МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт Математики, физики и информационных технологий

(наименование института полностью)

Кафедра «Прикладная математика и информатика»

(наименование кафедры полностью)

01.03.02 Прикладная математика и информатика

(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование

(направленность (профиль))

**курсовая работа**

по дисциплине (учебному курсу)

**«математическое и компьютерное моделирование 1»**

(наименование дисциплины (учебного курса))

на тему «Итерационные алгоритмы численного интегрирования»

|  |  |  |
| --- | --- | --- |
| Студент | ПМИб-1902а, А.А. Горбатюк  (группа, И.О. Фамилия) | (личная подпись) |
| Руководитель | А.И. Сафронов  (И.О. Фамилия) | (личная подпись) |
|  |  |  |

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Тольятти, 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт Математики, физики и информационных технологий

(наименование института полностью)

Кафедра «Прикладная математика и информатика»

(наименование кафедры полностью)

УТВЕРЖДАЮ

и.о. Заведующего кафедрой

Гущина О.М

(подпись) (И.О. Фамилия)

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

**ЗАДАНИЕ**

**на выполнение курсовой работы**

Студент 3 курса гр. ПМИб-1902а Горбатюк Артём Андреевич \_

1. Тема Итерационные алгоритмы численного интегрирования \_
2. Срок сдачи студентом законченной курсовой работы
3. Исходные данные к курсовой работе язык программироваания С++, библиотеки для распараллеливания OpenMP (omp.h) и Threads (thread.h), среда разработки JetBrains CLion
4. Содержание курсовой работы (перечень подлежащих разработке вопросов, разделов)

Введение, Постановка задачи на исследование, Проектирование и разработка параллельной программы, Анализ эффективности параллельных алгоритмов и разработанного программного обеспечения, Заключение, Список используемой литературы, Приложение

1. Ориентировочный перечень графического и иллюстративного материала пояснительная\_ записка; блок-схема скриншоты программы, презентация
2. Рекомендуемые учебно-методические материалы

1. Антонов А.С. Параллельное программирование с использованием OpenMP: учебное пособие / А.С. Антонов. – М.: Изд-во МГУ, 2009. - 77 с.

2. Bjarne Stroustrup. The C++ Programming Language. Fourth Edition. Addison Wesley, 2013..

7. Дата выдачи задания « » 20 г.

|  |  |  |
| --- | --- | --- |
| Руководитель курсовой работы | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | А.И. Сафронов  (И.О. Фамилия) |
|  |  |  |
| Задание принял к исполнению | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | А.А. Горбатюк  (И.О. Фамилия) |

# Аннотация

Курсовая работа представлена на 30 страницах и включает в себя 33 рисунка, 5 таблиц, 22 формулы и список литературы, состоящий из 9 источников.

Курсовая работа посвящена реализации численных методов интегрирования на языке программирования С++.

Объектом исследования являются следующие методы численного интегрирования: методы левого, среднего и правого прямоугольника, метод трапеций и метод парабол (метод Симпсона).

Целью курсовой работы является повышение скорости нахождения определённого интеграла алгоритмами численного интегрирования при помощи распараллеливания вычислений на несколько потоков.

Курсовая работа состоит из введения, трёх глав и заключения.

В первой главе описывается постановка задачи на исследование и описание математической модели объекта исследования с реализацией последовательного алгоритма.

Во второй главе описывается процесс проектирования и разработки параллельной программы.

