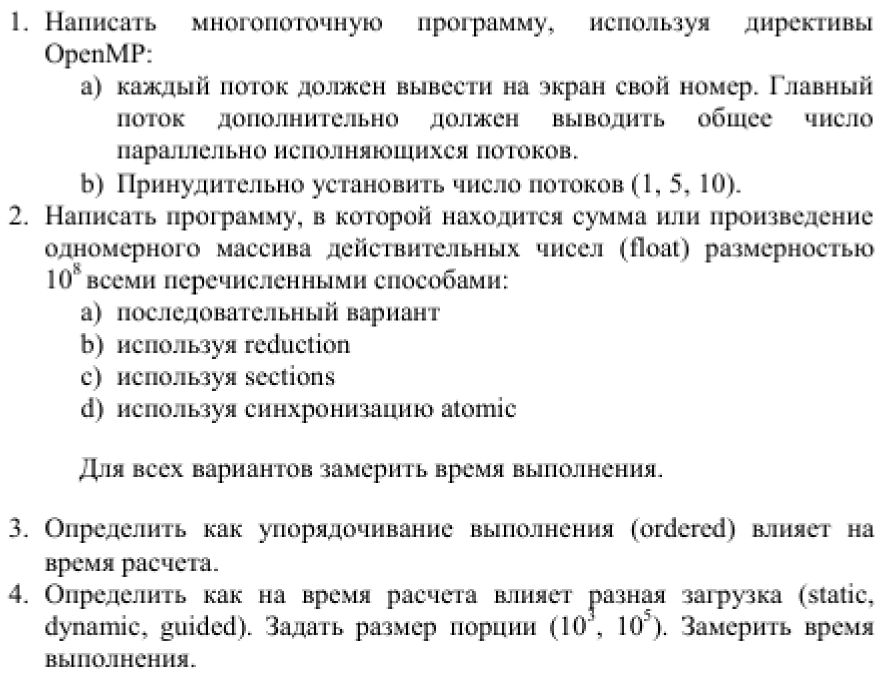
Лабораторная работа № 1 Знакомство с OpenMP	ФИО	Титов А.К.
	Группа	ИВТ 460
	Предмет	Введение в параллельное программирование
	Дата отчета	
	Оценка	
	Подпись преподавателя	

Постановка задачи



1. Написать многопоточную программу, используя директивы OpenMP:
    a) каждый поток должен вывести на экран свой номер. Главный поток дополнительно должен выводить общее число параллельно исполняющихся потоков.
    b) Принудительно установить число потоков (1, 5, 10).
2. Написать программу, в которой находится сумма или произведение одномерного массива действительных чисел (float) размерностью 10^8 всеми перечисленными способами:
    a) последовательный вариант
    b) используя reduction
    c) используя sections
    d) используя синхронизацию atomic

    Для всех вариантов замерить время выполнения.

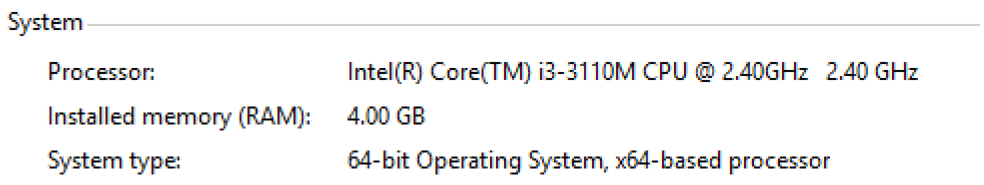
3. Определить как упорядочивание выполнения (ordered) влияет на время расчета.
4. Определить как на время расчета влияет разная загрузка (static, dynamic, guided). Задать размер порции (10^3, 10^5). Замерить время выполнения.

Индивидуальная задача (реализовать и протестировать на кластере и на собственном ПК)

Задача построения массивов значений нескольких функций на определенном отрезке. Подзадача – обработка одной функции.

