Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

# ОБРАБОТКА И ВЫПОЛНЕНИЕ ПРОГРАММ В OPENMP

Отчёт о лабораторной работе № 4 по дисциплине «Параллельное программирование»

Студент гр. 431-3

\_\_\_\_\_\_\_ Д.П. Андреев

«\_\_» \_\_\_\_\_\_\_\_\_ 2024

Проверил

Доцент каф. АСУ, к.т.н

\_\_\_\_\_\_\_ С.М. Алфёров

«\_\_» \_\_\_\_\_\_\_\_\_ 2024

Томск 2024

**1 Цель лабораторной работы**

Цель: освоить применение основных директив, функций и переменных окружения OpenMP на примере параллельной программы численного интегрирования.

## 2 Задание

Задание на лабораторную работу: Используя OpenMP и программу из первой лабораторной работы распараллелить вычисление интеграла с задачей количества потоков в аргументе запуска программы.

### 3 Использованные OpenMP функции

В программе для численного интегрирования были использованы несколько ключевых функций и директив OpenMP, которые обеспечивают параллельное выполнение, управление потоками и сбор результатов вычислений.

1. **omp\_set\_num\_threads(num\_threads)**: Эта функция позволяет задать количество потоков, которые будут участвовать в параллельных регионах программы. Мы использовали её для того, чтобы программа могла принимать количество потоков в качестве аргумента командной строки и запускаться с разным числом потоков, что даёт возможность гибкого управления параллелизацией. Если не указать это явно, OpenMP использует стандартное число потоков, определённое системой.
2. **omp\_get\_num\_threads()**: Вызов этой функции внутри параллельного региона позволяет узнать текущее количество потоков, которые участвуют в выполнении параллельного участка программы. В нашем случае функция была использована внутри блока #pragma omp single для вывода информации о количестве потоков только один раз, но уже после того, как они были созданы.
3. **omp\_get\_max\_threads()**: Функция возвращает максимальное возможное количество потоков, доступных на данной машине. Это полезно для диагностики и проверки того, сколько потоков может быть задействовано системой в данном окружении.
4. **omp\_get\_wtime()**: Эта функция возвращает текущее время в секундах, отмеренное с начала некоторой точной временной точки. Мы использовали её для измерения времени выполнения программы как до, так и после параллельных вычислений, что позволяет оценить производительность программы и эффект параллелизации.
5. **omp parallel**: Директива #pragma omp parallel создаёт параллельный регион, в котором каждый поток выполняет копию кода, заключённого в этот регион. Все потоки работают одновременно, а общие переменные могут разделяться между ними. Мы использовали этот регион для параллельного выполнения вычислений численного интеграла.
6. **omp for**: Директива #pragma omp for была использована для распределения итераций цикла по потокам. Она автоматически распределяет работу между потоками в параллельном регионе, чтобы каждый поток выполнял часть цикла. В нашем случае цикл, который вычисляет значения интеграла, был распределён между потоками, что ускоряет процесс интегрирования.
7. **schedule(static, chunk)**: Директива schedule задаёт способ распределения итераций цикла между потоками. В нашей программе мы использовали статическое распределение с размером блока chunk, равным 1000. Это означает, что каждый поток получает фиксированное количество итераций (1000) для выполнения. Статическое распределение помогает, если итерации цикла выполняются примерно за одинаковое время.

**reduction(+)**: Директива reduction позволяет объединить частные переменные, создаваемые в каждом потоке, в одну общую переменную. В нашем случае мы использовали редукцию для переменной sum, которая накапливает результат частных сумм, вычисленных каждым потоком. Операция + обозначает, что каждый поток будет добавлять свой вклад в общую переменную суммирования.

**4 Листинг программы**

Main.cpp:

#include <omp.h>

#include <cmath>

#include <iostream>

#include <cstdlib> // Для функции atoi

static constexpr double f(double a) {

    double c = 0.8;

    return sin(c \* a) \* cos(c \* a);

}

static constexpr double fi(double a) {

    double c = 0.8;

    return (1 / (2 \* c)) \* pow(sin(c \* a), 2);

}

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

    int n = 100000000; // Количество шагов для интегрирования

    double xl = -0.5, xh = 0.8; // Границы интегрирования

    double h = (xh - xl) / (double)n; // Шаг интегрирования

    double sum = 0.0;

    int num\_threads = 4; // Число потоков по умолчанию

    // Если указано число потоков через аргументы командной строки

    if (argc > 1) {

        num\_threads = atoi(argv[1]); // Чтение количества потоков

    }

    omp\_set\_num\_threads(num\_threads); // Устанавливаем число потоков

    double start\_time = omp\_get\_wtime(); // Время начала

    // Параллельный регион с директивой for и редукцией

    #pragma omp parallel

    {

        #pragma omp single

        {

            std::cout << "С использованием " << omp\_get\_num\_threads() << " потоков." << std::endl;

        }

        #pragma omp for reduction(+:sum) schedule(static, 1000)

        for (int i = 1; i <= n; i++) {

            double x = xl + h \* (i - 0.5);

            sum += f(x);

        }

    }

    double result = h \* sum; // Итоговая сумма

    double end\_time = omp\_get\_wtime(); // Время завершения

    // Вывод результатов в мастер-потоке

    std::cout << "Максимальное количество доступных потоков: " << omp\_get\_max\_threads() << std::endl;

    std::cout << "Результат интегрирования: " << result << std::endl;

    std::cout << "Ошибка: " << result - (fi(xh) - fi(xl)) << std::endl;

    std::cout << "Потраченое время: " << end\_time - start\_time << " seconds." << std::endl;

    return 0;

}

### 5 Примеры работы программы

Разберём работу программы. Пример работы изображён на рисунке 5.1.

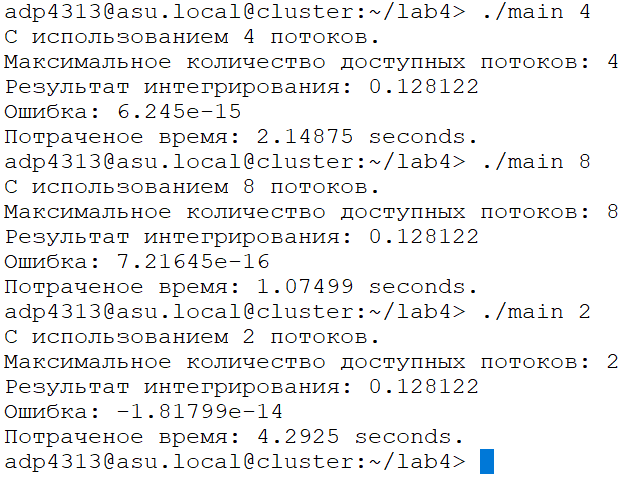


Рисунок 5.1 — Пример работы программы

На рисунке изображён процесс компиляции и вычисления интеграла согласно варианту в различное количество потоков.

### 6 Выводы

Таким образом, я изучил основы работы с технологией OpenMP. С помощью этого способа я решил задачу решения интеграла из первой лабораторной работы.