**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

**«Сибирский государственный университет науки и технологий   
имени академика М.Ф. Решетнева»**

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

Параллельные вычисления

|  |
| --- |
| Знакомство с OpenMP |

Руководитель А.Г. Зотин

подпись, дата инициалы, фамилия

Обучающийся БПИ20-02, 201219047 Р.А. Сухачев

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2022 г.

# ЦЕЛЬ РАБОТЫ

Получение практических навыков разработки простых параллельных программ с использованием OpenMP. Ознакомление с основными функциями и директивами OpenMP.

# порядок выполнения работы

1. Ознакомиться с теоретическими сведениями.

2. Выполнить задания 1–3.

3. Подготовить отчет по лабораторной работе.

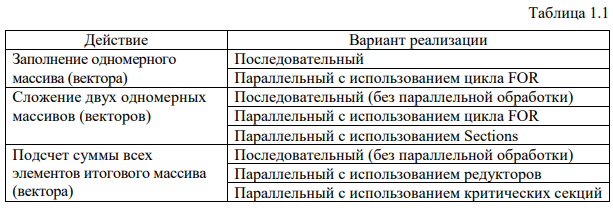
4.  Защитить лабораторную работу перед преподавателем.

# постановка задачи

Вариант 20

Задание 1.1. Реализуйте консольное приложение с поддержкой OpenMP, содержащее функции (с разным стилем вывода на экран – через cout и printf), в которых создается Х потоков, каждый из которых выводит сообщение приветствия от потока, например «Hello World! От потока #». До запуска функций программа должна вывести сообщение, показывающее максимально доступное в системе количество потоков. Пользователь может задать значение числа Х, которое может превышать максимально доступное в системе количество потоков.

Задание 1.2. Разработайте консольное приложение, реализующее сложение векторов и подсчет суммы всех элементов итогового вектора. Для каждой реализации функций выполните расчет времени. Для всех действий необходимо реализовать функции в нескольких вариантах, указанных в табл. 1.1.



При реализации заполнения массива (вектора) для каждого массива студент должен использовать свои функции вычисления значения элемен- 2 та в некоторой зависимости от порядкового номера элемента. Так, возможно использовать в основе тригонометрические функции (sin, cos), например: sin(i-50) + cos(i/2). Также допустимо использовать степенных функции с дробной степенью, умноженные на результат тригонометрических функций, например: pow(i, 2.0 / 3.0) \* sin(i / 2.0). Или вычисления тригонометрической функции от некоего логарифмического значения, например: cos(log(i / 2.7)) или sin(log10(i \* 0.3)).

Задание 1.3. Проведите экспериментальное исследование по обработке данных с разными реализациями алгоритмов на сборке Release. При исследовании используйте одномерные массивы типа double размерностью от 100 000 до 250 0001 . В ходе исследования необходимо выяснить время выполнения отдельных функций и полное время решения задачи (три шага: заполнение массивов, сложение массивов и подсчет суммы всех элементов итогового массива). Рассчитайте показатели эффективности для решения задачи. При заполнении времени в таблицах используйте среднеарифметическое значение для доверительного интервала, полученного для не менее чем 100 запусков2 выполнения реализации функции (набора функций для решения задачи). Для отдельных функций заполните табл. 1.2.



# ХОД РАБОТЫ

Задание 1.

Код программы:

#include <iostream>

#include <omp.h>

void coutHello()

{

std::cout << "Hello World!" << std::endl;

#pragma omp parallel

{

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

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

std::cout << "I'm thread " << i << " of " << n << "!\n";

}

}

void printfHello()

{

printf("Hello World!\n");

#pragma omp parallel

{

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

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

printf("I'm thread %d of %d!\n", i, n);

}

}

int main()

{

std::cout << "Max threads of the system is " << omp\_get\_max\_threads() << std::endl;

int count = 0;

std::cout << "Enter count of threads: ";

std::cin >> count;

omp\_set\_num\_threads(count);

coutHello();

std::cout << "=========================" << std::endl;