Результаты

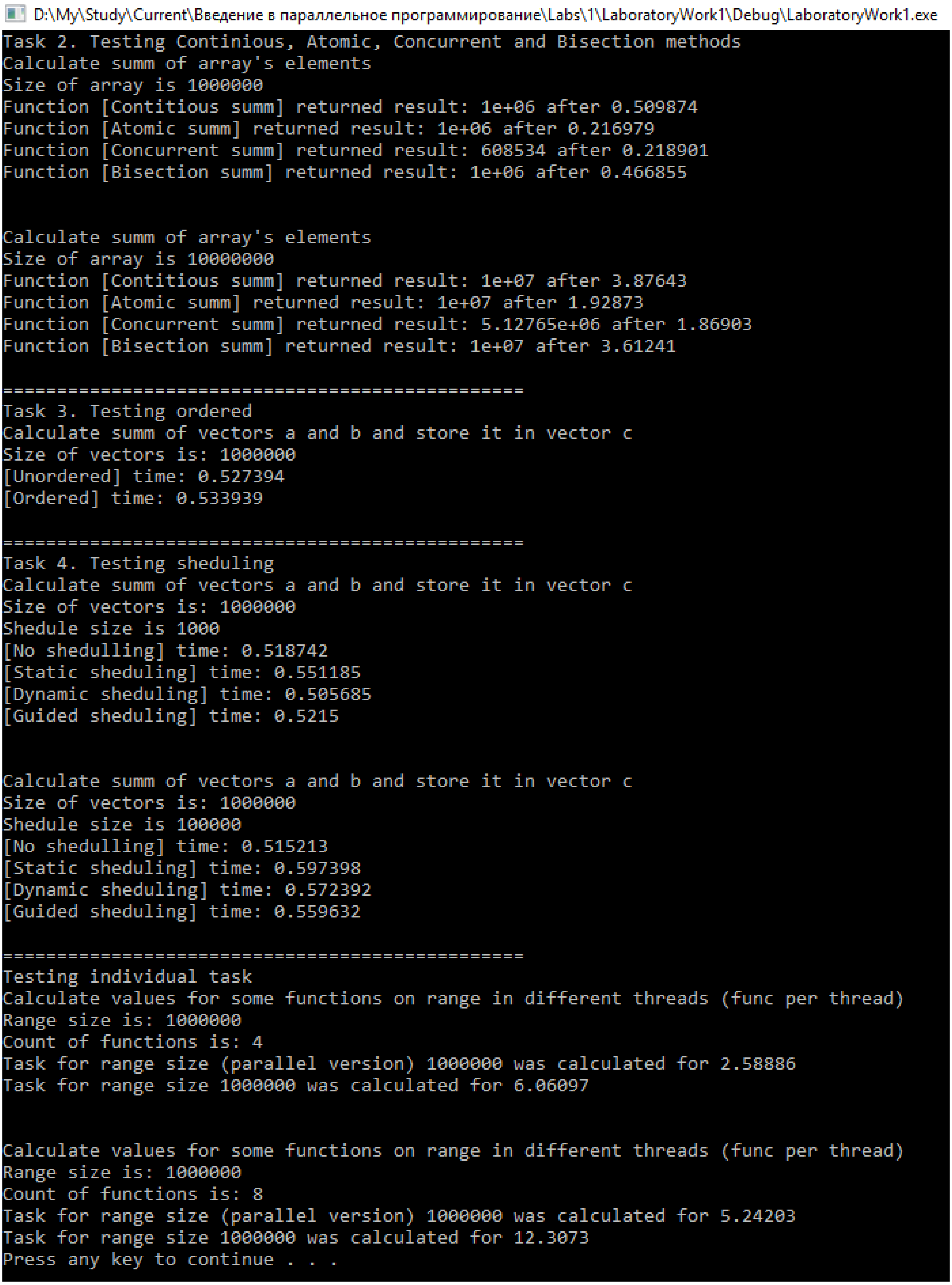
Конфигурация ПК



System

Processor:	Intel(R) Core(TM) i3-3110M CPU @ 2.40GHz 2.40 GHz
Installed memory (RAM):	4.00 GB
System type:	64-bit Operating System, x64-based processor

