НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №6**

з дисципліни **«**Паралельні та розподілені обчислення**»**

**ПОТОКИ У БІБЛІОТЕЦІ OpenMP**

Виконав:

студент 3 курсу гр. ІО-31

Долинний Олександр

№ ЗК 3110

Перевірив:

Корочкін О. В.

Київ 2015 р.

***Завдання:***

1.1 3.21 3.11

F1: A = sort(B) \* (MB \* MC)

F2: W = sort(R \* MT) \* (MX \* MS)

F3: R = sort(S + T) \* trans(MS \* MR)

***Лістинг програми:***

#include <iostream>

#include <algorithm>

#include <functional>

#include <omp.h>

using namespace std;

/\*\*

\* Parallel and distributed computing

Lab 6: Threads in OpenMP library

Dolinniy Alexandr

IO-31

08.11.2015

Func1: A = sort(B)\*(MB\*MC)

Func2: W = sort(R\*MT)\*(MX\*MS)

Func3: R = sort(S+T)\*trans(MS\*MR)

\*/

const int n = 3000;

void threadFunction1();

void threadFunction2();

void threadFunction3();

void f1(int b[], int mb[][n], int mc[][n], int a[]);

void f2(int r[], int mt[][n], int mx[][n], int ms[][n], int w[]);

void f3(int s[], int t[], int ms[][n], int mr[][n], int r[]);

void vectorFill(int v[n], int value);

void matrixFill(int matrix[n][n], int value);

void vectorOutput(int vector[n]);

void matrixOutput(int matrix[n][n]);

void add(int a[], int b[], int c[]);

int scalarMul(int a[], int b[]);

void mul(int a[], int ma[][n], int c[]);

void mul(int ma[][n], int mb[][n], int c[][n]);

void vectorSort(int v[]);

void matrixSort(int matrix[][n]);

void trans(int ma[][n]);

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

{

cout << "Lab6 started" << endl;

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

omp\_set\_num\_threads(3);

#pragma omp parallel

{

#pragma omp sections

{

#pragma omp section

{

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

cout << i << endl;

threadFunction1();

}

#pragma omp section

{

threadFunction2();

}

#pragma omp section

{

threadFunction3();

}

}

}

cout << "Lab 6 finished" << endl;

char ch;

cin >> ch;

return 0;

}

void threadFunction1(){

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

int b[n];

vectorFill(b, 1);

int mb[n][n];

matrixFill(mb, 1);

int mc[n][n];

matrixFill(mc, 1);

int a[n];

f1(b, mb, mc, a);

cout << "F1:" << endl;

vectorOutput(a);

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

}

void threadFunction2(){

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

int w[n];

int r[n];

vectorFill(r, 1);

int mt[n][n];

matrixFill(mt, 1);

int mx[n][n];

matrixFill(mx, 1);

int ms[n][n];

matrixFill(ms, 1);

f2(r, mt, mx, ms, w);

cout << "F2:" << endl;

vectorOutput(w);

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

}

void threadFunction3(){

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

int r[n];

int s[n];

vectorFill(s, 1);

int t[n];

vectorFill(t, 1);

int ms[n][n];

matrixFill(ms, 1);

int mr[n][n];

matrixFill(mr, 1);

f3(s, t, ms, mr, r);

cout << "F3:" << endl;

vectorOutput(r);

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

}

void f1(int b[], int mb[][n], int mc[][n], int a[]){

int m1[n][n];

matrixFill(m1, 0);

mul(mb, mc, m1);

vectorSort(b);

mul(b, m1, a);

}

void f2(int r[], int mt[][n], int mx[][n], int ms[][n], int w[]){

int m2[n][n];

matrixFill(m2, 0);

mul(mx, ms, m2);

int v2[n];

mul(r, mt, v2);

vectorSort(v2);

mul(v2, m2, w);

}

void f3(int s[], int t[], int ms[][n], int mr[][n], int r[]){

int v3[n];

add(s, t, v3);

vectorSort(v3);

int m3[n][n];

mul(ms, mr, m3);

trans(m3);

mul(v3, m3, r);

}

void vectorFill(int v[n], int value) {

#pragma omp parallel for

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

{

v[i] = value;

}

}

void matrixFill(int matrix[n][n], int value) {

#pragma omp parallel for

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

{

#pragma omp parallel for

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

matrix[i][j] = value;

}

}

}

void vectorOutput(int vector[n])

{

if (n <= 10)

{

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

{

cout << vector[i] << " ";

}

cout << endl;

}

}

void matrixOutput(int matrix[n][n])

{

if (n <= 10)

{

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

{

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

{

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

}

cout << endl;

}

}

}

void add(int a[], int b[], int c[]) {

c[n];

#pragma omp parallel for

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

{

c[i] = a[i] + b[i];

}

}

int scalarMul(int a[], int b[])

{

int c = 0;

#pragma omp parallel for

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

c += a[i] \* b[i];

}

return c;

}

void mul(int a[], int ma[][n], int c[]) {

c[n];

#pragma omp parallel for

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

{

c[j] = 0;

#pragma omp parallel for

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

{

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

}

}

}

void mul(int ma[][n], int mb[][n], int c[][n])

{

c[n][n];

#pragma omp parallel for

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

#pragma omp parallel for

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

c[i][j] = 0;

#pragma omp parallel for

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

{

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

}

}

}

}

void vectorSort(int v[])

{

sort(v, v + n);

}

void matrixSort(int matrix[][n])

{

#pragma omp parallel for

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

sort(matrix[i], matrix[i] + n, greater<int>());

}

}

void trans(int ma[][n])

{

#pragma omp parallel for

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

#pragma omp parallel for

for (int j = i; j < n; j++) {

int buf = ma[i][j];

ma[i][j] = ma[j][i];

ma[j][i] = buf;

}

}

}