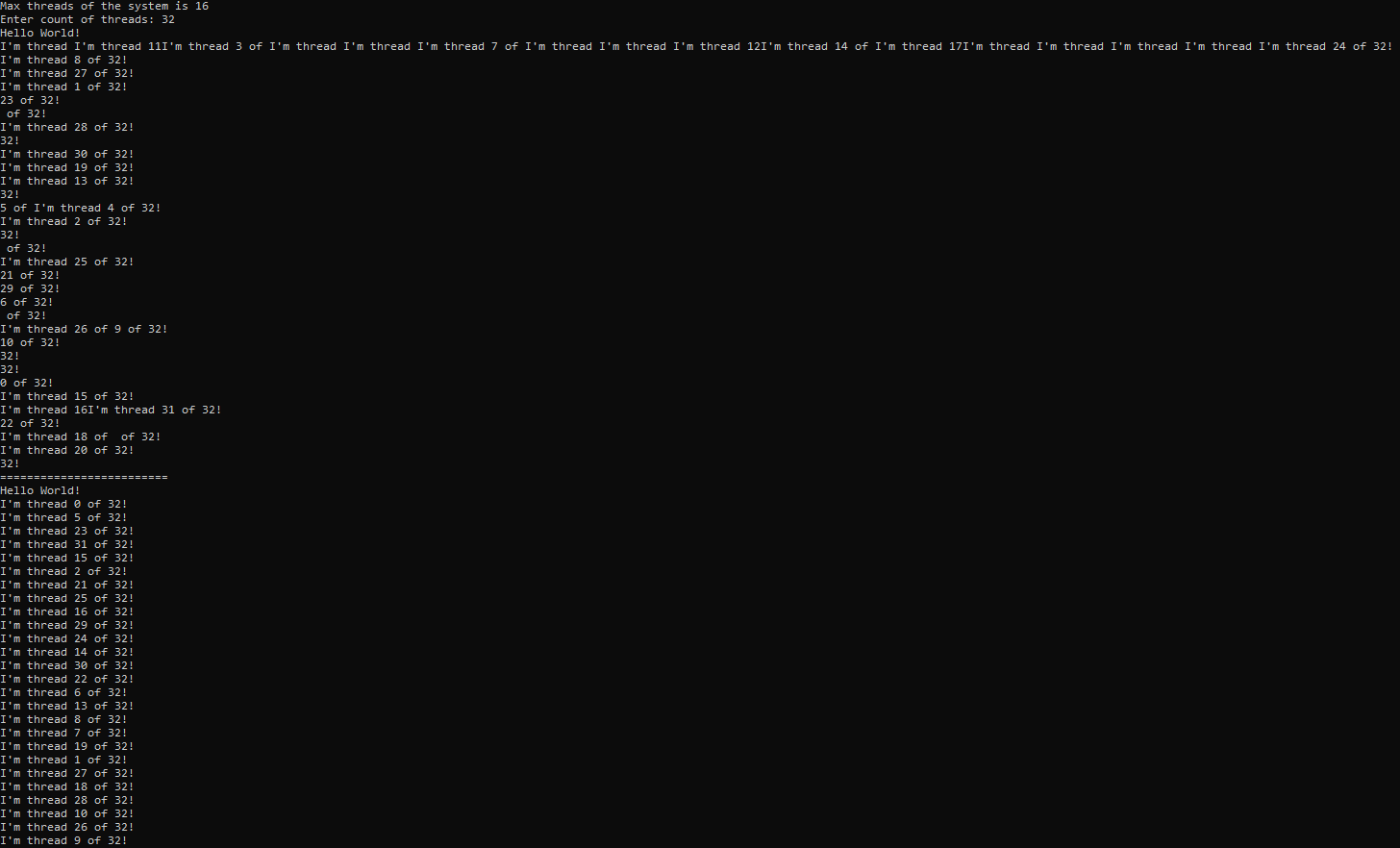
printfHello();

system("pause");

return 0;

}

Демонстрация работы программы представлена на рисунке 1.

Рисунок 1 – работы программы.

Задание 2.

Функция для заполнения массивов - sin(i)+cos(i/2).