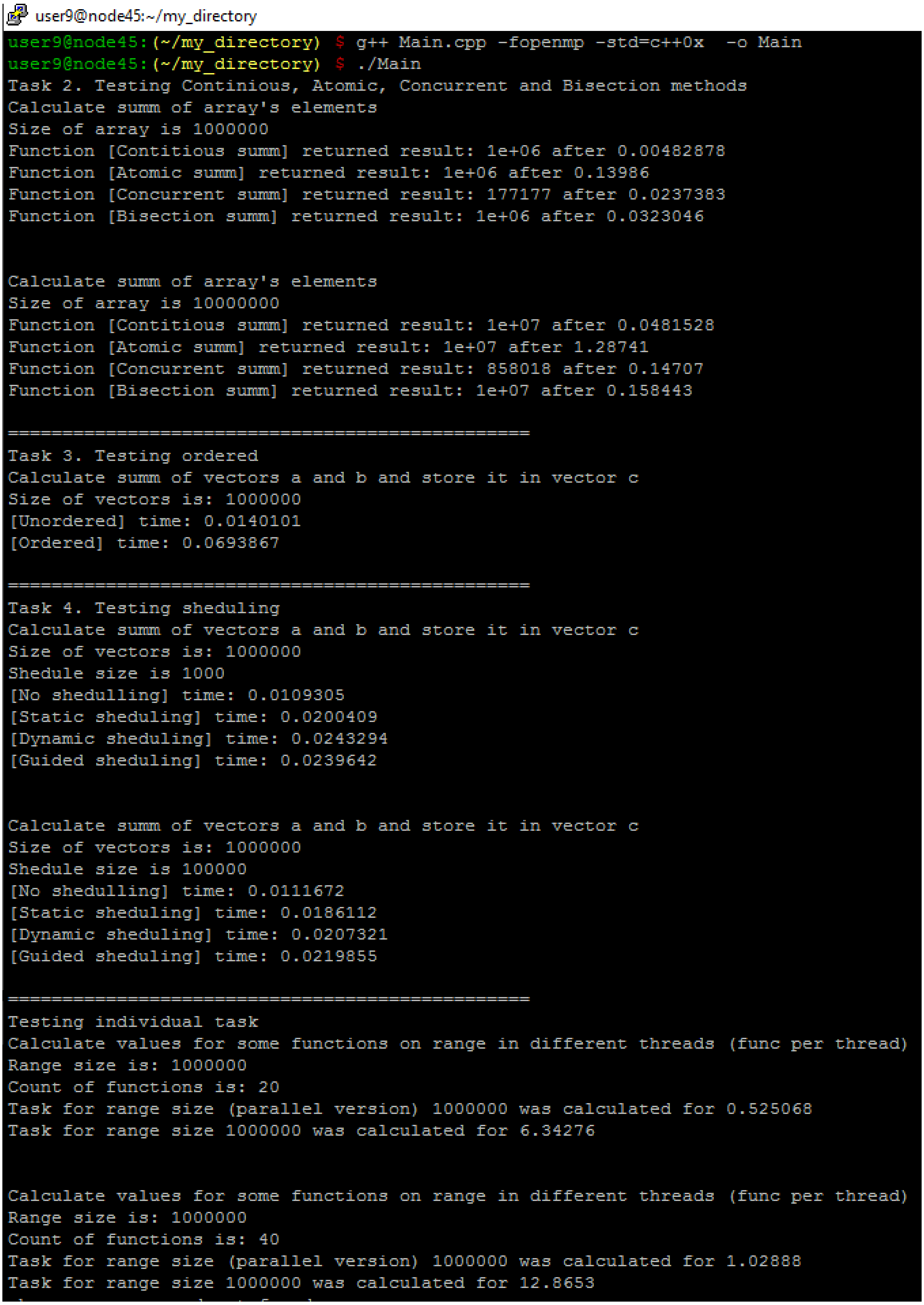
Результаты запуска на ПК



D:\My\Study\Current\Введение в параллельное программирование\Labs\1\LaboratoryWork1\Debug\LaboratoryWork1.exe

```
Task 2. Testing Continious, Atomic, Concurrent and Bisection methods
Calculate summ of array's elements
Size of array is 1000000
Function [Contitious summ] returned result: 1e+06 after 0.509874
Function [Atomic summ] returned result: 1e+06 after 0.216979
Function [Concurrent summ] returned result: 608534 after 0.218901
Function [Bisection summ] returned result: 1e+06 after 0.466855


Calculate summ of array's elements
Size of array is 10000000
Function [Contitious summ] returned result: 1e+07 after 3.87643
Function [Atomic summ] returned result: 1e+07 after 1.92873
Function [Concurrent summ] returned result: 5.12765e+06 after 1.86903
Function [Bisection summ] returned result: 1e+07 after 3.61241

================================================
Task 3. Testing ordered
Calculate summ of vectors a and b and store it in vector c
Size of vectors is: 1000000
[Unordered] time: 0.527394
[Ordered] time: 0.533939

================================================
Task 4. Testing sheduling
Calculate summ of vectors a and b and store it in vector c
Size of vectors is: 1000000
Shedule size is 1000
[No shedulling] time: 0.518742
[Static sheduling] time: 0.551185
[Dynamic sheduling] time: 0.505685
[Guided sheduling] time: 0.5215


Calculate summ of vectors a and b and store it in vector c
Size of vectors is: 1000000
Shedule size is 100000
[No shedulling] time: 0.515213
[Static sheduling] time: 0.597398
[Dynamic sheduling] time: 0.572392
[Guided sheduling] time: 0.559632

================================================
Testing individual task
Calculate values for some functions on range in different threads (func per thread)
Range size is: 1000000
Count of functions is: 4
Task for range size (parallel version) 1000000 was calculated for 2.58886
Task for range size 1000000 was calculated for 6.06097


Calculate values for some functions on range in different threads (func per thread)
Range size is: 1000000
Count of functions is: 8
Task for range size (parallel version) 1000000 was calculated for 5.24203
Task for range size 1000000 was calculated for 12.3073
Press any key to continue . . .
