Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Основи паралельного програмування

Лабораторна робота №5

**«Потоки в бібліотеці OpenMP»**

Виконала:

студентка групи ІВ-71

Молчанова В.С.

Перевірив:

Долголенко О.М.

Київ

2019 р.

**Завдання:**

F1: c = MAX(MA\*MB)\*(A\*B)

F2: MF = MAX(MG)\*(MH\*MK)

F3: T= SORT(O + P)\*TRANS(MR\*MS)

**Лістинг програми:**

**main.cpp**

#include <iostream>

#include <algorithm>

#include <functional>

#include <omp.h>

#include "math\_omp.h"

using namespace std;

// IV-71, Molchanova Varvara

// C = A\*(MA\*ME) + B + D

// ML = MIN(MF)\*MG + MAX(MH)\*(MK\*MF)

// T = (MO\*MP)\*S + MR\*SORT(S)

//To bulid:  g++ lab5.cpp math\_omp.cpp -o lab5

int\* f1Result;

int\*\* f2Result;

int\* f3Result;

void thread\_function1();

void thread\_function2();

void thread\_function3();

int\* f1(int a[], int b[], int d[], int\* ma[n], int\* me[n]);

int\*\* f2(int\* mf[n], int\* mg[n], int\* mh[n], int\* mk[n]);

int\* f3(int s[], int\* mo[n], int\* mp[n], int\* mr[n]);

int main(int argc, char \*argv[])

{

    cout << "Lab5 started" << endl << endl;

#pragma comment(linker, "/STACK:200000000")

    //omp\_set\_num\_threads(3);

#pragma omp parallel

    {

#pragma omp sections

        {

#pragma omp section

            {

                thread\_function1();

            }

#pragma omp section

            {

                thread\_function2();

            }

#pragma omp section

            {

                thread\_function3();

            }

        }

    }

    cout << endl << "F1 = " << endl;

    vector\_output(f1Result);

    cout << endl << "F2 = " << endl;

    matrix\_output(f2Result);

    cout << endl << "F3 = " << endl;

    vector\_output(f3Result);

    cout << endl << "Lab5 finished" << endl << endl;

    return 0;

}

void thread\_function1() {

    cout << "Thread 1 started" << endl;

    // C = A\*(MA\*ME) + B + D

    int a[n];

    vector\_fill(a, 1);

    int b[n];

    vector\_fill(b, 1);

    int d[n];

    vector\_fill(d, 1);

    int\* ma[n];

    matrix\_fill(ma, 1);

    int\* me[n];

    matrix\_fill(me, 1);

    f1Result = f1(a, b, d, ma, me);

    cout << "Thread 1 finished" << endl;

}

void thread\_function2() {

    cout << "Thread 2 started" << endl;

    // ML = MIN(MF)\*MG + MAX(MH)\*(MK\*MF)

    int\* mf[n];

    matrix\_fill(mf, 1);

    int\* mg[n];

    matrix\_fill(mg, 1);

    int\* mh[n];

    matrix\_fill(mh, 1);

    int\* mk[n];

    matrix\_fill(mk, 1);

    f2Result = f2(mf, mg, mh, mk);

    cout << "Thread 2 finished" << endl;

}

void thread\_function3() {

    cout << "Thread 3 started" << endl;

    // T = (MO\*MP)\*S + MR\*SORT(S)

    int s[n];

    vector\_fill(s, 1);

    int\* mo[n];

    matrix\_fill(mo, 1);

    int\* mp[n];

    matrix\_fill(mp, 1);

    int\* mr[n];

    matrix\_fill(mr, 1);

    f3Result = f3(s, mo, mp, mr);

    cout << "Thread 3 finished" << endl;

}

int\* f1(int a[], int b[], int d[], int\* ma[n], int\* me[n]) {

    // C = A\*(MA\*ME) + B + D

    return add(add(mul(a, mul(ma, me)), b), d);

}

int\*\* f2(int\* mf[n], int\* mg[n], int\* mh[n], int\* mk[n]) {

    // ML = MIN(MF)\*MG + MAX(MH)\*(MK\*MF)

    return add(mul(matrix\_min(mf), mg), mul(matrix\_max(mh), mul(mk, mf)));

}

int\* f3(int s[], int\* mo[n], int\* mp[n], int\* mr[n]) {

    // T = (MO\*MP)\*S + MR\*SORT(S)

    int\* v\_temp1;

    int\* v\_temp2;

    v\_temp1 = mul(s, (mo, mp));

    vector\_sort(s);

    v\_temp2 = mul(s, mr);

    return add(v\_temp1, v\_temp2);

