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

Кахерський О.І.

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.**

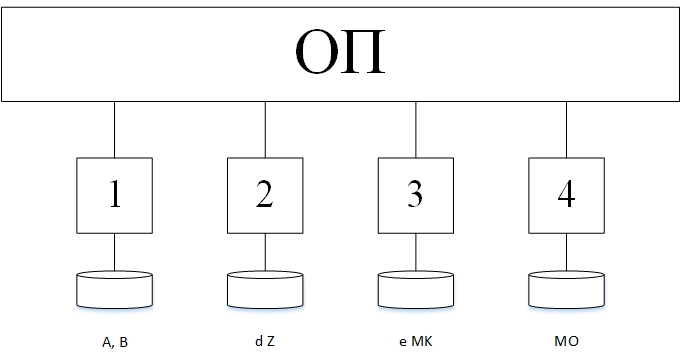


Рис.1 Структурна схема ПКС

**Виконання роботи**

**Етап 1.** Побудова паралельного алгоритму

2. A = merge(Buf);

Спільний ресурс: d, e, MO

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

i = (5 .. n), tID – номер поточної задачі, j = i/tID.

**Задача Т1**

|  |  |  |
| --- | --- | --- |
| **№** | **Дія** | **ТС/КД** |
| 1. | Ввід В |  |
| 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 |  |
| 8. | Виконати обчислення A = merge(Buf) |  |
| 9. | Вивід А |  |

**Задача Т2**

|  |  |  |
| --- | --- | --- |
| **№** | **Дія** | **ТС/КД** |
| 1. | Ввід d,Z |  |
| 2. | Надіслати сигнал Т1,3,i про завершення вводу |  |
| 3. | Очікувати сигнал про завершення вводу в Т1,3,4 |  |
| 4. | Копіювати , e2 = e, MO2 = MO | КД |
| 5. | Обчислити Buf[2] = |  |
| 6. | Надіслати сигнал T1,3..i про завершення обчислень |  |
| 7. | Очікувати сигнал про завершення обчислень Buf в інших задачах |  |

**Задача Т3**

|  |  |  |
| --- | --- | --- |
| **№** | **Дія** | **ТС/КД** |
| 1. | Ввід MO |  |
| 2. | Надіслати сигнал Т1,2,4,i 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,i про завершення обчислень |  |
| 7. | Очікувати сигнал про завершення обчислень Buf в інших задачах |  |

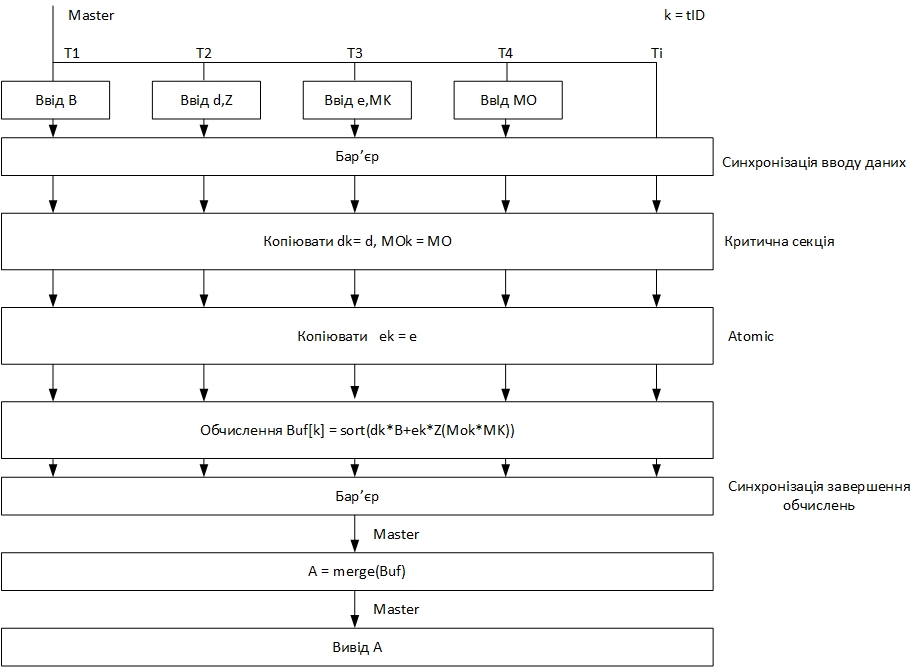
**Задача Т4**

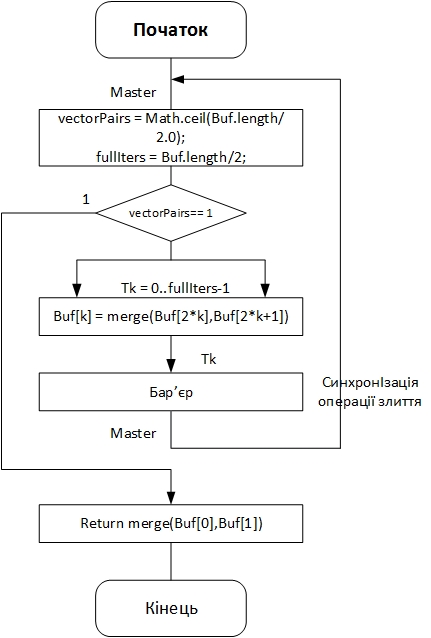
|  |  |  |
| --- | --- | --- |
| **№** | **Дія** | **ТС/КД** |
| 1. | Ввід e, MK |  |
| 2. | Надіслати сигнал Т1..3, Тi про завершення вводу |  |
| 3. | Очікувати сигнал про завершення вводу в Т1..3 |  |
| 4. | Копіювати , e4 = e, MO4 = MO | КД |
| 5. | Обчислити Buf[4] = |  |
| 6. | Надіслати сигнал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