```

Результаты запуска на кластере



```
user9@node45:~/my_directory
user9@node45:(~/my_directory) $ g++ Main.cpp -fopenmp -std=c++0x  -o Main
user9@node45:(~/my_directory) $ ./Main
Task 2. Testing Continious, Atomic, Concurrent and Bisection methods
Calculate summ of array's elements
Size of array is 1000000
Function [Contitious summ] returned result: 1e+06 after 0.00482878
Function [Atomic summ] returned result: 1e+06 after 0.13986
Function [Concurrent summ] returned result: 177177 after 0.0237383
Function [Bisection summ] returned result: 1e+06 after 0.0323046


Calculate summ of array's elements
Size of array is 10000000
Function [Contitious summ] returned result: 1e+07 after 0.0481528
Function [Atomic summ] returned result: 1e+07 after 1.28741
Function [Concurrent summ] returned result: 858018 after 0.14707
Function [Bisection summ] returned result: 1e+07 after 0.158443

================================================
Task 3. Testing ordered
Calculate summ of vectors a and b and store it in vector c
Size of vectors is: 1000000
[Unordered] time: 0.0140101
[Ordered] time: 0.0693867

================================================
Task 4. Testing sheduling
Calculate summ of vectors a and b and store it in vector c
Size of vectors is: 1000000
Shedule size is 1000
[No shedulling] time: 0.0109305
[Static sheduling] time: 0.0200409
[Dynamic sheduling] time: 0.0243294
[Guided sheduling] time: 0.0239642


Calculate summ of vectors a and b and store it in vector c
Size of vectors is: 1000000
Shedule size is 100000
[No shedulling] time: 0.0111672
[Static sheduling] time: 0.0186112
[Dynamic sheduling] time: 0.0207321
[Guided sheduling] time: 0.0219855

================================================
Testing individual task
Calculate values for some functions on range in different threads (func per thread)
Range size is: 1000000
Count of functions is: 20
Task for range size (parallel version) 1000000 was calculated for 0.525068
Task for range size 1000000 was calculated for 6.34276


Calculate values for some functions on range in different threads (func per thread)
Range size is: 1000000
Count of functions is: 40
Task for range size (parallel version) 1000000 was calculated for 1.02888
Task for range size 1000000 was calculated for 12.8653
```