Код программы:

#include <iostream>

#include <fstream>

#include <omp.h>

#include <vector>

#include <Math.h>

using namespace std;

typedef double(\*TestFunctTempl)(vector<double>&, vector<double>&, vector<double>&, int&);

double FillArrayConsistent(vector<double>& a, int& size)

{

double t\_start = omp\_get\_wtime();

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

{

a[i] = (sin(i - 50) + cos(i / 2));

}

double t\_end = omp\_get\_wtime();

return t\_end - t\_start;

}

double FillArrayParallel(vector<double>& a, int& size)

{

double t\_start = omp\_get\_wtime();

#pragma omp parallel for

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

{

a[i] = (sin(i - 50) + cos(i / 2));

}

double t\_end = omp\_get\_wtime();

return t\_end - t\_start;

}

double SumArrayConsistent(vector<double> a, vector<double> b, vector<double>& c, int size)

{

double t\_start = omp\_get\_wtime();

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

{

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

}

double t\_end = omp\_get\_wtime();

return t\_end - t\_start;

}

double SumArrayParallel(vector<double> a, vector<double> b, vector<double>& c, int size)

{

double t\_start = omp\_get\_wtime();

#pragma omp parallel for

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

{

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

}

double t\_end = omp\_get\_wtime();

return t\_end - t\_start;

}

double SumArraySections(vector<double> a, vector<double> b, vector<double>& c, int size)

{

double t\_start = omp\_get\_wtime();

c.clear();

int n\_t, i = 0;

#pragma omp parallel

{

n\_t = omp\_get\_max\_threads();

}

int st = 0, st1 = size / n\_t, st2 = size \* 2 / n\_t, st3 = size \* 3 / n\_t, se = size;

#pragma omp parallel sections private(i)

{

#pragma omp section

{

for (i = st; i < st1; i++)

{

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

}

}

#pragma omp section

{

for (i = st1; i < st2; i++)

{

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

}

}

#pragma omp section

{

if (n\_t > 2)

{

for (i = st2; i < st3; i++)

{

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

}

}

}

#pragma omp section

{

if (n\_t > 3)

{

for (i = st3; i < se; i++)

{

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

}

}

}

}

double t\_end = omp\_get\_wtime();

return t\_end - t\_start;

}

double SumResultConsistent(vector<double>& c, double sum, int size)

{

double t\_start = omp\_get\_wtime();

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

{

sum += c[i];

}

double t\_end = omp\_get\_wtime();

return t\_end - t\_start;

}

double SumResultReduction(vector<double>& c, double sum, int size)

{

double t\_start = omp\_get\_wtime();

#pragma omp parallel for reduction(+:sum)

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

{

sum += c[i];

}

double t\_end = omp\_get\_wtime();

return t\_end - t\_start;

}

double sumResultCritical(vector<double>& c, double sum, int size)

{

double t\_start = omp\_get\_wtime();

double temp = 0;

#pragma omp parallel for

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

#pragma omp critical

temp += c[i];

sum = temp;

double t\_end = omp\_get\_wtime();

return t\_end - t\_start;

}

double TestFillConsistent(vector<double>& a, vector<double>& empty1, vector<double>& empty2, int& size)

{

return FillArrayConsistent(a, size);

}

double TestFillParallel(vector<double>& a, vector<double>& empty1, vector<double>& empty2, int& size)

{

return FillArrayParallel(a, size);

}

double TestSumConsistent(vector<double>& a, vector<double>& b, vector<double>& c, int& size)

{

return SumArrayConsistent(a, b, c, size);

}

double TestSumParallel(vector<double>& a, vector<double>& b, vector<double>& c, int& size)

{

return SumArrayParallel(a, b, c, size);

}

double TestSumSections(vector<double>& a, vector<double>& b, vector<double>& c, int& size)

{

return SumArraySections(a, b, c, size);

}

double TestSumResultConsistent(vector<double>& empty1, vector<double>& empty2, vector<double>& c, int& size)

{

double sum = 0;

return SumResultConsistent(c, sum, size);

}

double TestSumResultReduction(vector<double>& empty1, vector<double>& empty2, vector<double>& c, int& size)

{

double sum = 0;

return SumResultReduction(c, sum, size);

}

double TestSumResultCritical(vector<double>& empty1, vector<double>& empty2, vector<double>& c, int& size)

{

double sum = 0;

return sumResultCritical(c, sum, size);

