Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
по курсу «Анализ алгоритмов»

«Параллельное программирование»

Студент Маслова Марина Дмитриевна

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Волкова Лилия Леонидовна

2021 г.

Содержание

Введение

Сегодня программирование используется во многих научных и социальных областях. Компьютерам требуется производить все более трудоемкие вычисления на больших объемах данных. При этом предъявляются требования к скорости вычислений.

Одним из возможных решений увеличения производительности компьютера, то есть скорости решения задач является параллельное программирование. На одном устройстве паралельные вычисления можно организовать с помощью **многопоточности** – способности цетрального процессора или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой. При последовательной реализации какого-либо алгоритма, его программу выполняет только одно ядро процессора. Если же реализовать алгоритм так, что независимые вычислительные задачи смогут выполнять несколько ядер параллельно, то это приведет к ускорению решения всей задачи в целом[1].

Для реализации паралельных вычислений требуется выделить те участки алгоритма, которые могут выполняться паралльно без изменения итогового резульатат, также необходми правильно организовать работу с данными, чтобы не потерять вычисленные значения.

Одной из распространненых задач, решение которой используется в различных областях, является численное итегрирование, поэтому в данной лабораторной работе будет предпринята попытка ускорить вычисления определенных интегралов.

Целью данной работы является получение навыков параллельного программирования на основе алгоритма численного интегрирования методом средних прямоугольников.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

- изучить метод средних прямоугольников для численного интегрирования;
- описать возможности распараллеливания данного алгоритма;
- разработать последовательный и параллельный алгоритмы;

– реализовать каждый алгоритм;
– провести тестирование реализованных алгоритмов;
– провести сравнительный анализ алгоритмов по времени работы реализаций;
– сделать выводы по полученным результатам.

1 Аналитическая часть

В данном разделе представлено теоретическое описание алгоритмов численного интегрирования методом средних прямоугольников.

1.1 Последовательный алгоритм

Пусть требуется вычислить определенный интеграл 1.1:

$$I = \int_a^b f(x)dx, \tag{1.1}$$

Суть всех методов численного интегрирования, в том числе и метода средних прямоугольников, состоит в замене подынтегральной функции $f(x)$ вспомогательной, интеграл от которой легко вычисляется в элементарных функциях.

Предположим, что $f(x)$ непрерывна на $[a, b]$, n – натуральное и $\Delta x = \frac{b-a}{n}$. Разделим интервал $[a, b]$ на n подынтервалов длиной Δx и найдем среднюю точку m_i каждого i-ого подыинтервала[2]. Тогда определенный интеграл может быть вычислен по формуле 1.2:

$$I_n = \sum_{i=1}^{n} f(m_i)\Delta x, \tag{1.2}$$

При этом, чем больше n, тем ближе вычисленное значение, к реальному значению интеграла, то есть 1.3:

$$I = \lim_{n\to\infty} I_n, \tag{1.3}$$

Понятно, что с технической точки зрения нельзя разделить интервал на бесконесное число подынтервалов. Это и не требуется, так как необходимой точности вычислений можно достичь и при конечном n. Для этого интервал сначала разбивают на m и $m + 1$ (начиная с $m = 2$) подынтервала, применяют численный метод для каждого количества и вычисляют разницу между ними, если разница меньше заданной точности ε, то вычисления прекращают, а результатом является последнее вычисленное занчение.

1.2 Параллельный алгоритм

В алгоритме численного интегрирования методом средних прямоугольников вычисления на каждом из подынтервалов происходят независимо, поэтому есть возможность произвести распараллеливание данных вычислений. Количество отрезков, на которых производит вычисление один поток, будет определяться количеством потоков, а итоговое значение интеграла будет храниться в разделяемой переменной, доступ к которой будут иметь все потоки, каждый из которых будет прибавлять вычисленную им сумму к итоговому результату.

