Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Основи паралельного програмування

Лабораторна робота №5

**«OpenMP. Бар’єри, критичні секції»**

Виконала:

студентка групи ІВ-71

Молчанова В.С.

Перевірив:

Корочкін О.В.

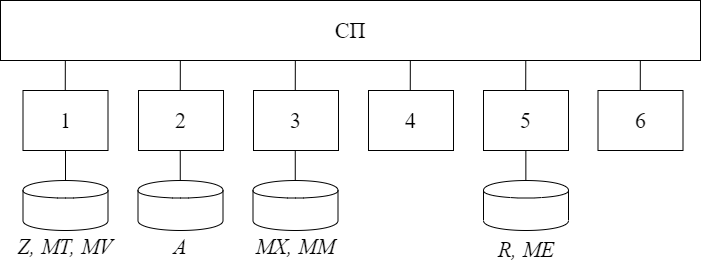
Київ

2020 р.

**Мета роботи**: розробка програми для ПКС зі СП.

## Завдання

* *Р=6*
* *А = (R\*MT)\*ME + min(R)\*(Z\*MX) + sort(Z)\*(MM\*MV)*



* Мова програмування: C++
* Засоби: бар’єри, критичні секції OpenMP.

## Виконання роботи

### 1. Паралельний алгоритм

Алгоритм обчислення виразу:

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 

Спільні ресурси: .

### 2. Алгоритми потоків

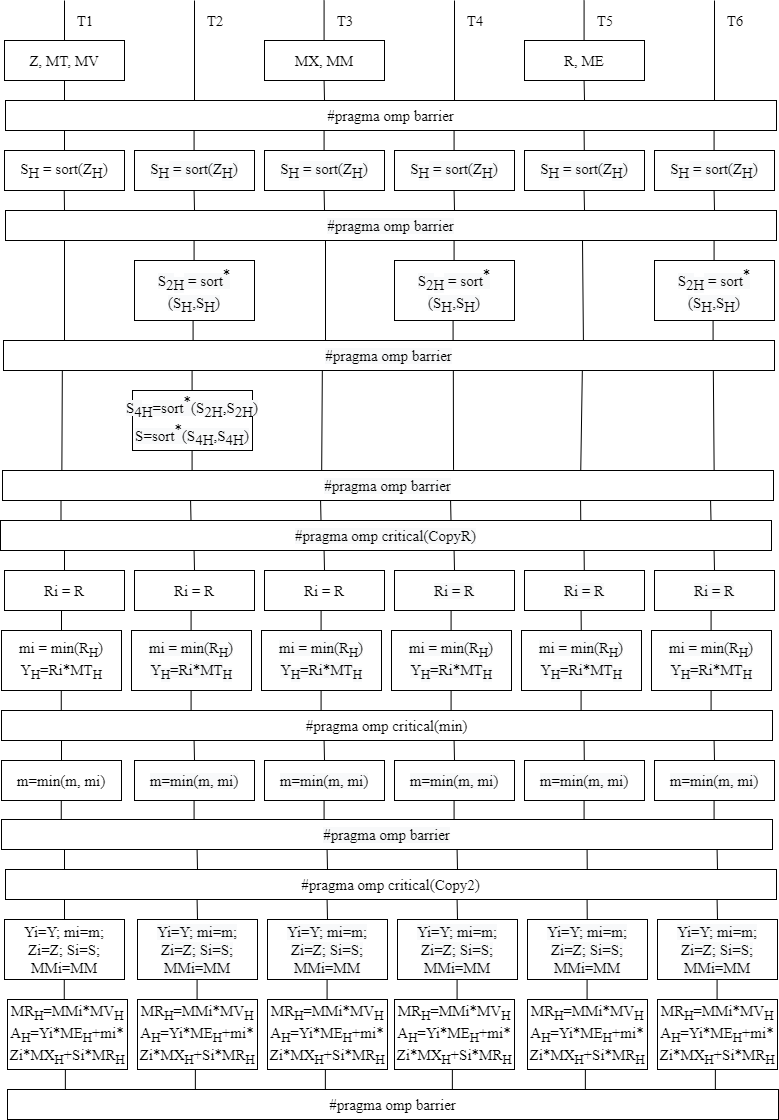
|  |  |  |
| --- | --- | --- |
|  | **Т2** | ТС КД |
| 1. | Очікувати введення даних в решті потоків |  |
| 2. |  |  |
| 3. | Очікувати завершення сортування в Т1 |  |
| 4. |  |  |
| 5. | Очікувати завершення сортування\* в Т4 |  |
| 6. |  |  |
| 7. | Очікувати завершення сортування\* в Т6 |  |
| 8. |  |  |
| 9. | Копіювати *R2=R* | КД |
| 10. |  |  |
| 11. |  |  |
| 12. | Сигнал решті потоків про завершення обчислення |  |
| 13. | Очікувати завершення обчислення в решті потоків |  |
| 14. | Копіювати *Y2=Y, m2=m, Z2=Z, S2=S, MM2=MM* | КД |
| 15. | Обчислити |  |
| 16. | Обчислити |  |
| 17. | Очікувати завершення обчислення в решті потоків |  |
| 18. | Вивести *A* |  |

