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”

на тему:

«Параллельное программированиеs»

Студент **Голикова С. М.**

Группа **ИУ7-55Б**

Оценка (баллы)

Преподаватели **Волкова Л. Л., Строганов Ю. В.**

Москва, 2022г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Многопоточность — способность центрального процессора или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой. Этот подход отличается от многопроцессорности, так как многопоточность процессов и потоков совместно использует ресурсы одного или нескольких ядер: вычислительных блоков, кэш-памяти ЦПУ или буфера перевода с преобразованием. 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.

В тех случаях, когда многопроцессорные системы включают в себя несколько полных блоков обработки, многопоточность направлена на максимизацию использования ресурсов одного ядра, используя параллелизм на уровне потоков, а также на уровне инструкций. Поскольку эти два метода являются взаимодополняющими, их иногда объединяют в системах с несколькими многопоточными ЦП и в ЦП с несколькими многопоточными ядрами.

Для реализации паралельных вычислений требуется выделить те участки алгоритма, которые могут выполняться паралльно без изменения итогового результата. Также необходимо организовать корректную работу с данными, чтобы не потерять вычисленные значения.

Целью данной лабораторной работы является получение навыков параллельного программирования на основе алгоритмов численного интегрирования методом средних прямоугольников и методом трапеций.

В рамках выполнения работы необходимо решить следующие задачи:

— изучить методы средних прямоугольников и трапеций для численного

интегрирования;

— описать возможности распараллеливания данных алгоритмов;
— разработать последовательный и параллельный алгоритмы;
— реализовать каждый алгоритм;
— провести тестирование реализованных алгоритмов;
— провести сравнительный анализ алгоритмов по времени работы реализаций.

1 Аналитический раздел

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

1.1 Метод средних прямоугольников

Пусть требуется вычислить следующий определенный интеграл:

$$I = \int_a^b f(x)dx, \tag{1.1.1}$$

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 каждый и найдем среднюю точку m_i каждого i-ого подыинтервала [1]. Тогда определенный интеграл может быть вычислен по следующей формуле:

$$I_n = \sum_{i=1}^{n} f(m_i)\Delta x, \tag{1.1.2}$$

1.2 Метод трапеций

Пусть требуется вычислить следующий определенный интеграл:

$$I = \int_a^b f(x)dx, \tag{1.2.1}$$

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 каждый. Множество точек, задающих подынтервалы, $P = \{x_0, x_1, \ldots, x_n\}$ [1]. Тогда определенный интеграл может быть вычислен по следующей формуле:

$$I_n = \frac{1}{2}\Delta x(f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n), \tag{1.2.2}$$

1.3 Достижение заданной точности

Чем больше количество подынтервалов n, тем ближе вычисленное значение к реальному значению интеграла, то есть справедлива формула 1.3.1.

$$I = \lim_{n \to \infty} I_n, \tag{1.3.1}$$

Понятно, что с технической точки зрения нельзя разделить интервал на бесконечное 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 значение.

1.4 Параллельные алгоритмы

В алгоритмах численного интегрирования методом средних прямоугольников и методом трапеций 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.

Вывод

В данном разделе были рассмотрены алгоритмы численного интегрирования методом средних прямоугольников и методом трапеций. Кроме того, был описан механизм распараллеливания данных алгоритмов.

2 Конструкторский раздел

В данном разделе представлены последовательный и параллельный алгоритмы численного интегрирования методом средних прямоугольников и методом трапеций.

2.1 Разработка алгоритмов

На рисунках 1-3 представлены схемы последовательных алгоритмов численного интегрирования, на рисунках 4-7 – схемы параллельных алгоритмов.