}

**math\_omp.h**

#include <iostream>

#include <algorithm>

#include <functional>

#include <omp.h>

using namespace std;

const int n = 3;

void vector\_fill(int v[n], int value);

void matrix\_fill(int\* matrix[n], int value);

void vector\_output(int vector[n]);

void matrix\_output(int\* matrix[n]);

int\* add(int a[n], int b[n]);

int\*\* add(int\* ma[n], int\* mb[n]);

int scalar\_mul(int a[n], int b[n]);

int\*\* mul(int a, int\* ma[n]);

int\* mul(int a[n], int\* ma[n]);

int\*\* mul(int\* ma[n], int\* mb[n]);

void vector\_sort(int v[n]);

int matrix\_max(int\* ma[n]);

int matrix\_min(int\* ma[n]);

**math\_omp.cpp**

#include <iostream>

#include <algorithm>

#include <functional>

#include <omp.h>

#include "math\_omp.h"

using namespace std;

void vector\_fill(int v[n], int value) {

#pragma omp parallel for

    for (auto i = 0; i < n; i++)

    {

        v[i] = value;

    }

}

void matrix\_fill(int\* matrix[n], int value) {

#pragma omp parallel for

    for (auto i = 0; i < n; i++)

    {

        matrix[i] = new int[n];

#pragma omp parallel for

        for (auto j = 0; j < n; j++) {

            matrix[i][j] = value;

        }

    }

}

void vector\_output(int vector[n])

{

    if (n <= 10)

    {

        for (auto i = 0; i < n; i++)

        {

            cout << vector[i] << " ";

        }

        cout << endl;

    }

}

void matrix\_output(int\* matrix[n])

{

    if (n <= 10)

    {

        for (auto i = 0; i < n; i++)

        {

            for (auto j = 0; j < n; j++)

            {

                cout << matrix[i][j] << " ";

            }

            cout << endl;

        }

    }

}

void vector\_sort(int v[])

{

    sort(v, v + n);

}

int matrix\_min(int\* ma[n]) {

    auto n = ma[0][0];

# pragma omp parallel for

    for (auto i = 0; i < n; i++) {

# pragma omp parallel for

        for (auto j = 0; j < n; j++) {

            n = (n < ma[i][j]) ? n : ma[i][j];

        }

    }

    return n;

}

int matrix\_max(int\* ma[n]) {

    auto n = ma[0][0];

# pragma omp parallel for

    for (auto i = 0; i < n; i++) {

# pragma omp parallel for

        for (auto j = 0; j < n; j++) {

            n = (n > ma[i][j]) ? n : ma[i][j];

        }

    }

    return n;

}

int\* add(int a[], int b[]) {

    int\* result = new int[n];

#pragma omp parallel for

    for (auto i = 0; i < n; i++)

    {

        result[i] = a[i] + b[i];

    }

    return result;

}

int\*\* add(int\* ma[n], int\* mb[n]) {

    int\*\* result = new int\*[n];

#pragma omp parallel for

    for (auto i = 0; i < n; i++) {

        result[i] = new int[n];

#pragma omp parallel for

        for (auto j = 0; j < n; j++) {

            result[i][j] = ma[i][j] + mb[i][j];

        }

    }

    return result;

}

int scalar\_mul(int a[], int b[])

{

    auto c = 0;

#pragma omp parallel for

    for (auto i = 0; i < n; i++) {

        c += a[i] \* b[i];

    }

    return c;

}

int\*\* mul(int a, int\* ma[n])

{

    int\*\* result = new int\*[n];

#pragma omp parallel for

    for (auto i = 0; i < n; i++) {

        result[i] = new int[n];

#pragma omp parallel for

        for (auto j = 0; j < n; j++) {

            result[i][j] = ma[i][j] \* a;

        }

    }

    return result;

}

int\* mul(int a[], int\* ma[n]) {

    int\* result = new int[n];

#pragma omp parallel for

    for (auto j = 0; j < n; j++)

    {

        result[j] = 0;

#pragma omp parallel for

        for (auto k = 0; k < n; k++)

        {

            result[j] += a[k] \* ma[k][j];

        }

    }

    return result;

}

int\*\* mul(int\* ma[n], int\* mb[n])

{

    int\*\* result = new int\*[n];

#pragma omp parallel for

    for (auto i = 0; i < n; i++) {

        result[i] = new int[n];

#pragma omp parallel for

        for (auto j = 0; j < n; j++) {

            result[i][j] = 0;

#pragma omp parallel for

            for (auto k = 0; k < n; k++)

            {

                result[i][j] += ma[i][k] \* mb[k][j];

            }

        }

    }

    return result;

}

**Результат роботи:**

Lab5 started

Thread 1 started

Thread 1 finished

Thread 2 started

Thread 2 finished

Thread 3 started

Thread 3 finished

F1 =

11 11 11

F2 =

4 4 4

4 4 4

4 4 4

F3 =

6 6 6

Lab5 finished