1.3 Вывод

В данном разделе был рассмотрен алгоритм численного интегрирования методом средних прямоугольнико, так же был описан механизм распараллеливания данного алгоритма. Из представленных описаний можно предъявить ряд требований к разрабатываемому программному обеспечению:

- на вход должны подаваться пределы интегрирования, заданная точность, а также число потоков для параллельного алгоритма;
- на выходе должны выдаваться вычисленные значения определенного интеграла каждым из алгоритмов, причем результаты должны совпадать;
- интегрируемые функции могут быть предложены пользователю на выбор.

2 Конструкторская часть

В данном разделе разрабатываются последовательный и параллельный алгоритмы, численного интегрирования методом средних прямоугольников, структура программы и способы её тестироваия.

2.1 Разработка алгоритмов

На рисунках 2.1-2.2 представлена схема алгоритма последовательного алгоритма численного интегрирования методом средних прямоугольников с заданной точностью, на рисунках 2.3-2.5 — схема параллельного алгоритма.

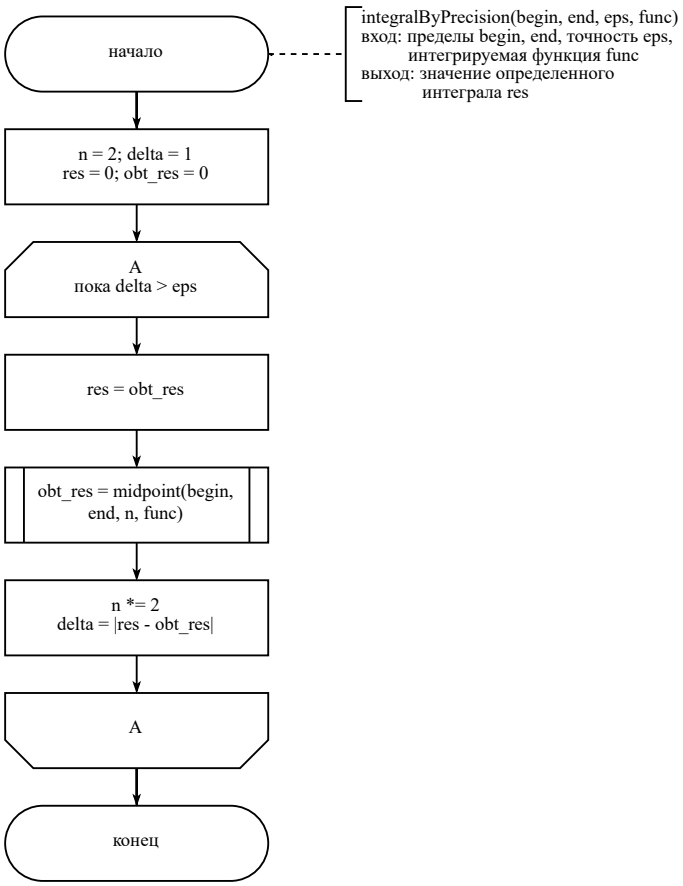

Рисунок 2.1 – Схема последовательного алгоритма численного интегрирования с заданной точностью