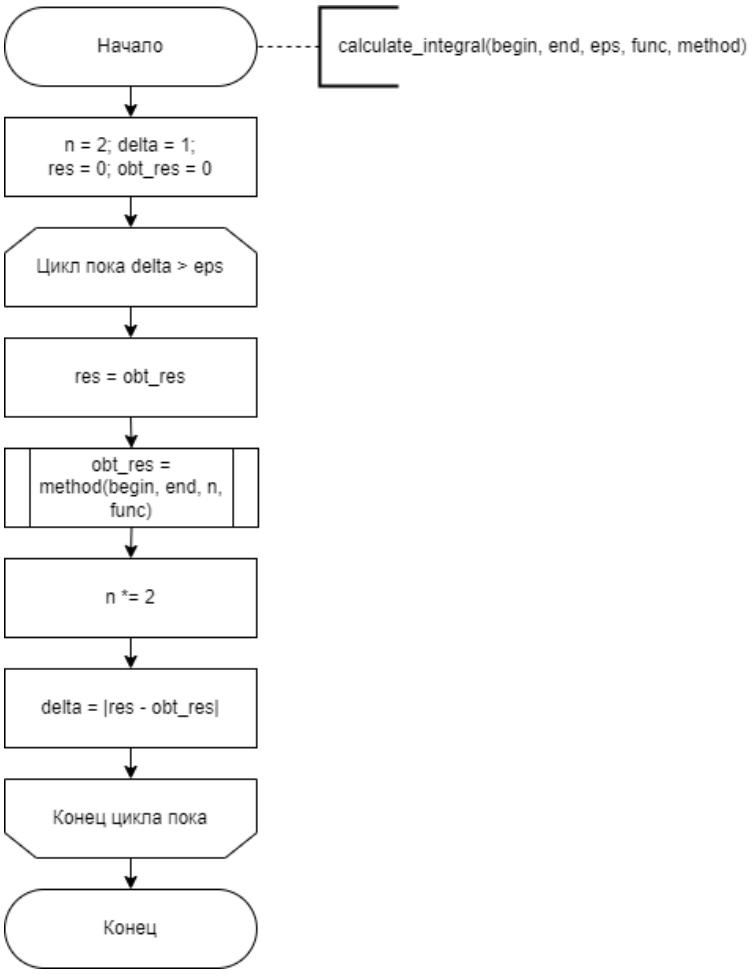

Рисунок 1 – Схема последовательного алгоритма численного интегрирования с заданной точностью

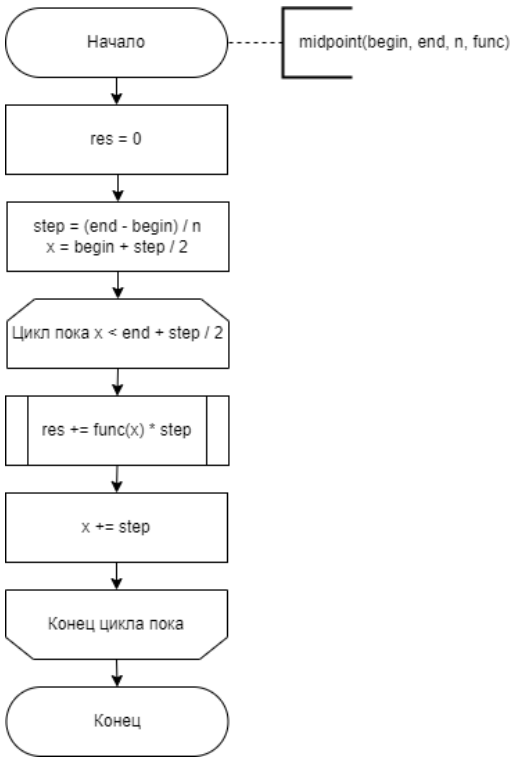

Рисунок 2 – Схема последовательного алгоритма численного интегрирования методом средних прямоугольников при заданном n

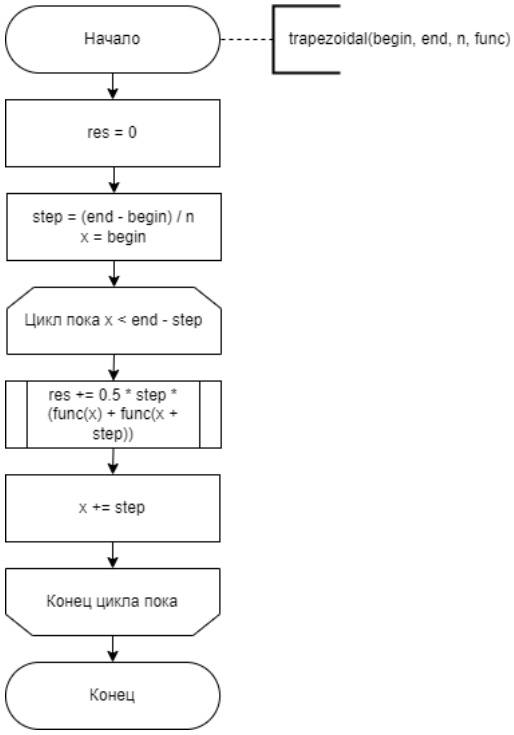

Рисунок 3 – Схема последовательного алгоритма численного интегрирования методом трапеций при заданном n

