Лабораторная работа №6

В данной лабораторной работе использовались lock и barrier. Lock – самая медленная из всех синхронизаций ввиду того, что замки блокируют ядро, ввиду чего другие реализации например в виде критических секций, которые используют некий легкий замок быстрее, так как не связаны накладными расходами. Barrier – в данном случае является просто инструментом, который позволяет распаралеливать данные при ограниченом количестве памяти, но при этом распределить их между несколькими переменнами а потом соединить вместе, что ускоряет программу с синхронизацией, если вычисления равномерны.

Результаты с массивом 15000000. Распоралеливание на 3 потока, локи с 3 потоками, рарьер с 3 потоками:

0.0410001s with reduction 7.49974e+11

38.307s with lock 7.49974e+11

0.391s with barrier 7.49974e+11

Код:

#include <iostream>

#include "header.h"

#include <math.h>

#include <time.h>

using namespace std;

const int N = 15000000;

void **sixth** (){

//srand(time(NULL));

int i;

double composition2 = 0;

double timed2 = 0;

double finish, start;

int\* A = new int[N];

double\* B = new double[N];

double\* C = new double[N];

random\_device rd;

mt19937 mersenne(rd());

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

{

A[i] = (mersenne() % 100000)+1;

B[i] = 1+(mersenne() % 100000);

C[i] = 1+(mersenne() % 100000);

}

start = omp\_get\_wtime();

#pragma omp parallel private(timed2) num\_threads(3)

{

#pragma omp for reduction(+:composition2)

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

{

if ((A[i] % 2) == 0)

{

timed2 = B[i] / C[i];

}

else

{

timed2 = B[i] + A[i];

}

composition2 += timed2;

}

}

finish = omp\_get\_wtime();

cout << finish - start << "s with reduction ";

cout << composition2 << endl;

composition2 = 0;

start = omp\_get\_wtime();

omp\_lock\_t lock;

omp\_init\_lock(&lock);

#pragma omp parallel private(timed2) num\_threads(3)

{

#pragma omp for

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

{

if ((A[i] % 2) == 0)

{

timed2 = B[i] / C[i];

}

else

{

timed2 = B[i] + A[i];

}

omp\_set\_lock (&lock);

composition2 += timed2;

omp\_unset\_lock (&lock);

}

}

omp\_destroy\_lock (&lock);

finish = omp\_get\_wtime();

cout << finish - start << "s with lock ";

cout << composition2 << endl;

composition2 = 0;

double sum[3] = {0,0,0};

start = omp\_get\_wtime();

#pragma omp parallel private(timed2) num\_threads(3)

{

#pragma omp for nowait

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

{

if ((A[i] % 2) == 0)

{

timed2 = B[i] / C[i];

}

else

{

timed2 = B[i] + A[i];

}

sum[omp\_get\_thread\_num()]+=timed2;

}

#pragma omp barrier

composition2 = sum[0] + sum[1] + sum[2];

}

finish = omp\_get\_wtime();

cout << finish - start << "s with barrier ";

cout << composition2 << endl;

}