В третьей главе описывается анализ реализованных алгоритмов и разработанного программного обеспечения

# Содержание

[Введение 1](#_Toc39351574)

[1. Основные понятия 2](#_Toc39351575)

[1.1 Описание объекта исследования 2](#_Toc39351576)

[1.2 Метод прямоугольников 2](#_Toc39351577)

[1.3 Метод трапеций 3](#_Toc39351578)

[1.4 Метод парабол (метод Симпсона) 4](#_Toc39351579)

[2. Проектирование и разработка алгоритмов численного интегрирования 6](#_Toc39351580)

[2.1 Проектирование и реализация последовательной программы 6](#_Toc39351581)

[2.2 Обзор и выбор технологий разработки параллельного программного обеспечения 11](#_Toc39351582)

[2.3 Разработка параллельного алгоритма 15](#_Toc39351583)

[2.4 Реализация параллельной программы с использованием OpenMP (технология А) 16](#_Toc39351584)

[2.5 Реализация параллельной программы с использованием C++11 Threads (технология B) 19](#_Toc39351585)

[3 Анализ эффективности параллельных алгоритмов и разработанного программного обеспечения 22](#_Toc39351586)

[3.1 Теоретическое исследование эффективности параллельного алгоритма 22](#_Toc39351587)

[3.2 Разработка методики оценки эффективности программного обеспечения 23](#_Toc39351588)

[3.3 Произведение эксперимента на эффективность 24](#_Toc39351589)

[3.4 Анализ эффективности 28](#_Toc39351590)

[Заключение 29](#_Toc39351591)

[Список используемой литературы 30](#_Toc39351592)

# Введение

Интегралом называется площадь, заключённая между графиком функции *f(x)* на отрезке и осью абсцисс. Интеграл – это результат непрерывного суммирования бесконечно большого числа бесконечно малых слагаемых. Интеграл имеет большое значение в математической сфере деятельности: с его помощью можно найти такие физические величины как площадь, объём, энергию, работу, давление, массу, электрический заряд и многое другое.

Целью курсовой работы является повышение скорости нахождения определённого интеграла алгоритмами численного интегрирования при помощи распараллеливания процессов вычисления.

Задачами курсовой работы являются:

* анализ численных методов;
* выбор средств реализации;
* проектирование программы;
* написание программного кода.

Конечным результатом курсовой работы является программа, которая способна выполнять вычисление определённых интегралов итерационным способом последовательно и в несколько потоков, причём распараллеливание должно осуществляться двумя разными технологиями.

# 1. Основные понятия

## Описание объекта исследования

Под численным интегрированием понимают вычисление определённого интеграла при помощи одного из нескольких численных методов. Существует большое количество численных методов, вот некоторые из них:

* метод прямоугольников
* метод трапеций
* метод парабол (метод Симпсона)
* метод Гаусса
* метод Гаусса – Кронрода
* метод Чебышёва

В рамках курсовой работы будут рассмотрены и применены только три метода: метод прямоугольников, трапеций и Симпсона.

## 1.2 Метод прямоугольников

Данный метод численного интегрирования заключается в следующем: отрезок интегрирования [*a;b*] разбивается на *n* прямоугольников, где ширина прямоугольника (или шаг интегрирования) *h* вычисляется по формуле , а высота прямоугольника равна значению функции в данной точке. Затем суммируются площади полученных прямоугольников.

Если отрезок [*a, b*] является элементарным и не подлежит дальнейшему разбиению, то значение интеграла на этом отрезке можно найти по одной из трёх формул:

* 1. Формула **правых** прямоугольников:
  2. Формула **средних** прямоугольников:
  3. Формула **левых** прямоугольников:

Так как в данном методе весь отрезок интегрирования [*a,b*] разбивается на *n* **равных** отрезков, то выражение нужно заменить на шаг интегрирования *h*. Тогда значение интеграла от функции можно найти по формулам:

* 1. Формула **правых** прямоугольников:
  2. Формула **средних** прямоугольников:
  3. Формула **левых** прямоугольников:

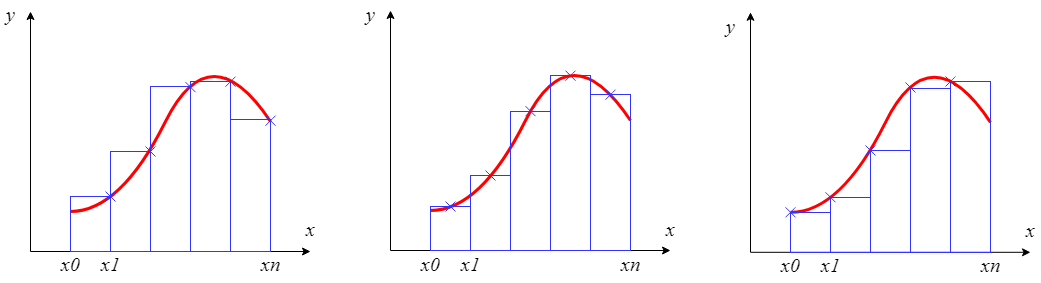
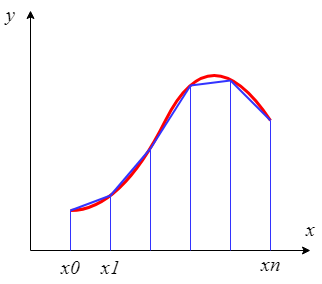
 Графическое представление метода прямоугольников показано на рисунке 1.1. На данном рисунке показано вычисление интеграла по формуле правых, средних и левых прямоугольников.

Рисунок 1.1 Метод прямоугольников.

## 1.3 Метод трапеций

Данный метод очень схож с методом прямоугольников. Здесь также отрезок интегрирования [*a,b*] разбивается на *n* интервалов, а на каждом интервале строится трапеция. Следовательно, для вычисления значения интеграла необходимо найти площадь трапеции, а не прямоугольника. Графическое представление метода трапеций изображено на рисунке 1.2.

Рисунок 1.2 Метод трапеций

Площадь трапеции находится по формуле , где *a* и *b* – основания трапеции, *h* – высота трапеции. На рисунке 1.2 в качестве оснований трапеций выступают , а высотой трапеции является шаг интегрирования . В таком случае конечная формула будет выглядеть следующим образом:

Стоит отметить, что метод трапеций может быть получен путём нахождения среднего арифметического между результатами применения формул метода правых и левых прямоугольников.

## 1.4 Метод парабол (метод Симпсона)

Суть данного метода заключается в том, чтобы разбить отрезок интегрирования на множество интервалов [], *i =* 1,2,3…,*n* и найти значение квадратичной параболы на каждом интервале, то есть значение функции под интегралом приближается функцией , которая проходит через точки , , *.* Расстояние между двумя ближайшими точками *h* вычисляется по формуле . Подмена исходной функции квадратичной параболой делается для того, чтобы была возможность вычислить интеграл по формуле **Ньютона-Лейбница**:

Также здесь применяется свойство интегралов:

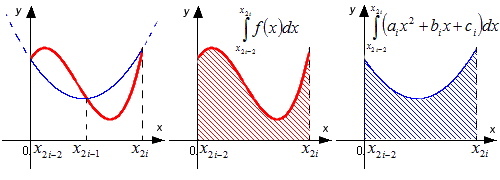
Графическое представление метода Симпсона показано на рисунке 1.3.

Рисунок 1.3 Метод Симпсона

Для того чтобы найти значение интеграла, необходимо вычислить значение функции в точках *a* и *b*, сумму значений функций в нечётных точках *S1*, сумму значений функций в чётных точках *S2*. Тогда конечные формулы примут следующий вид:

# 2. Проектирование и разработка алгоритмов численного интегрирования

## Проектирование и реализация последовательной программы

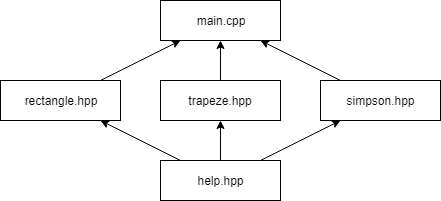
 Структура программы представлена на рисунке 1.4. Старт программы начинается из файла *main.cpp*. Пользователю выводится сообщение о том, какая подынтегральная функция будет вычисляться при помощи методов, затем пользователь должен ввести нижнюю и верхнюю границы интегрирования, а также количество интервалов, на которые будет делиться введённый отрезок интегрирования. На данном этапе программа будет иметь вид, представленный на рисунке 1.5.