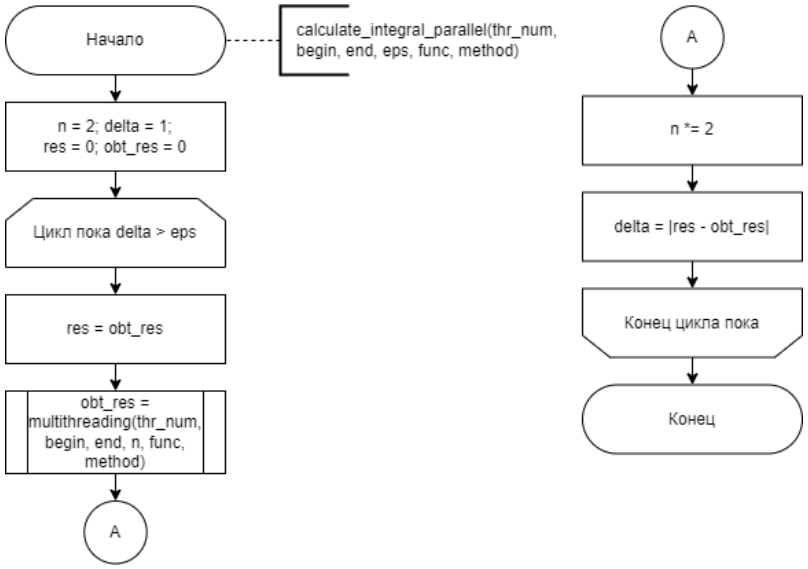

Рисунок 4 – Схема параллельного алгоритма численного интегрирования с заданной точностью

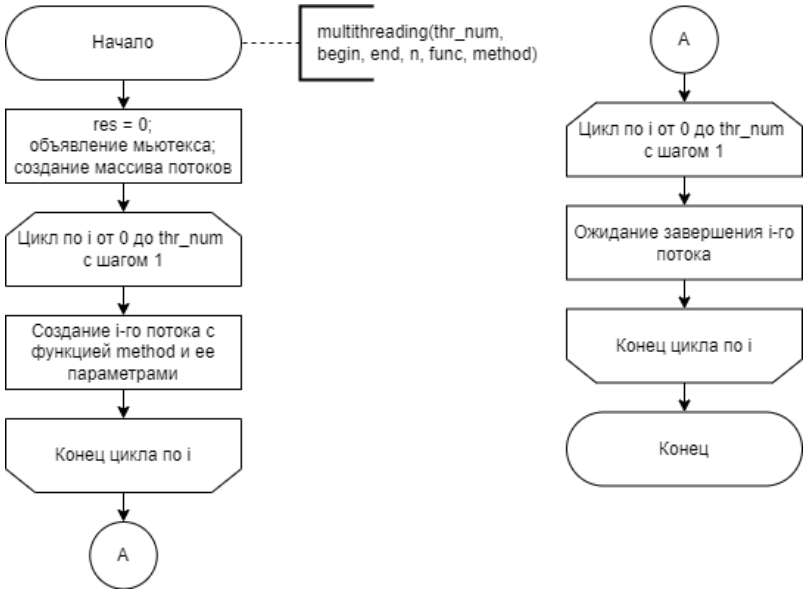

Рисунок 5 – Схема создания потоков для параллельного алгоритма численного интегрирования