Выводы

Приложение А. Код программы

```
// Попробуй 1 - использовать функции замера времени из stdlib (clock) (Результат тот же)
// Попробуй 2 - использовать другую версию OpenMP, ибо эта может не поддерживать вектора из stdlib
(все также)
// TODO Попробуй 3 - в индивидуальном задании использовать разные функции (мб компилятор параллелит
их хорошо и без omp, т.к. они одинаковые)
#include <vector>
#include <iostream>
#include <omp.h>
#include <cmath>
#include <string>
#include <cstdlib>

// Task 1
typedef double(*calculate_summ) (const std::vector<double> & a);
void task1() {
    omp_set_num_threads(10);

    #pragma omp parallel
    {
        int my_id = omp_get_thread_num();
        std::cout << my_id;
    }
}

// Task 2
double continious_summ(const std::vector<double> & a) {
    double summ = 0.0;
    int n = a.size();
    for (int i = 0; i < n; i++) {
        summ += a[i];
    }

    return summ;
}

double concurrent_summ(const std::vector<double> & a) {
    double summ = 0.0;
    int n = a.size();
    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        summ += a[i];
    }

    return summ;
}

double bisection_summ(const std::vector<double> & a) {
    double summ = 0.0;
    int n = a.size();

    int blocks_count = omp_get_num_procs();
    int block_size = n / blocks_count;
    std::vector<double> summs(blocks_count);

    #pragma omp parallel for
    for (int i = 0; i < blocks_count; i++) {
        for (int j = i*block_size; j < (i + 1)*block_size; j++) {
            summs[i] += a[j];
        }
    }

    for (int i = 0; i < blocks_count; i++) {
        summ += summs[i];
    }

    return summ;
}
```

```cpp
double atomic_summ(const std::vector<double> & a) {
    double summ = 0.0;
    int n = a.size();

    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        #pragma omp atomic
        summ += a[i];
    }

    return summ;
}

void task2(int n) {
    std::cout << "Calculate summ of array's elements\n"
        << "Size of array is " << n << std::endl;
    std::vector<double> a(n, 1.0);

    double start_time, end_time, execution_time;
    std::vector<calculate_summ> functions = { continious_summ, atomic_summ, concurrent_summ,
bisection_summ };
    std::vector<std::string> functions_names = { "[Contitious summ]", "[Atomic summ]", "[Concurrent
summ]", "[Bisection summ]" };

    double result = 0.0;
    for (int i = 0; i < functions.size(); i++) {
       start_time = omp_get_wtime();
       result = functions[i](a);
       end_time = omp_get_wtime();

       execution_time = end_time - start_time;
       std::cout << "Function " << functions_names[i]
           << " returned result: " << result
           << " after " << execution_time << std::endl;
    }
}


// Ordered
void task3(int n) {
    std::cout << "Calculate summ of vectors a and b and store it in vector c" << std::endl
        << "Size of vectors is: " << n << std::endl;
    std::vector<double> a(n, 1.0), b(n, 1.0), c(n, 1.0);

    double start_time, end_time = 0;

    start_time = omp_get_wtime();
    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        c[i] = a[i] + b[i];
    }
    end_time = omp_get_wtime();
    std::cout << "[Unordered] time: " << end_time - start_time << std::endl;

    start_time = omp_get_wtime();
    #pragma omp parallel for ordered
    for (int i = 0; i < n; i++) {
        c[i] = a[i] + b[i];
    }
    end_time = omp_get_wtime();
    std::cout << "[Ordered] time: " << end_time - start_time << std::endl;
}
```

```cpp
// Static, dynamic, guided (10^3, 10^5)
void task4(int n, int schedule_size) {
    std::cout << "Calculate summ of vectors a and b and store it in vector c" << std::endl
        << "Size of vectors is: " << n << std::endl
        << "Shedule size is " << schedule_size << std::endl;
    std::vector<double> a(n, 1.0), b(n, 1.0), c(n, 1.0);

    double start_time, end_time = 0;

    start_time = omp_get_wtime();
    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        c[i] = a[i] + b[i];