**Задача Тi**

|  |  |  |
| --- | --- | --- |
| **№** | **Дія** | **ТС/КД** |
| 1. | Очікувати сигнал про завершення вводу в Т1..4 |  |
| 2. | Копіювати , ei = e, MOi = MO | КД |
| 3. | Обчислити Buf[i] = |  |
| 4. | Надіслати сигнал іншим задачам про завершення обчислення виразу |  |
| 5. | Очікувати сигнал про завершення обчислень Buf[i] інших задачах |  |

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





Алгоритм виконання злиття відсортованих масивів

**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;

typedef int\* vector;

typedef int\*\* matrix;

int d, e;

int N,P,H;

vector A,B,Z;

matrix MO, MK;

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

#pragma omp critical

{

copy\_d = d;

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(Z, Buf, startIndex, endIndex, N);

addVectorsMultConstans(B, 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); //количество пар векторов (+ 1 вектор без пары, если он есть)

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:

B = generateVector(N);

break;

case 1:

d = generateConstant();

Z = generateVector(N);

break;

case 2:

e = generateConstant();

MK = generateMatrix(N);

break;

case 3:

MO = generateMatrix(N);

break;

}

#pragma omp barrier

int d\_i, e\_i = 0;

matrix MO\_i = NULL;

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

int endIndex = -1;

int startIndex = -1;

#pragma omp for private(startIndex, endIndex)

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(d\_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 <= 60)

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;

//cout << N;

//cout << P;

H = N / P;

A = new int[N];

B = NULL;

Z = NULL;

MO = NULL;

MK = NULL;

task();

cout << "Lab4 finished" << endl;

system("pause");

return 0;

}

#include "omp.h"

#include<iostream>

typedef int\* vector;

typedef int\*\* matrix;

using namespace std;

/\*\*

\* метод повертає згенеровану матрицю розмірністю N\*N

\*/

matrix generateMatrix(int N){

matrix m = new vector[N];

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

m[i] = new int[N];

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

m[i][j] = 1;

}

}

return m;

}

/\*\*

\* метод повертає згенерований вектор розмірністю N

\*/

vector generateVector(int N){

vector vector = new int[N];

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

vector[i] = 1;

}

return vector;

}

/\*\*

\* метод повертає згенеровану константу

\*/

int generateConstant(){

return 1;

}

/\*\*

\* перемножає рядки matrA з діапазона [startIndex, endIndex) на matrB. Результат записується в result

\*/

matrix multMatr(matrix matrA, matrix matrB, int startIndex, int endIndex, int N){

matrix result = new vector[N];

for (int i = startIndex; i < endIndex; i++) {

result[i] = new int[N];

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

result[i][j] = 0;

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

result[i][j] += matrA[i][k] \* matrB[k][j];

}

}

}

return result;

}

/\*\*

\* Сортировка вставками вектора в діапазоні [startIndex, endIndex)

\*/

void sortVector(vector vect, int N){

for (int i = 1; i < N; ++i)

{

int buf = vect[i];

for (int j = i - 1; j >= 0; --j)

{

if(vect[j] > buf){

vect[j+1] = vect[j];

vect[j] = buf;

}

}

}

}

//слияние векторов (однопоточное)

vector mergeVectors(vector left, vector right, int leftLen, int rightLen){

vector result = new int[leftLen + rightLen];

int leftIndex = 0;

int rightIndex = 0;

int i = 0;

while(leftIndex < leftLen && rightIndex < rightLen){

if(left[leftIndex] < right[rightIndex]){

result[i++] = left[leftIndex++];

}

else{

result[i++] = right[rightIndex++];

}

}

if(leftIndex >= leftLen)

while(rightIndex < rightLen)

result[i++] = right[rightIndex++];

else

while(leftIndex < leftLen)

result[i++] = left[leftIndex++];

return result;

}

/\*\*

\* Перемножає вектор на рядки матриці в діапазоні [startIndex, endIndex)

\*/

vector multVectMatr(vector vectA, matrix matrB, int startIndex, int endIndex, int N){

vector result = new int[N];

for (int i = startIndex; i < endIndex; ++i){

result[i] = 0;

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

result[i] += vectA[j] \* matrB[i][j];

}

}

return result;

}

vector copyVector(vector vectA, int startIndex, int endIndex){

vector result = new int[endIndex - startIndex];

for (int i = startIndex; i < endIndex; ++i)

{

result[i - startIndex] = vectA[i];

}

return result;

}

matrix copyMatrix(matrix matrA, int N){

matrix result = new vector[N];

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

{

result[i] = new int[N];

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

{

result[i][j] = matrA[i][j];

}

}

return result;

}

void printVector(vector vect, int n){

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

cout << vect[i] <<" ";

cout << endl;

}

void addVectorsMultConstans(vector vect1, vector vect2, vector result, int copy\_d, int copy\_e, int startIndex, int endIndex){

for (int i = startIndex; i < endIndex; ++i){

result[i] = copy\_d \* vect1[i] + copy\_e \* vect2[i];

}

}