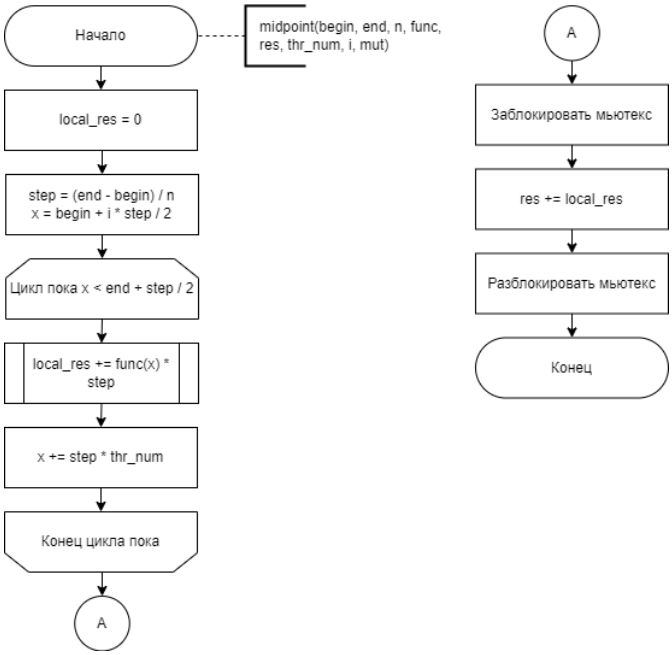

Рисунок 6 – Схема параллельного алгоритма численного интегрирования методом средних прямоугольников при заданных n и номере потока

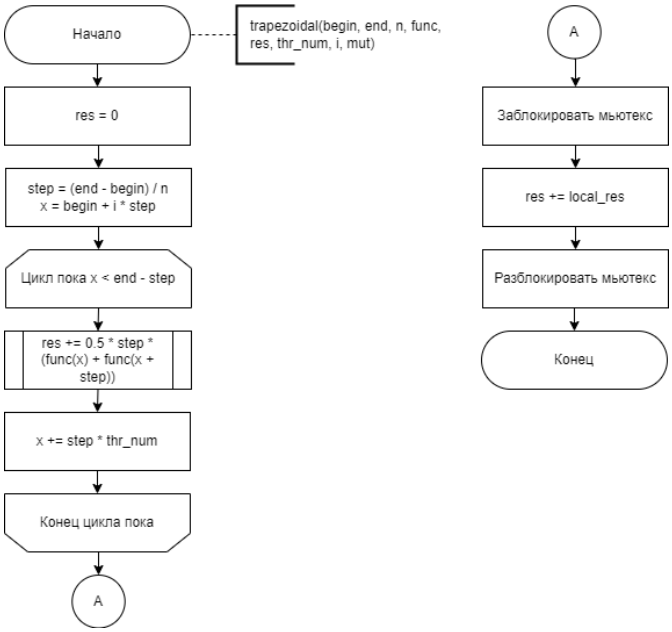

Рисунок 7 – Схема параллельного алгоритма численного интегрирования методом трапеций при заданных n и номере потока

Вывод

Были разработаны схемы последовательного и параллельного алгоритмов численного интегрирования.

3 Технологический раздел

В данном разделе описаны требования к программному обеспечению, средства реализации, приведены реализации алгоритмов и данные для тестирования.

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

В качестве языка программирования для реализации лабораторной работы был выбран язык C++ [2]. Данный язык предоставляет необходимые библиотеки для работы с потоками.

Для визуализации данных эксперимента был выбран язык программирования Python [3], так как он предоставляет большое число настроек параметров графика с использованием простого синтаксиса.

Для замера процессорного времени использовалась функция библиотеки chrono [4] `std::chrono::system_clock::now()`.

3.3 Используемые структуры данных

В программе используется структура interval_t, описанная в листинге 1. Ее полями являются начало интервала, конец интервала, точность вычислений.

Листинг 1 – Структура interval_t

```
1 struct interval_t
2 {
3     double begin;
4     double end;
5     double eps;
6 };
```

3.4 Реализации алгоритмов

В листингах 2-4 представлены реализации последовательных алгоритмов численного итегрирования методом средних прямоугольников и методом трапеций с заданной точностью.

В листингах 5-8 представлены реализации параллельных алгоритмов численного итегрирования методом средних прямоугольников и методом трапеций с заданной точностью.

Листинг 2 – Последовательный алгоритм численного интегрирования с заданной точностью

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

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

    return res;
}
```

Листинг 3 – Последовательный алгоритм численного интегрирования методом средних прямоугольников при заданном n

```
long double midpoint(double begin, double end, unsigned int n,
            function_t func)
{
    double res = 0;
    double step = (end - begin) / n;
    double x = begin + step / 2;

    while (x < end + step / 2)
    {
        res += func(x) * step;
        x += step;
    }
    return res;
}
```

Листинг 4 – Последовательный алгоритм численного интегрирования методом трапеций при заданном n