    }
    end_time = omp_get_wtime();
    std::cout << "[No shedulling] time: " << end_time - start_time << std::endl;

    start_time = omp_get_wtime();
    #pragma omp parallel for schedule(static, schedule_size)
    for (int i = 0; i < n; i++) {
        c[i] = a[i] + b[i];
    }
    end_time = omp_get_wtime();
    std::cout << "[Static sheduling] time: " << end_time - start_time << std::endl;

    start_time = omp_get_wtime();
    #pragma omp parallel for schedule(dynamic, schedule_size)
    for (int i = 0; i < n; i++) {
        c[i] = a[i] + b[i];
    }
    end_time = omp_get_wtime();
    std::cout << "[Dynamic sheduling] time: " << end_time - start_time << std::endl;

    start_time = omp_get_wtime();
    #pragma omp parallel for schedule(guided, schedule_size)
    for (int i = 0; i < n; i++) {
        c[i] = a[i] + b[i];
    }
    end_time = omp_get_wtime();
    std::cout << "[Guided sheduling] time: " << end_time - start_time << std::endl;
}


double math_function(double x) {
    //srand(NULL);
    return pow(sin(x) + cos(x), 2) + pow(sin(x) + cos(x), 4) + sqrt(exp(x)); // +rand();
}


// Individual task
std::vector<double> create_range(int range_size, double from, double to) {
    std::vector<double> range(range_size, 0.0);
    double range_step = (to - from) / (double)range_size;
    for (int i = 0; i < range_size; i++) {
        range[i] = i * range_step;
    }

    return range;
}
```

```
void individual_task(int range_size, int functions_count = 4, double from = 0.0, double to = 15.0) {
    std::cout << "Calculate values for some functions on range in different threads (func per
thread)\n"
        << "Range size is: " << range_size << '\n'
        << "Count of functions is: " << functions_count << std::endl;

    std::vector<double> range = create_range(range_size, from, to);
    std::vector<std::vector<double> > functions_values(functions_count, std::vector<double>
(range_size, 0.0));

    double start_time, end_time;

    // Parallel version
    start_time = omp_get_wtime();
    #pragma omp parallel for
    for (int i = 0; i < functions_count; i++) {
        for (int j = 0; j < range_size; j++) {
            functions_values[i][j] = math_function(range[j]); /*functions[i](range[j]);*/
        }
    }
    end_time = omp_get_wtime();
    std::cout << "Task for range size (parallel version) " << range_size
        << " was calculated for " << end_time - start_time << std::endl;


    // Continious version
    start_time = omp_get_wtime();
    for (int i = 0; i < functions_count; i++) {
        for (int j = 0; j < range_size; j++) {
            functions_values[i][j] = math_function(range[j]); /*functions[i](range[j]);*/
        }
    }
    end_time = omp_get_wtime();
    std::cout << "Task for range size " << range_size
        << " was calculated for " << end_time - start_time << std::endl;
}


void test_task2() {
    std::cout << "Task 2. Testing Continious, Atomic, Concurrent and Bisection methods" <<
std::endl;
    task2(pow(10, 6));
    std::cout << std::endl << std::endl;
    task2(pow(10, 7));
}

void test_task3() {
    std::cout << "Task 3. Testing ordered" << std::endl;
    task3(pow(10, 6));
}

void test_task4() {
    std::cout << "Task 4. Testing sheduling" << std::endl;
    task4(pow(10, 6), pow(10, 3));
    std::cout << std::endl << std::endl;
    task4(pow(10, 6), pow(10, 5));
}

void test_individual_task() {
    int func_count = omp_get_num_procs();
    std::cout << "Testing individual task" << std::endl;
    individual_task(pow(10, 6), func_count);
    std::cout << std::endl << std::endl;
    individual_task(pow(10, 6), func_count*2);
}

void put_delimiter() {
    std::cout << std::endl << "================================================" << std::endl;
}
```

```
int main() {
    test_task2(); put_delimiter();
    test_task3(); put_delimiter();
    test_task4(); put_delimiter();
    test_individual_task();
    system("pause");
}
```