Рисунок 1.4 Структура последовательной программы

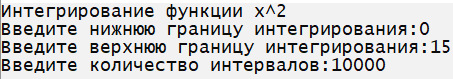


Рисунок 1.5 Ввод данных для интегрирования

Здесь также осуществляется проверка на правильность введённых данных: нижняя граница отрезка должна быть меньше или равна верхней, а количество интервалов должно быть больше 0. Листинг проверки представлен на рисунке 1.6. Сообщения о некорректности данных будут выводиться пользователю до тех пор, пока не будут получены правильные данные.

Далее идёт вычисление интеграла методом правых, средних, левых прямоугольников, трапеций и Симпсона. Они описаны в соответствующих заголовочных файлах: метод прямоугольников – *rectangle.hpp,* метод трапеций – *trapeze.hpp*, метод Симпсона – *simpson.hpp.* Подынтегральная функция задаётся в файле *help.hpp.* Вынесение данной функции в отдельный файл обусловлено удобством изменения самой интегрируемой функции всего лишь в одном месте таким образом, чтобы изменялись конечные результаты каждого из численного метода, также это позволяет избавиться от дублирования кода.

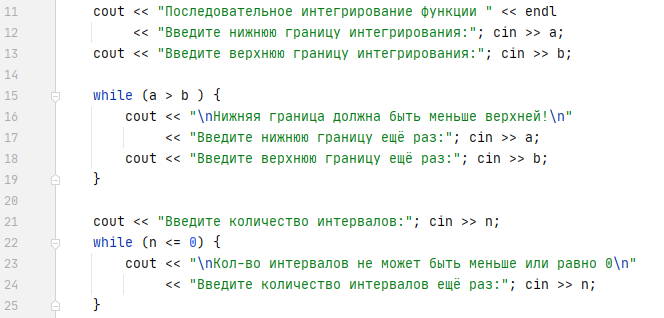


Рисунок 1.6 Листинг проверки введённых значений

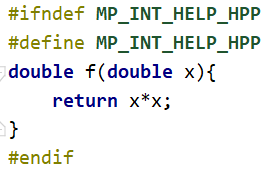
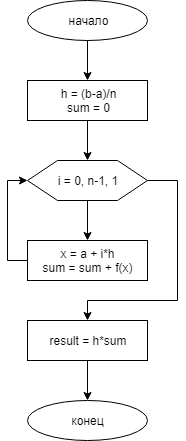
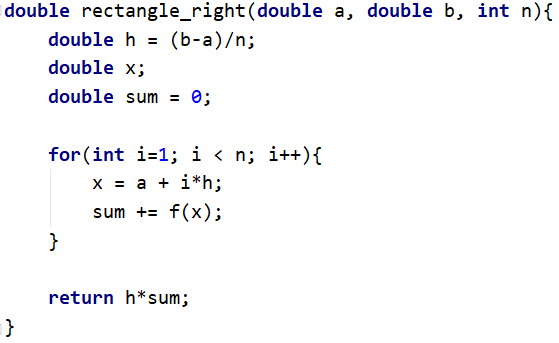


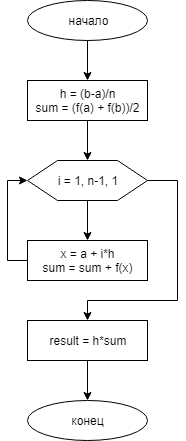
Рисунок 1.7 Содержимое файла *help.hpp*

На рисунках 1.8, 1.10, 1.12 представлены блок-схемы алгоритмов методов численного интегрирования.