```
long double trapezoidal(double begin, double end, unsigned int n,
            function_t func)
{
    double res = 0;
    double step = (end - begin) / n;
    double x = begin;

    while (x < end - step)
    {
        res += 0.5 * step * (func(x) + func(x + step));
        x += step;
    }
    return res;
}
```

Листинг 5 – Параллельный алгоритм численного интегрирования с заданной точностью

```
long double calculate_integral_parallel(int threads_num,
    interval_t &interval,
            function_t func, void (&method)(double, double,
               unsigned int,
            function_t, long double &, int, int, mutex &))
{
    unsigned long n = 2;
    long double delta = 1;
    long double res = 0, obt_res = 0;

    while (delta > interval.eps)
    {
        res = obt_res;
        obt_res = multithreading(threads_num, interval, n, func,
                                 method);
        n *= 2;
        delta = abs(res - obt_res);
    }

    return res;
}
```

Листинг 6 – Создание потоков для алгоритмов численного интегрирования

```
long double multithreading(int threads_num, interval_t &interval,
            unsigned int n, function_t func, void (&method)
            (double, double, unsigned int, function_t,
            long double &, int, int, mutex &))
{
    long double res = 0;
    mutex res_mut;
    vector<thread> threads(threads_num);

    for (int i = 0; i < threads_num; i++)
    {
        threads[i] = thread(method, 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 7 – Параллельный алгоритм численного интегрирования методом средних прямоугольников при заданных n и номере потока

```
void midpoint_parallel(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 8 – Параллельный алгоритм численного интегрирования методом трапеций при заданных n и номере потока

```
void trapezoidal_parallel(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;

    while (x < end - step)
    {
        local_res += 0.5 * step * (func(x) + func(x + step));
        x += step * threads_num;
    }

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

3.5 Тестирование

В таблице 1 приведены функциональные тесты для алгоритмов интегрирования на функции $f(x) = x^2$. Тесты пройдены успешно.

Таблица 1 – Функциональные тесты

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

Вывод

Были описаны требования к программному обеспечению, средства реализации, приведены реализации алгоритмов и тесты, успешно пройденные программой.

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени:

— операционная система Windows 10;

— память 8 ГБ;

— процессор Intel® Core™ i5-6260U @ 1.80 ГГц, 2 физических ядра, 4 логических ядра.

Замеры времени выполнения реализаций алгоритмов проводились на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно разработанным приложением.

4.2 Демонстрация работы программы

На рисунке 8 представлен пример работы программы.

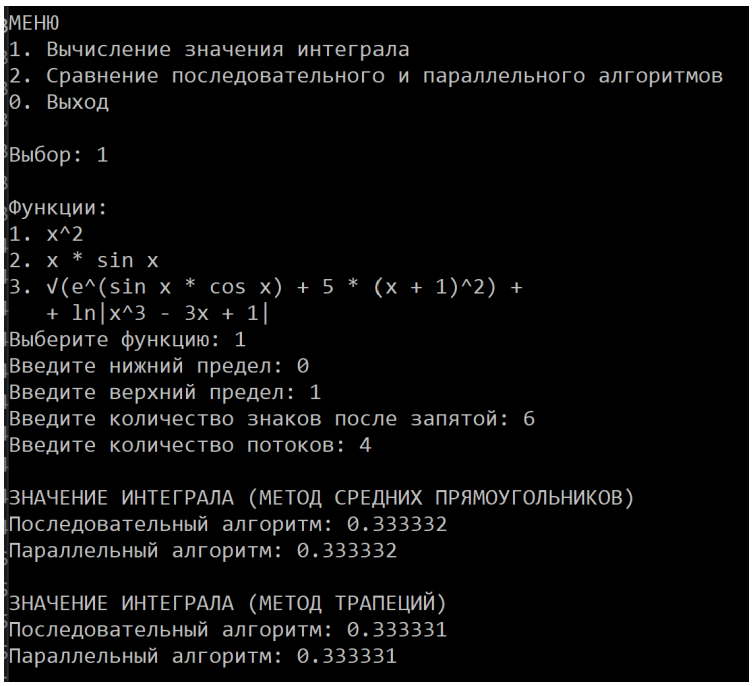

Рисунок 8 – Пример работы программы

4.3 Сравнение времени выполнения реализаций алгоритмов

Для оценки времени работы последовательной и параллельной реализации алгоритма численного интегрирования 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 32, 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.