}

double AvgTrustedInterval(double& avg, vector<double>& times, int& cnt)

{

double sd = 0, newAVg = 0;

int newCnt = 0;

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

{

sd += (times[i] - avg) \* (times[i] - avg);

}

sd /= (cnt - 1.0);

sd = sqrt(sd);

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

{

if (avg - sd <= times[i] && times[i] <= avg + sd)

{

newAVg += times[i];

newCnt++;

}

}

if (newCnt == 0) newCnt = 1;

return newAVg / newCnt;

}

double TestIter(void\* Funct, vector<double>& a, vector<double>& b, vector<double>& c, int size, int iterations)

{

double curtime = 0, avgTime = 0, avgTimeT = 0, correctAVG = 0;;

vector<double> Times(iterations);

cout << endl;

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

{

curtime = ((\*(TestFunctTempl)Funct)(a, b, c, size)) \* 1000;

Times[i] = curtime;

avgTime += curtime;

cout << "+";

}

cout << endl;

avgTime /= iterations;

cout << "AvgTime:" << avgTime << endl;

avgTimeT = AvgTrustedInterval(avgTime, Times, iterations);

cout << "AvgTimeTrusted:" << avgTimeT << endl;

return avgTimeT;

}

void test\_functions(void\*\* Functions, vector<string> fNames)

{

vector<double> a, b, c;

int iters = 100;

int nd = 0;

double times[4][8][3];

for (int size = 100000; size <= 250000; size += 50000)

{

a.resize(size);

b.resize(size);

c.resize(size);

FillArrayConsistent(b, size);

for (int threads = 1; threads <= 4; threads++)

{

omp\_set\_num\_threads(threads);

for (int alg = 0; alg <= 7; alg++)

{

if (threads == 1)

{

if (alg == 0 || alg == 2 || alg == 5) {

times[nd][alg][0] = TestIter(Functions[alg], a, b, c, size, iters);

times[nd][alg][1] = times[nd][alg][0];

times[nd][alg][2] = times[nd][alg][0];

}

}

else

{

if (alg != 0 && alg != 2 && alg != 5)

{

times[nd][alg][threads - 2] = TestIter(Functions[alg], a, b, c, size, iters);

}

}

}

}

nd++;

}

ofstream fout("output.txt");

for (int threads = 1; threads <= 4; threads++)

{

cout << "Thread " << threads << " --------------" << endl;

for (int ND = 0; ND < 4; ND++)

{

switch (ND)

{

case 0:

cout << "\n----------100000 elements of array----------" << endl;

break;

case 1:

cout << "\n----------150000 elements of array----------" << endl;

break;

case 2:

cout << "\n----------200000 elements of array----------" << endl;

break;

case 3:

cout << "\n----------250000 elements of array----------" << endl;

break;

default:

break;

}

for (int alg = 0; alg < 8; alg++)

{

if (threads == 1)

{

if (alg == 0 || alg == 2 || alg == 5) {

cout << fNames[alg] << "\t" << times[ND][alg][0] << " ms." << endl;

fout << times[ND][alg][0] << endl;

}

}

else

{

if (alg != 0 && alg != 2 && alg != 5)

{

cout << fNames[alg] << "\t" << times[ND][alg][threads - 2] << " ms." << endl;

fout << times[ND][alg][threads - 2] << endl;

}

}

}

}

}

fout.close();

}

int main()

{

void\*\* Functions = new void\* [8] { FillArrayConsistent, FillArrayParallel, TestSumConsistent, TestSumParallel, TestSumSections, TestSumResultConsistent, TestSumResultReduction, TestSumResultCritical };

vector<string> function\_names = { "Consistent filling","Parallel filling using loop FOR",

"Consistent sum two vectors",

"Parallel sum two vectors using loop FOR", "Parallel sum two vectors using Sections",