|  |  |  |
| --- | --- | --- |
|  | **Т4** | ТС КД |
| 1. | Очікувати введення даних в решті потоків |  |
| 2. |  |  |
| 3. | Очікувати завершення сортування в Т3 |  |
| 4. |  |  |
| 5. | Сигнал Т2 про завершення сортування\* |  |
| 6. |  |  |
| 7. | Копіювати *R4=R* | КД |
| 8. |  |  |
| 9. |  |  |
| 10. | Сигнал решті потоків про завершення обчислення |  |
| 11. | Очікувати завершення обчислення в решті потоків |  |
| 12. | Копіювати *Y4=Y, m4=m, Z4=Z, S4=S, MM4=MM* | КД |
| 13. | Обчислити |  |
| 14. | Обчислити |  |
| 15. | Сигнал T2 про завершення обчислення |  |

|  |  |  |
| --- | --- | --- |
|  | **Т6** | ТС КД |
| 1. | Очікувати введення даних в решті потоків |  |
| 2. |  |  |
| 3. | Очікувати завершення сортування в Т5 |  |
| 4. |  |  |
| 5. | Сигнал Т2 про завершення сортування\* |  |
| 6. |  |  |
| 7. | Копіювати *R6=R* | КД |
| 8. |  |  |
| 9. |  |  |
| 10. | Сигнал решті потоків про завершення обчислення |  |
| 11. | Очікувати завершення обчислення в решті потоків |  |
| 12. | Копіювати *Y6=Y, m6=m, Z6=Z, S6=S, MM6=MM* | КД |
| 13. | Обчислити |  |
| 14. | Обчислити |  |
| 15. | Сигнал T2 про завершення обчислення |  |

|  |  |  |
| --- | --- | --- |
|  | **Тi, де** | ТС КД |
| 1. | Ввести дані:   * Якщо *i* = 1: *Z, MT, MV* * Якщо *i* = 3: *MX, MM* * Якщо *i* = 5: *R, ME* |  |
| 2. | Сигнал решті потоків про введення даних |  |
| 3. | Очікувати введення даних в решті потоків |  |
| 4. |  |  |
| 5. | Сигнал Т{*i*+1} про завершення сортування |  |
| 6. | Копіювати *Ri=R* | КД |
| 7. |  |  |
| 8. |  |  |
| 9. | Сигнал решті потоків про завершення обчислення |  |
| 10. | Очікувати завершення обчислення в решті потоків |  |
| 11. | Копіювати *Yi=Y, mi=m, Zi=Z, Si=S, MMi=MM* | КД |
| 12. | Обчислити |  |
| 13. | Обчислити |  |
| 14. | Сигнал T2 про завершення обчислення |  |

### 3. Структурна схема взаємодії потоків



### 4. Лістинг програми

#include <iostream>

#include <omp.h>

using namespace std;

const int N = 6;

const int P = 6;

const int H = N / P;

const int FILLER = 1;

int getStart(int threadNumber) {

return (threadNumber - 1) \* H;

}

int getEnd(int threadNumber) {

return (threadNumber \* H) - 1;

}

int\* inputVector() {

int\* result = new int[N];

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

result[i] = 1;

}

return result;

}

int\* copyVector(int\* vector) {

int\* result = new int[N];

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

result[i] = vector[i];

}

return result;

}

int\*\* createMatrix() {

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

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

result[i] = new int[N];

}

return result;

}

int\*\* inputMatrix() {

int\*\* result = createMatrix();

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

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

result[i][j] = 1;

}

}

return result;

}

int\*\* copyMatrix(int\*\* matrix) {

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] = matrix[i][j];

}

}

return result;

}

void outputVector(int\* vector) {

if (N < 10) {

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

int val = vector[i];

std::cout << val << " ";

}

std::cout << " \n";

}

else std::cout << vector[0] << " \n";

}

void outputMatrix(int\*\* matrix) {

if (N < 10) {

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

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

int val = matrix[i][j];

std::cout << val << " ";

}

std::cout << " \n";