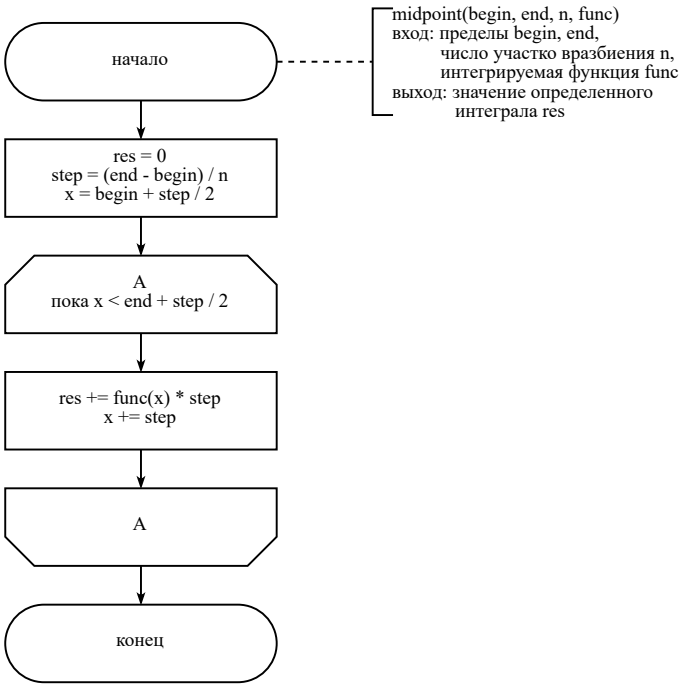

Рисунок 2.2 – Схема последовательного алгоритма численного интегрирования методом средних прямоугольников при заданном количестве участков разбиения

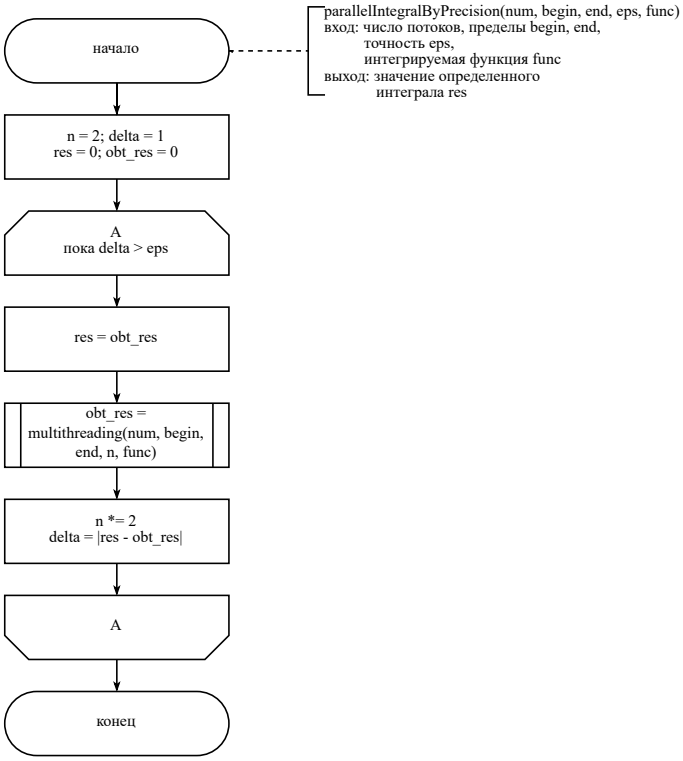

Рисунок 2.3 – Схема параллельного алгоритма численного интегрирования с заданной точностью

