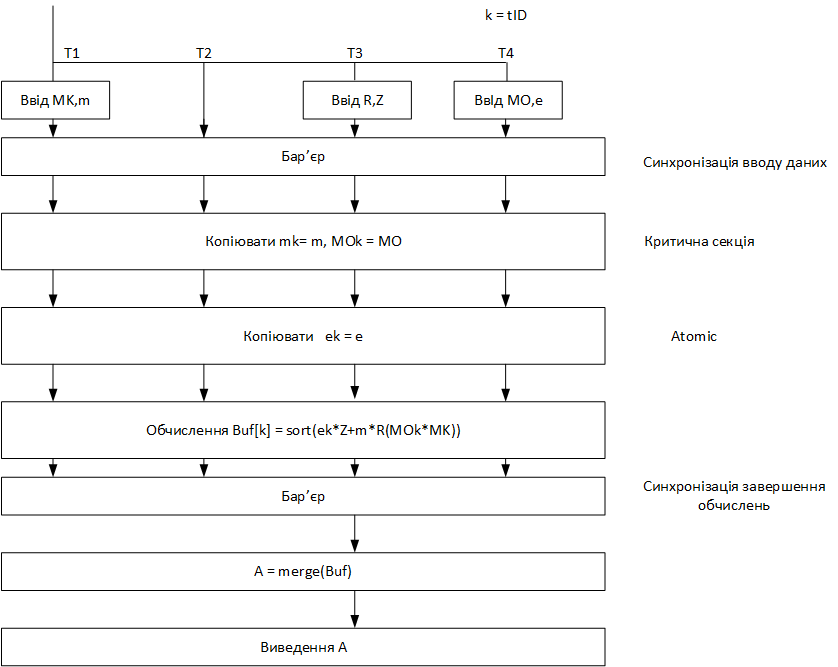
**Задача Т3**

|  |  |  |
| --- | --- | --- |
| **№** | **Дія** | **ТС/КД** |
| 1. | Ввід Z,R |  |
| 2. | Надіслати сигнал Т1,2,4 про завершення вводу |  |
| 3. | Очікувати сигнал про завершення вводу в Т1,2,4 |  |
| 4. | Копіювати , e3 = e, MO3 = MO | КД |
| 5. | Обчислити Buf[3] = |  |
| 6. | Надіслати сигнал T1,2,4 про завершення обчислень |  |
| 7. | Очікувати сигнал про завершення обчислень Buf в інших задачах |  |
| 8. | Виконати обчислення A = merge(Buf) |  |
| 9. | Вивід А |  |

**Задача Т4**

|  |  |  |
| --- | --- | --- |
| **№** | **Дія** | **ТС/КД** |
| 1. | Ввід MO,e |  |
| 2. | Надіслати сигнал Т1,2,3 про завершення вводу |  |
| 3. | Очікувати сигнал про завершення вводу в Т1..3 |  |
| 4. | Копіювати , e4 = e, MO4 = MO | КД |
| 5. | Обчислити Buf[4] = |  |
| 6. | Надіслати сигнал T1,3,4 про завершення обчислень |  |
| 7. | Очікувати сигнал про завершення обчислень Buf в інших задачах |  |

**Етап 3. Розробка структурної схеми взаємодії задач**



**Етап 4. Лістинг коду**

#include "matrixOperations.h"

#include<windows.h>

#include<iostream>

#include "omp.h"

#include <cmath>

#ifdef \_DEBUG

#undef \_DEBUG

#include <omp.h>

#define \_DEBUG

#else

#include <omp.h>

#endif

using namespace std;

/\*------------------------------------------------------------------

--

-- Parallel programming --

-- Laboratory work #4 OpenMP --

-- Task: A = sort(e\*Z+m\*R(MO\*MK)) --

-- --

-- Author: Kobylynskiy Dmytro, group IP-31 --

-- Date:06.04.2016 --

-- --

------------------------------------------------------------------ \*/

typedef int\* vector;

typedef int\*\* matrix;

int e,m;

int N,P,H;

vector A,R,Z;

matrix MO, MK;

void copySharedResources(int &copy\_m, int &copy\_e, matrix &copy\_MO){

#pragma omp critical

{

copy\_m = m;

copy\_MO = copyMatrix(MO, N);