Рисунок 1.8 Алгоритм метода левых прямоугольников

Рисунок 1.9 Листинг метода правых прямоугольников

Метод правых прямоугольников отличается от метода левых тем, что идёт смещение *xi* на единицу вправо, то есть *i =* 1,2,…,n, а в методе средних прямоугольников берётся .

Рисунок 1.10 Алгоритм метода трапеций

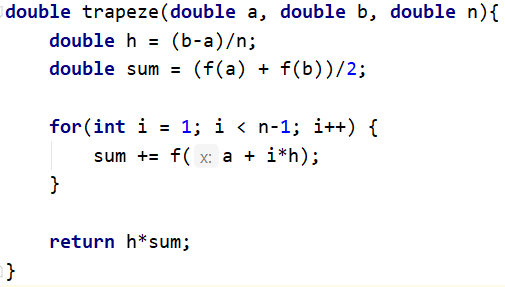
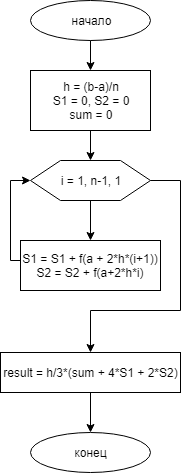


Рисунок 1.11 Листинг метода трапеций

Метод трапеций учитывает крайнюю левую и крайнюю правую границы отрезка интегрирования, поэтому обе точки повторно не рассматриваются, а счётчик цикла идёт от 1 до *n-*1.

Рисунок 1.12 Алгоритм метода Симпсона

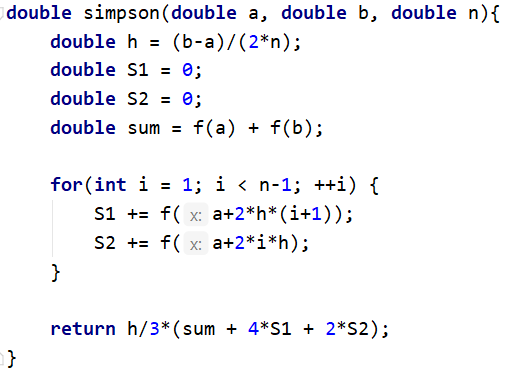


Рисунок 1.13 Листинга метода Симпсона

Так как метод Симпсона также учитывает значение функции на крайних точках, то и цикл будет проходить по точкам *xi*, *i* = 1,2,…,*n*-1. Стоит заметить, что за одну итерацию цикла одновременно вычисляется значение функции в чётных и нечётных точках.

Для того чтобы проверить работоспособность численных методов, было вычислено теоретическое значение интеграла по формуле Ньютона-Лейбница.

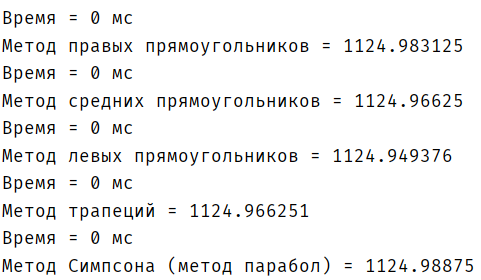
 Результат программы при *n* = 100 000 показан на рисунке 1.14.

Рисунок 1.14 Результат работы программы

Среди вычисленных значений интеграла методами прямоугольников, трапеций и Симпсона наиболее точным оказался метод Симпсона, это обусловлено тем, что данный метод находит значение интеграла путём приближения параболой, а график функции *x*2 является параболой.

## 2.2 Обзор и выбор технологий разработки параллельного программного обеспечения

**OpenMP (Open Multi-Processing)** является стандартом для многопроцессорных систем с общей памятью библиотек для таких языков программирования как Fortran и C/C++. OpenMP является надстройкой над Pthreads – POSIX-интерфейсом для организации потоков. Этот стандарт реализует выполнение программ с помощью многопоточных вычислений: При выполнении кода вне параллельных областей программа выполняется главным (master) потоком. Затем алгоритм входит в параллельную область, главный поток разделяется на подчинённые (slave) потоки. При выходе из параллельной области подчинённые потоки уничтожаются.