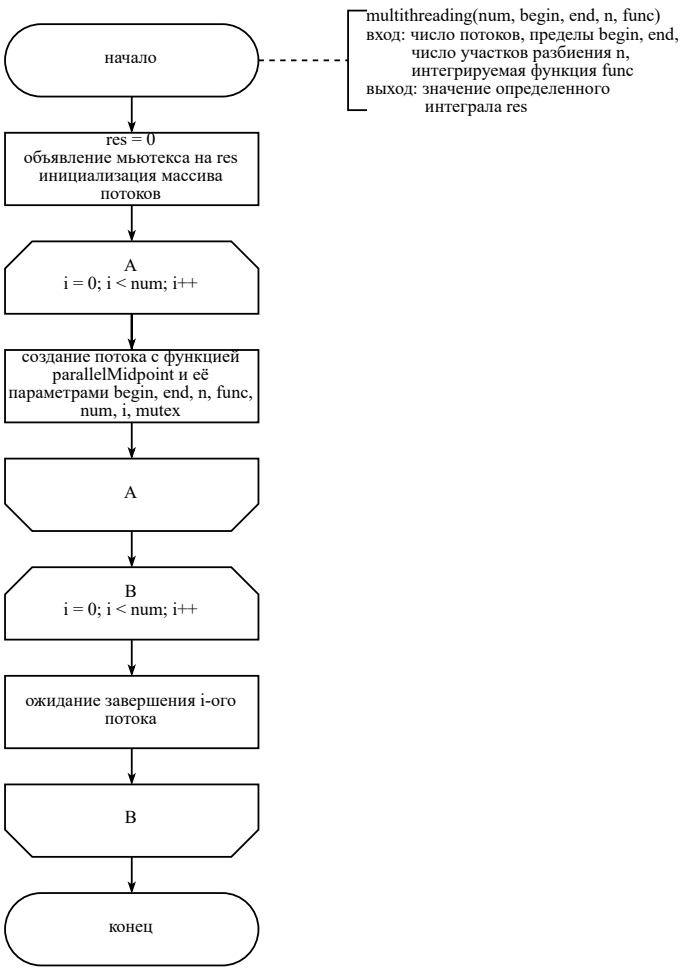

Рисунок 2.4 – Создание потоков для параллельного алгоритма численного интегрирования

2.2 Структура разрабатываемого ПО

Для реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Каждый из алгоритмов будет представлен отдельной функцией, при необходимости будут выделены подпрограммы для каждой из них. Также будут реализованы функции для ввода-вывода и функция, вызывающая все подпрограммы для связности и полноценности программы.

2.3 Классы эквивалентности при тестировании

Для тестирования программного обеспечения во множестве тестов будут выделены следующие классы эквивалентности:

– совпадение верхнего и нижнего пределов интегрирования;

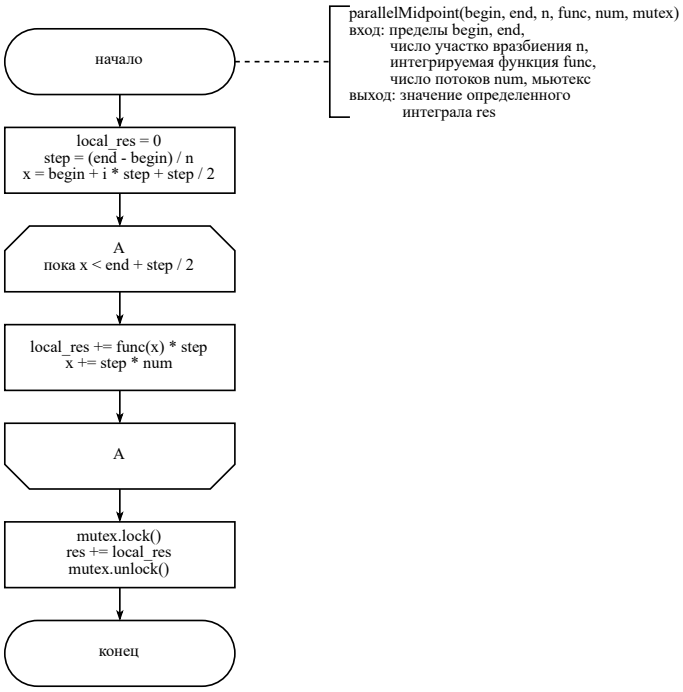

Рисунок 2.5 – Схема параллельного алгоритма численного интегрирования методом средних прямоугольников при заданном количестве участков разбиения

– положительные пределы интегрирования;
– верхний предел интегрирования меньше нижнего предела;
– произвольные пределы интегрирования.

2.4 Вывод

В данном разделе были разработаны последовательный и параллельный алгоритмы, была описана структура разрабатываемого ПО. Для дальнейшей проверки правильности работы программы были выделены классы эквивалентности тестов.

3 Технологическая часть

В данном разделе описаны требования к программному обеспечению, средства реализации, приведены листинги кода и данные, на которых будет проводиться тестирование.

3.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

- выбор режима работы: для единичного эксперимента и для массовых экспериментов;
- в режиме единичного эксперимента выбор функции для интегрирования, ввод пределов интегрирования, точности и числа потоков для параллельной реализации;
- в режиме массовых экспериментов измерение времени работы каждого из алгоритмов в зависимости от точности и числа потоков.