}

std::cout << "\n" << endl;

}

else std::cout << matrix[0][0] << endl;;

}

void sort(int\* source, int\* target, int start, int end) {

for (int i = start; i <= end; i++)

{

target[i] = source[i];

}

for (int i = start; i <= end; i++)

{

for (int j = i; j <= end; j++)

{

if (target[j] < target[i])

{

int buf = target[j];

target[j] = target[i];

target[i] = buf;

}

}

}

}

void mergeSort(int\* vec, int start1, int end1, int end2)

{

int start2 = end1 + 1;

int i1 = start1;

int i2 = start2;

int i = 0;

int\* res = new int[end1 - start1 + end2 - start2 + 2];

while (i1 <= end1 || i2 <= end2)

{

if (vec[i1] < vec[i2])

{

res[i] = vec[i1];

i1++;

if (i1 > end1)

{

while (i2 <= end2)

{

i++;

res[i] = vec[i2];

i2++;

}

}

}

else

{

res[i] = vec[i2];

i2++;

if (i2 > end2)

{

while (i1 <= end1)

{

i++;

res[i] = vec[i1];

i1++;

}

}

}

i++;

}

i1 = 0;

for (int j = start1; j <= end2; j++)

{

vec[j] = res[i1];

i1++;

}

}

int main() {

int m = INT\_MAX;

int\* R = new int[N];

int\* Z = new int[N];

int\* S = new int[N];

int\* Y = new int[N];

int\* A = new int[N];

int\*\* ME = createMatrix();

int\*\* MT = createMatrix();

int\*\* MM = createMatrix();

int\*\* MX = createMatrix();

int\*\* MV = createMatrix();

int\*\* MR = createMatrix();

// input

#pragma omp parallel num\_threads(6)

{

int threadNumber = omp\_get\_thread\_num() + 1;

int start = getStart(threadNumber);

int end = getEnd(threadNumber);

#pragma omp critical(start)

{

cout << "Thread " << threadNumber << " started..." << endl;

}

//-----------------Input data------------------

switch (threadNumber)

{

case 1:

{

Z = inputVector();

MT = inputMatrix();

MV = inputMatrix();

break;

}

case 3:

{

MX = inputMatrix();

MM = inputMatrix();

break;

}

case 5:

{

R = inputVector();

ME = inputMatrix();

break;

}

default:

break;

}

#pragma omp barrier

//---------------S\_H = sort(Z\_H)---------------

sort(Z, S, start, end);

#pragma omp barrier

//-----------S\_2H = sort\*(S\_H,S\_H)-------------

if (threadNumber % 2 == 0) {

mergeSort(S, getStart(threadNumber - 1), getEnd(threadNumber - 1), end);

}

#pragma omp barrier

if (threadNumber == 2) {

//----------S\_4H = sort\*(S\_2H,S\_2H)------------

mergeSort(S, getStart(1), getEnd(2), end);

//-----------S = sort \* (S\_4H, S\_2H)-----------

mergeSort(S, getStart(1), end, getEnd(6));

}

#pragma omp barrier

int\* Ri;

#pragma omp critical(CopyR)

{

Ri = R;

}

//----------------mi = min(R\_H)----------------

//----------------Y\_H = Ri\*MT\_H----------------

int mi = INT\_MAX;

#pragma omp parallel for

for (int i = start; i <= end; i++) {

if (Ri[i] < mi) {

mi = Ri[i];

}

Y[i] = 0;

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

Y[i] += Ri[j]\*MT[j][i];

}

}

//----------------m = min(m, mi)----------------

#pragma omp critical(min)

{

if (mi < m) {

m = mi;

}

}

#pragma omp barrier

int\* Yi;

int\* Zi;

int\* Si;

int\*\* MMi;

#pragma omp critical(Copy2)

{

Yi = Y;

mi = m;

Zi = Z;

Si = S;

MMi = MM;

}

//---A\_H = Yi\*ME\_H + mi\*Zi\*MX\_H + Si\*MMi\*MV\_H---

#pragma omp parallel for

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

for (int j = start; j <= end; j++) {

MR[i][j] = 0;

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

MR[i][j] += MMi[i][k] \* MV[k][j];

}

}

}

#pragma omp parallel for

for (int i = start; i <= end; i++) {

A[i] = 0;

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

A[i] += Yi[j] \* ME[j][i] + mi\*Zi[j]\*MX[j][i] + Si[j]\*MR[j][i];

}

}

#pragma omp barrier

#pragma omp critical(finish)

{

if (threadNumber == 2) {

outputVector(A);

}

cout << "Thread " << threadNumber << " finished." << endl;

}

}

}