К преимуществам данной технологии можно отнести возможность создания программ по принципу «написал последовательную программу и распараллелил». В OpenMP нет нужды поддерживать последовательный и параллельный варианты программ. Это обеспечивается благодаря тому, что технология реализуется через директивы компилятора вида *#pragma omp \*directive-name\* option1 option2*, то есть обычный компилятор просто проигнорирует директивы и скомпилирует последовательный вариант. Одним из достоинств OpenMP его разработчики считают поддержку так называемых orphan(оторванных) директив, то есть директивы синхронизации и распределения работы могут не входить непосредственно в лексический контекст параллельной области.

Недостатками стандартами является узкая специализация на мультипроцессорные системы и системы с распределённой памятью, на которых смоделирована общая память, организация взаимодействия потоков через общие переменные, а не передачу сообщений между потоками.

**MPI (Messages Passing Interface)** – это стандарт API для параллельных вычислений, который отлично подходит для систем с распределённой памятью. Такие системы называются кластерами – набор соединённых сетью компьютеров. В кластерах повышение производительности достигается за счёт увеличения количества вычислительных устройств.

Программа с MPI сразу представляет из себя набор нескольких процессов, в отличии от OpenMP, где она может в один момент выполняться одним потоком, а в другой уже несколькими. Количество процессоров, на которых будет выполняться программа, передаётся в качестве параметра при запуске. Библиотека MPI предоставляет инструменты для синхронизации и обмена данных.

Технология реализуется через процедуры, названия которых начинается с MPI\_. В начале параллельного алгоритма обязательно надо выполнить *MPI\_Init(int\* argc, char \*\*\*argv)*, это позволит использовать остальные MPI-процедуры. Если параллельная часть алгоритма закончилась, то необходимо выполнить *MPI\_Finalize(void)* для запрета обращения к MPI-процедурам.

К преимуществам технологии относят высокую масштабируемость и переносимость кода между разными компьютерными системами. Недостатками является большая сложность программирования с применением технологии MPI из-за того, что MPI является низкоуровневым инструментом, зависимость скорости вычислений от характеристик сети, в которой подключены вычисляющие устройства, а также наличие специального программного обеспечения при выполнении программ на одном устройстве.

**C++11 Threads.** С опубликованием 11 стандарта языка С++ появился новый объект *std::thread*. Это означает, что добавлена поддержка многопоточности. Потоки обеспечивают более тонкое управление, чем процессы, но влекут за собой более существенные издержки, которые возникают из-за особенностей механизмов хранения раздельных копий переменных и передачи исключений между потоками. Однако для успешной компиляции программы с использованием *std::thread* пользователям операционной системы Windows необходимо наличие компилятора MinGW, собранного специально для работы с POSIX-потоками. В противном случае компилятор будет игнорировать подключение библиотеки с потоками (рисунок 2.1).



Рисунок 2.1 Ошибка при компиляции программы

Преимущества поточной технологии заключается в следующем:

1. Затраты ресурсов на создание потоков меньше, чем на создание процессов;
2. Переключение между потоками для ОС менее накладно, чем переключение между процессами;

К недостаткам можно отнести то, что если программист забудет применить *join()* или *detach()* для потоков, то работа программы будет завершена. Если применить *join()*, то алгоритм будет дожидаться завершения работы потока, а *detach()* разорвёт связь с главным потоком, тогда программа завершит свою работу, а поток продолжит действовать автономно.

Для реализации параллельного алгоритма было принято решение использовать две технологии: OpenMP и C++11 Threads. OpenMP был выбран из-за простоты реализации параллельного алгоритма путём добавления директивы перед циклом for в последовательный вариант программы, а также из-за наличия опыта работы с данной технологией. Технология C++11 Threads была выбрана из-за простоты в освоении, из-за отсутствия требования постороннего программного обеспечения как в MPI.