3.2 Средства реализации

Для реализации данной лабораторной работы выбран компилируемый язык програмирования `C++`[3], так как он предоставляет необходимые библиотеки для работы с потоками. Интерпретируемый язык программирования высокого уровня Python[4] был выбран для визуализации данных эксперимента, так как он предоставляет большое число настроек параметров графика с использованием простого синтаксиса.

В качестве среды разработки выбран текстовый редактор Vim[5] с установленными плагинами автодополнения и поиска ошибок в процессе написания, так как он реализует быстрое перемещение по тексту программы и простое взаимодействие с командной строкой.

Замеры времени проводились при помощи функции `std::chrono::system_clock::now()` из библиотеки chrono[6].

3.3 Листинги кода

В данном подразделе представлены листинги кода ранее описанных алгоритмов:

– последовательный алгоритм численного итегрирования методом средних прямоугольников с заданной точностью (листинги 3.1-3.2);
– параллельный алгоритм численного итегрирования методом средних прямоугольников с заданной точностью (листинг 3.3-3.5).

Листинг 3.1 – Функция интегрирования методом средних прямоугольников с заданным числом участков разбиения (последовательный алгоритм)

```
1  long double midpoint(double begin, double end, unsigned int n, function_t func)
2  {
3      double res = 0;
4      double step = (end - begin) / n;
5      double x = begin + step / 2;
6
7      while (x < end + step / 2)
8      {
9          res += func(x) * step;
10         x += step;
11     }
12
13     return res;
14 }
```

Листинг 3.2 – Функция вычисления интеграла с заданной точностью (последовательный алгоритм)

```
long double integralByPrecision(interval_t &interval, function_t func)
{
    unsigned long n = 2;
    long double delta = 1;
    long double res = 0, obt_res = 0;

    while (delta > interval.eps)
    {
        res = obt_res;
        obt_res = midpoint(interval.begin, interval.end, n, func);
        n *= 2;
        delta = abs(res - obt_res);
    }

    return res;
}
```

Листинг 3.3 – Функция интегрирования методом средних прямоугольников с заданным числом участков разбиения (параллельный алгоритм)

```
void parallelMidpoint(double begin, double end, unsigned int n, function_t func,
                      long double &res, int threads_num, int i, mutex &mut)
{
    double local_res = 0;
    double step = (end - begin) / n;
    double x = begin + i * step + step / 2;

    while (x < end + step / 2)
    {
        local_res += func(x) * step;
        x += step * threads_num;
    }

    mut.lock();
    res += local_res;
    mut.unlock();
}
```

Листинг 3.4 – Функция создания потоков для интегрирования методом средних прямоугольников

```
long double multithreading(int threads_num, interval_t &interval,
                           unsigned int n, function_t func)
{
    long double res = 0;
    mutex res_mut;

    vector<thread> threads(threads_num);

    for (int i = 0; i < threads_num; i++)
    {
        threads[i] = thread(parallelMidpoint, interval.begin, interval.end, n,
                            func, ref(res), threads_num, i, ref(res_mut));
    }

    for (int i = 0; i < threads_num; i++)
    {
        threads[i].join();
    }

    return res;
}
```

Листинг 3.5 – Функция вычисления интеграла с заданной точностью (параллельный алгоритм)

```
long double integralByPrecision(interval_t &interval, function_t func)
{
    unsigned long n = 2;
    long double delta = 1;
    long double res = 0, obt_res = 0;

    while (delta > interval.eps)
    {
        res = obt_res;
        obt_res = midpoint(interval.begin, interval.end, n, func);
        n *= 2;
        delta = abs(res - obt_res);
    }

    return res;
}
```

3.4 Описание тестирования

В таблице 3.1 приведены функциональные тесты для алгоритмов интегрирования на функции $f(x) = x^2$.

Таблица 3.1 – Функциональные тесты

Пределы интегрирования	**Ожидаемый результат**
0 0	0
0 1	0.3333
1 0	-0.3333
-1 1	0.6666

3.5 Вывод

В данном разделе были реализованы последовательный и параллельный алгоритмы численного интегрирования методом средних прямоугольников. Также были написаны тесты для каждого класса эквивлентности, описанного в конструкторском разделе.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

– Операционная система: Manjaro [7] Linux x86_64.