В таблицах 2 и 4 представлены результаты измерения времени работы параллельных реализаций алгоритмов в зависимости от числа потоков. На рисунках 9 и 11 представлены соответствующие графики. Для сравнения в таблицах и на графиках приведено также время последовательных реализаций.

В таблицах 3 и 5 представлены результаты измерения времени работы последовательных и паралелльных реализаций алгоритмов в зависимости от числа знаков после запятой, соответствующие заданной точности вычислений интеграла. На рисунках 10 и 12 представлены соответствующие графики. Для наглядости на графике представлены только точки с точностью от 4 до 7 знаков после запятой.

Таблица 2 – Зависимость времени работы реализации от числа потоков (метод средних прямоугольников)

Число потоков	**Время, нс**
последовательный	956216734
1	949013547
2	456833477
4	404351737
8	410132845
16	428526431
32	502791420

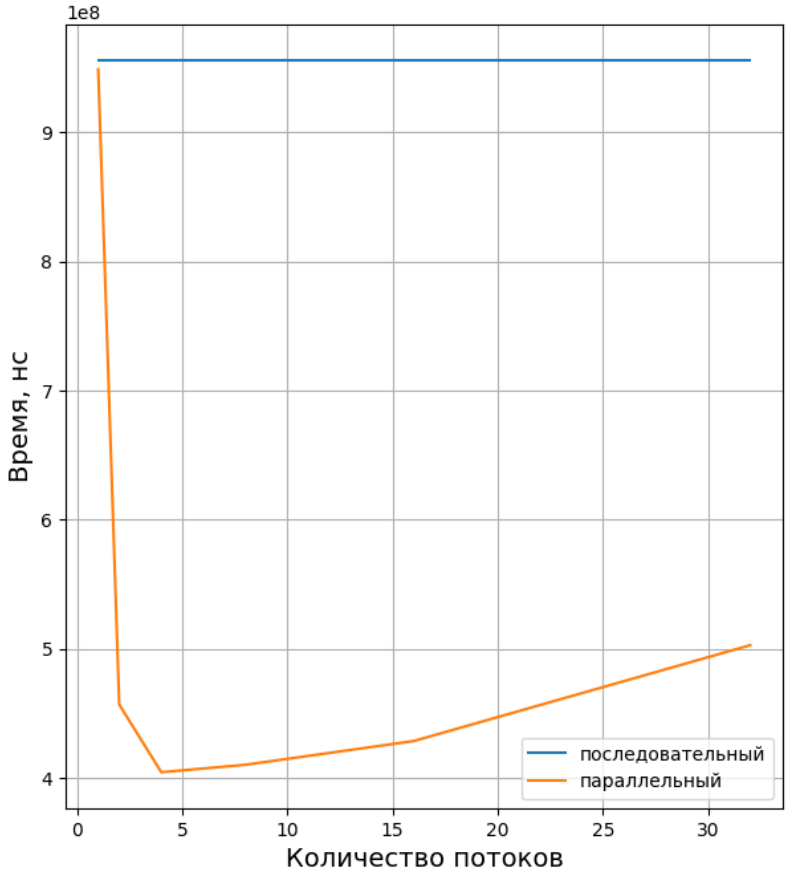

Рисунок 9 – Зависимость времени работы реализации от числа потоков (метод средних прямоугольников)

Таблица 3 – Зависимость времени работы реализации от точности (метод средних прямоугольников)

Точность	**4 потока, нс**	**Последовательный, нс**
1e-1	10301220	527000
1e-2	17464660	2857800
1e-3	30899890	15451550
1e-4	138586470	31622690
1e-5	554727780	1561291940
1e-6	1088159260	3061249760
1e-7	1855569650	4569952270