"Consistent sum the last vector", "Parallel sum last vector using Reduction",

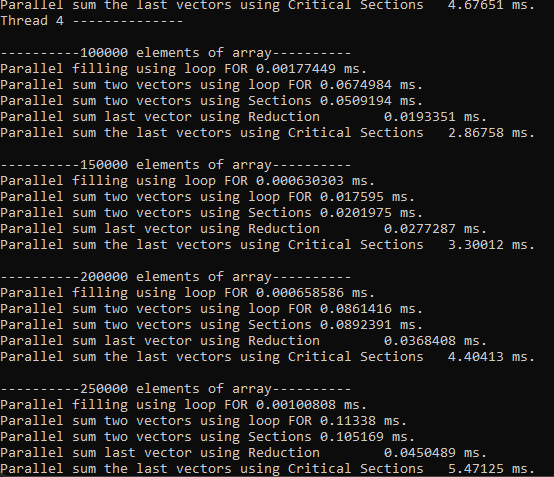
"Parallel sum the last vectors using Critical Sections" };

test\_functions(Functions, function\_names);

return 0;

}

Демонстрация работы программы представлена на рисунке 2.

Рисунок 2 – демонстрация работы программы 2

Задание 3.

Конфигурация компьютера и параметры операционной системы на котором производится выполнение данного задания:

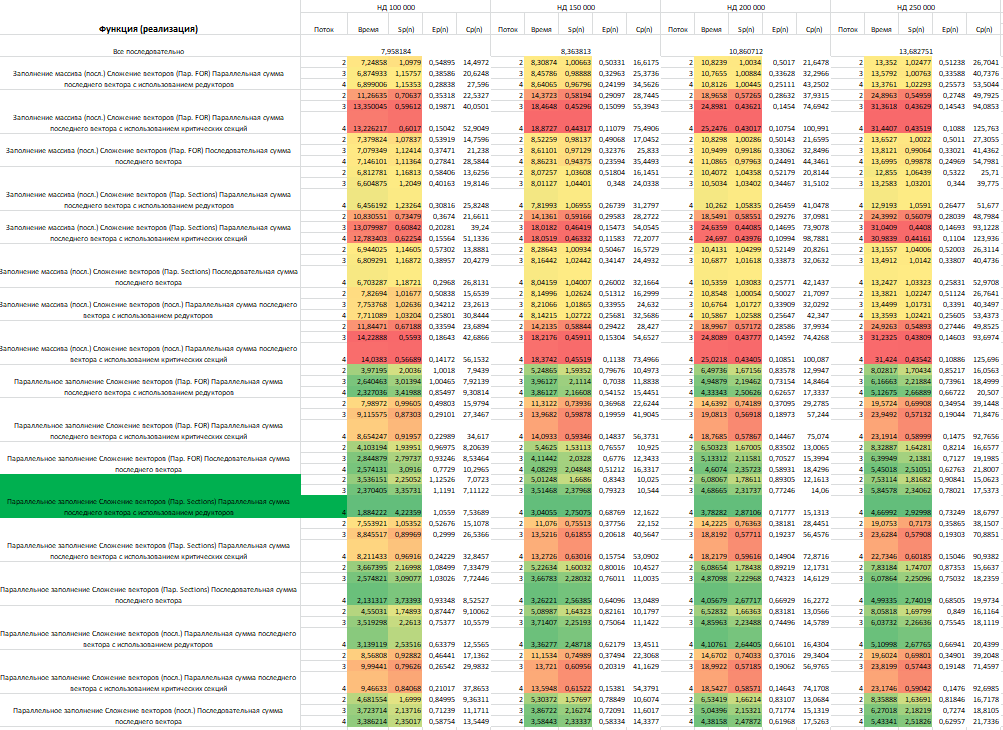
Процессор: Intel Core i3 1115G4 3 ГГц (4.1 ГГц) – 2 ядра 4 потока

Операционная система: Windows 11

Таблица 1 – результаты замеров времени выполнения отдельных функций



Таблица 2 – результаты замеров времени работы всей программы с разными наборами функций



Вывод: в ходе выполнения работы были получены замеры времени выполнения работы функций в последовательном и многопоточном режимах. Работа функций в параллельном режиме с использованием цикла For, использованием секций, а также параллельной редукции проходит быстрее, в то время как работа функции в параллельном режиме с использованием критических секций проходит, напротив, медленнее.

# ВЫВОДЫ

Были получены практические навыки разработки простых параллельных программ с использованием OpenMP. Было проведено ознакомление с основными функциями и директивами OpenMP.