– Память: 8 GiB.

– Процессор: Intel® Core™ i5-8265U, 4 физических ядра, 8 логических ядра[8].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

4.2 Примеры работы программы

На рисунке 4.1 представлены результаты работы программы.

```
РАСПАРАЛЛЕЛИВАНИЕ АЛГОРИТМА ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ

     МЕНЮ
1 -- подсчет интеграла;
2 -- сравнение последовательного и параллельного алгоритмов;
0 -- выход.

Выбор:
1
Функции:
     1 -- x^2;
     2 -- x * sin x;
     3 -- √(e^(sin x * cos x) + 6 * |x + 6|) +
          + ln|x^3 - 3x + 1|;
Выберете функцию: 3
Введите нижний предел: -10
Введите верхний предел: 10
Введите количество знаков после запятой: 6
Введите количество потоков: 4

ЗНАЧЕНИЕ ИНТЕГРАЛА
Последовательный алгоритм: 198.740798
Время: 812399544 нс
Параллельный алгоритм: 198.740798
Время: 306246758 нс
```

Рисунок 4.1 – Пример работы программы

4.3 Результаты тестирования

Программа была протестирования на входных данных, приведенных в таблице 3.1. Полученные результаты работы программы совпали с ожидаемыми результатами.

4.4 Постановка эксперимента по замеру времени

Для оценки времени работы последовательной и параллельной реализации алгоритма численного интегрирования методом средних прямоугольников был проведен эксперимент, в котором определялось влияние количества потоков и точности вычислений на время работы алгоритмов. Тестирование проводилось на количестве потоков, равном степеням 2 от 1 до 64, и на точности вычислений от 1 до 7 знаков после запятой. Так как от запуска к запуску время, затрачиваемое на выполнение алгоритма, менялось в определенном промежутке, необходимо было усреднить вычисляемые значения. Для этого каждый алгоритм в каждом случае запускался по 10 раз, и для полученных 10 значений определялось среднее арифметическое, которое заносилось в таблицу результатов.

Результаты эксперимента были представлены в виде таблиц и графиков, приведенных в следующем подразделе.

4.5 Результаты эксперимента

В таблице 4.1 представлены результаты измерения времени работы параллельного алгоритма в зависимости от числа потоков. На рисунке представлен соответствующий график, для сравнения на графике приведено также время последовательной реализации.

В таблице 4.2 представлены результаты измерения времени работы последовательного и паралелльного алгоритмов в зависимости от числа знаков после запятой, соответствующие заданной точности вычислений интеграла. На рисунке представлен соответствующий график, для наглядости на графике представлены только точки с точностью от 4 до 7 знаков после запятой.

Таблица 4.1 – Время работы от числа потоков (* – последовательный)

Число потоков	**Время, нс**
*	880622058
1	839013147
2	446833477
4	338605815
8	300881333
16	317716474
32	622792430
64	1561156672