}

#pragma omp atomic

copy\_e += e;

}

void calculateEquation(int copy\_d, int copy\_e, matrix copy\_MO, int startIndex, int endIndex){

matrix Buf = multMatr(copy\_MO, MK, startIndex, endIndex, N);

vector BufVect1 = multVectMatr(R, Buf, startIndex, endIndex, N);

addVectorsMultConstans(Z, BufVect1, A, copy\_d, copy\_e, startIndex, endIndex);

}

vector mergeParts(matrix vectorParts, vector vectorsSizes, int partsCount){

cout << "vectorsSizes" << endl;

if (partsCount == 1)

return vectorParts[0];

int vectorPairs = (int)ceil(partsCount / 2.0);

int fullIters = partsCount / 2;

matrix currentParts = vectorParts;

vector currentSizes = vectorsSizes;

while (vectorPairs > 1){

matrix nextIteration = new vector[vectorPairs];

vector nextIterationSizes = new int[vectorPairs];

cout << "parts count: " << partsCount << endl;

cout << "fullIters: " << fullIters << endl;

cout << "vectorPairs: " << vectorPairs << endl;

cout << "currentThread: " << omp\_get\_thread\_num() << endl;

omp\_set\_num\_threads(fullIters);

#pragma omp parallel for

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

int left = 2 \* i;

int right = 2 \* i + 1;

cout << omp\_get\_thread\_num() << " doing " << i << " iteration"<<endl;

nextIteration[i] = mergeVectors(currentParts[left], currentParts[right], currentSizes[left], currentSizes[right]);

nextIterationSizes[i] = currentSizes[left] + currentSizes[right];

}

if (fullIters < vectorPairs){

nextIteration[fullIters] = currentParts[partsCount - 1];

nextIterationSizes[fullIters] = currentSizes[partsCount - 1];

}

currentParts = nextIteration;

currentSizes = nextIterationSizes;

vectorPairs = (int)ceil(vectorPairs / 2.0);

fullIters = vectorPairs / 2;

}

return mergeVectors(currentParts[0], currentParts[1], currentSizes[0], currentSizes[1]);

}

void task(){

matrix sortedVectors = new vector[P];

vector sortedVectorsSizes = new int[P];

omp\_set\_num\_threads(P);

#pragma omp parallel

{

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

#pragma omp critical

cout << "Task " << tID << " started" << endl;

switch(tID){

case 0:

MK = generateMatrix(N);

m = generateConstant();

break;

case 1:

break;

case 2:

A = generateVector(N);

Z = generateVector(N);

R = generateVector(N);

break;

case 3:

e = generateConstant();

MO = generateMatrix(N);

break;

}

#pragma omp barrier

int m\_i, e\_i = 0;

matrix MO\_i = NULL;

copySharedResources(m\_i, e\_i, MO\_i);

int endIndex = -1;

int startIndex = -1;

#pragma omp for

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

startIndex = i\*H;

endIndex = (i == P - 1) ? N : (i + 1)\*H;

#pragma omp critical

cout << tID << " startIndex = " << startIndex << ", endIndex = " << endIndex << endl;

sortedVectorsSizes[i] = endIndex - startIndex;

calculateEquation(m\_i, e\_i, MO\_i, startIndex, endIndex);

sortedVectors[i] = copyVector(A, startIndex, endIndex);

sortVector(sortedVectors[i], endIndex - startIndex);

}

}

A = mergeParts(sortedVectors, sortedVectorsSizes, P);

if (N <= 8)

printVector(A, N);

}

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

{

cout << "Lab4 started" << endl;

cout << "input N" << endl;

cin >> N;

cout << "input P" << endl;

cin >> P;

H = N / P;

A = new int[N];

R = NULL;

Z = NULL;

MO = NULL;

MK = NULL;

task();

cout << "Lab4 finished" << endl;

system("pause");

return 0;

}