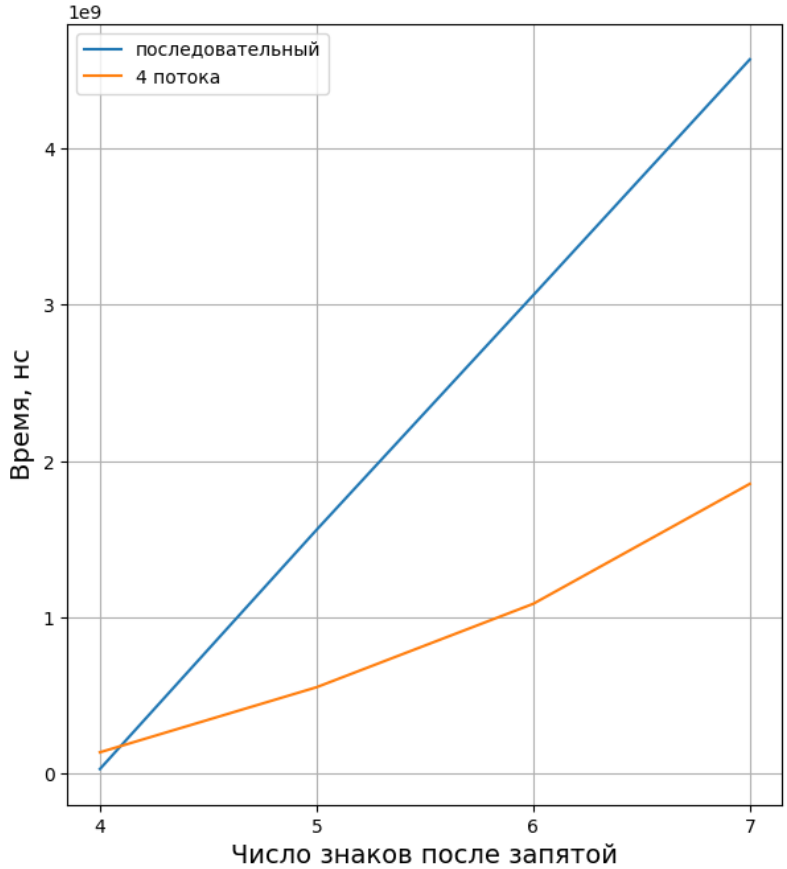

Рисунок 10 – Зависимость времени работы реализации от точности (метод средних прямоугольников)

Таблица 4 – Зависимость времени работы реализации от числа потоков (метод трапеций)

Число потоков	**Время, нс**
последовательный	1234645093
1	1224537253
2	592657522
4	404351737
8	422857284
16	465930467
32	622873546

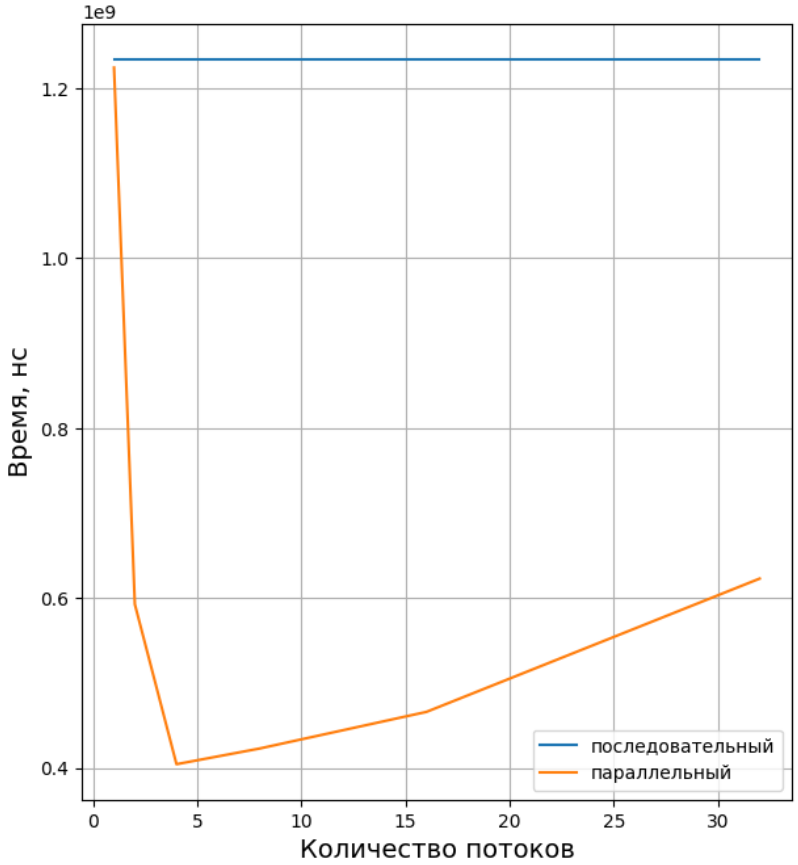

Рисунок 11 – Зависимость времени работы реализации от числа потоков (метод трапеций)

Таблица 5 – Зависимость времени работы реализации от точности (метод трапеций)