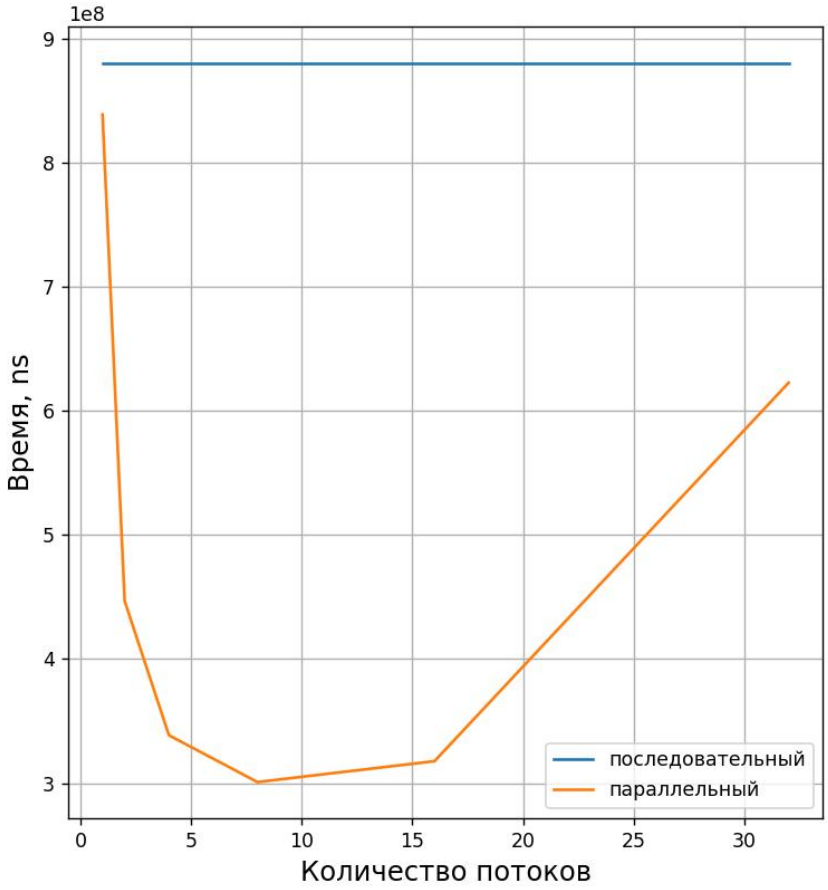

Рисунок 4.2 – График зависимости времени работы от числа потоков

Таблица 4.2 – Время работы от точности

Точность	**8 потоков, нс**	**Последовательный, нс**
1e-1	24051708	167321
1e-2	161824161	1063368
1e-3	264771774	8724506
1e-4	364664568	23780663
1e-5	666893339	849861484
1e-6	967859493	1701109328
1e-7	1272447909	2549415743

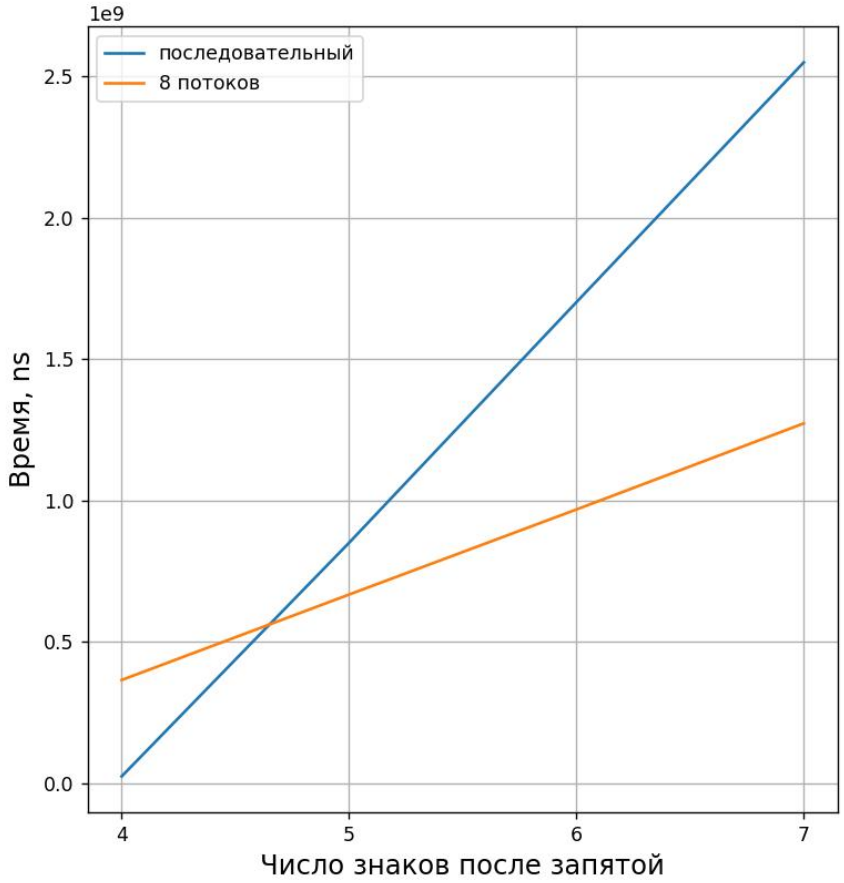

Рисунок 4.3 – График зависимости времени работы реализаций от точности

4.6 Вывод

По результатам эксперимента можно сделать следующие выводы:

- при движении от наименьшего числа потоков к числу потоков равному числу логических ядер процессора время уменьшается;
- при превышении числа логических ядер время увеличивается, так как затрачивается время на переключение ядра между потоками;
- последовательная реализация работает в 3 раза медленнее параллельной реализации на оптимальном числе потоков (8);
- при этом параллельная реализация только при числе потоков большем 32 работает медленнее, что говорит о том, что при таком числе реализуемых потоков выигрыш от распараллеливания перекрывается временем, затрачиваемым на переключения между потоками;
- в зависимости от точности выигрыш от распараллеливания начинает проявляться только при точности от 5 и выше знаков после запятой (выигрыш в 2 раза); это значит, что меньшей точности время, затрачиваемое на создание и запуск потоков, больше чем выигрыш, получаемый от распараллеливания.

Таким образом, для достижения наибольшей скорости вычислений необходимо использовать число потоков равное числу логических потоков на машине. При этом нецелесообразно использовать параллельную реализацию при проведении вычислений интегралов с точностью ниже, чем 5 знаков после запятой. В таком случае, необходимо либо использовать параллельную реализацию, либо уменьшить число потоков, чтобы затрачивать меньше времени на их создание.

Заключение

В ходе исследования был проведен сравнительный аналализ последовательной и параллельной реализации алгорима численного интегрирования методом средних прямоугольников. В результате исследования было выяснено, что при распараллеливании можно добиться улучшения скорости вычислений в 3 раза, но при условии использования оптимального числа потоков, равного числу логических ядер процессора, и вычисления интегралов в точностью от 5 знаков после запятой.

В ходе выполения лабораторной работы:

- были описаны и разработаны последовательный и параллельный алгоритм численного интегрирования методом средних прямоугольников;
- был реализован каждый из описанных алгоритмов;
- по экспериментальным данным были сделаны выводы об эффективности по времени каждого из реализованных алгоритмов;
- были получены зависимости времени работы паралелльного алгоритма от числа потоков и времени работы каждого из алгритмов от точности вычислений.

Таким образом, все поставленные задачи были выполнены, а цель достигнута.

Список литературы

[1] Многопоточность [Электронный ресурс]. Режим доступа: https://ru.coursera.org/lecture/ios-multithreading/chto-takoie-mnoghopotochnost-4MMgN (дата обращения: 02.12.2021).

[2] Numerical Integration - Midpoint, Trapezoid, Simpson's rule [Электронный ресурс]. Режим доступа: https://math.libretexts.org/Courses/Mount_Royal_University/MATH_2200%3A_Calculus_for_Scientists_II/2%3A_Techniques_of_Integration/2.5%3A_Numerical_Integration_-_Midpoint%2C_Trapezoid%2C_Simpson's_rule (дата обращения: 02.12.2021).

[3] Документация по языку C++ [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170 (дата обращения: 02.12.2021).

[4] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 12.10.2021).

[5] welcome home : vim online [Электронный ресурс]. Режим доступа: https://www.vim.org/ (дата обращения: 12.10.2021).

[6] Date and time utilities [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono (дата обращения: 02.12.2021).

[7] Manjaro — enjoy the simplicity [Электронный ресурс]. Режим доступа: https://manjaro.org/ (дата обращения: 17.10.2021).

[8] Процессор Intel® Core™ i5-8265U [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/149088/intel-core-i5-8265u-processor-6m-cache-up-to-3-90-ghz.html (дата обращения: 17.10.2021).