Точность	**4 потока, нс**	**Последовательный, нс**
1e-1	13298345	684562
1e-2	22701673	3708234
1e-3	39765463	20056836
1e-4	179563421	40367467
1e-5	721563426	2026647583
1e-6	1613763452	3973745637
1e-7	2412874563	5942874563

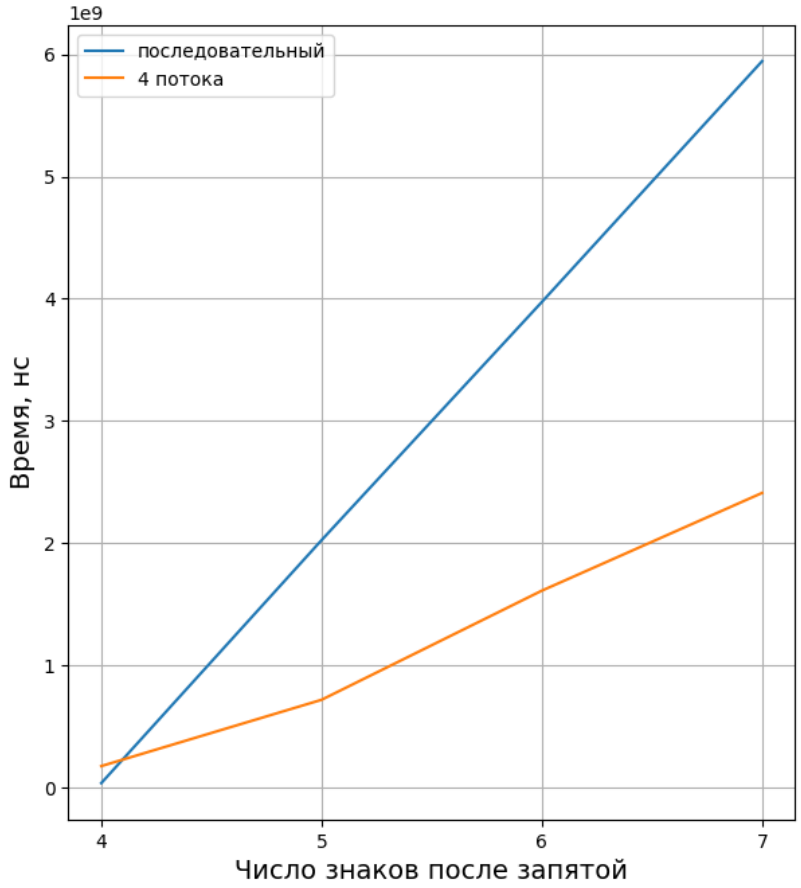

Рисунок 12 – Зависимость времени работы реализации от точности (метод трапеций)

Вывод

По результатам эксперимента можно сделать следующие выводы:

— по мере уменьшения числа потоков до числа логических ядер процессора время работы реализаций алгоритмов уменьшается;
— при превышении числа логических ядер время увеличивается, так как затрачивается время на переключение ядра между потоками;
— в зависимости от точности выигрыш от распараллеливания начинает проявляться только при точности от 5 знаков после запятой; это значит, что при 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 ядер процессора. При этом нецелесообразно использовать параллельную реализацию при проведении вычислений интегралов с точностью ниже, чем 5 знаков после запятой.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были решены следующие задачи:

- изучены методы средних прямоугольников и трапеций для численного интегрирования;
- описаны возможности распараллеливания данных алгоритмов;
- разработаны последовательный и параллельный алгоритмы;
- реализован каждый алгоритм;
- проведено тестирование реализованных алгоритмов;
- проведен сравнительный анализ алгоритмов по времени работы реализаций.

Поставленная цель была достигнута: были получены навыки параллельного программирования на основе алгоритмов численного интегрирования методом средних прямоугольников и методом трапеций.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. The Midpoint and Trapezoidal Rules [Электронный ресурс]. Режим доступа: https://courses.lumenlearning.com/calculus2/chapter/the-midpoint-and-trapezoidal-rules/ (дата обращения: 17.11.2022).

2. Документация по языку C++ [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170 (дата обращения: 17.11.2022).

3. Изучаем Python, том 1, 5-е изд / Лутц, Марк. — Пер. с англ. – СПб.: ООО “Диалектика”, 2019. — 832 с.

4. Date and time utilities [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono (дата обращения: 17.11.2022).