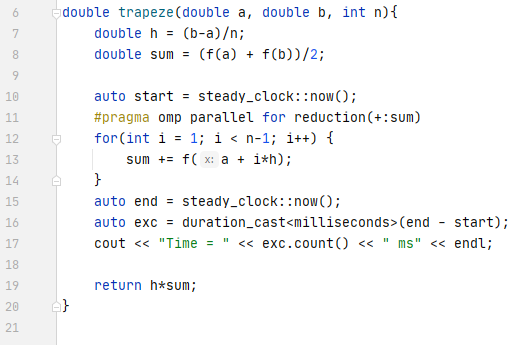
## 2.3 Разработка параллельного алгоритма

Алгоритм численного интегрирования отлично подходит для распараллеливания на системе с общей памятью. В данном алгоритме нет зависимостей по данным, нет зависимостей между итерациями. Он имеет только единственный фрагмент для распараллеливания – цикл for. Вычисления будут производиться на 4 потоках.

Здесь есть слабое место – разделяемая переменная для накопления результата вычислений. Это можно решить при помощи использования блокировок (omp\_lock\_t для OpenMP и mutex для C++11 Parallel Technologies). Однако это увеличит время выполнения параллельного фрагмента кода и, вероятнее всего, это приведёт к потере производительности, так как одновременно вычислять может только один поток. Поэтому следует применить другой способ: использовать приватных переменный для каждого потока, а после вычислений просуммировать переменные.

В OpenMP для таких случаев имеется директива *reduction*, эта директива указывает, чтобы в каждом потоке создались свои локальные переменные, выполнили вычисления, а по завершению работы будут сложены значения в единственную переменную. В C++11 Parallel Technologies такого механизма нет, поэтому необходимо сначала объявить отдельные переменные для каждого потока, а затем вручную сложить значения.

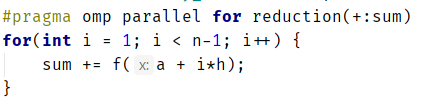
## Реализация параллельной программы с использованием OpenMP (технология А)



В последовательном варианте программы у каждого метода численного интегрирования имеется цикл for (рисунок 2.2), который и необходимо распараллелить.

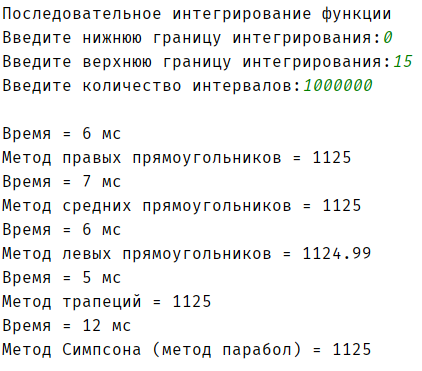
Рисунок 2.2 Вариант последовательного вычисления метода трапеций.

Средства OpenMP позволяют сделать распараллеливание очень просто: достаточно добавить директиву *#pragma omp parallel for*. Однако при таком варианте будет «гонка данных» (data race) – ситуация, когда несколько процессов обращаются к одному ресурсу одновременно. При выходе из цикла значение будет неправильным, поэтому надо воспользоваться директивой *reduction(+:sum).* В OpenMP при применении редукции создание локальных переменных для каждого потока скрыто от программиста. Теперь гонка данных будет исключена и будет вычисленное корректное значение. Конечный вид функции, вычисляющей методом трапеций представлен на рисунке 2.3. Остальные методы интегрирования выполнены аналогичным образом.

Рисунок 2.3 Фрагмент параллельного вычисления метода трапеций с использованием OpenMP.

Алгоритм параллельного вычисления метода трапеций изображён на рисунке 2.4.

Рисунок 2.4 Алгоритм параллельного вычисления интеграла методом трапеций.

 Протестируем программу и сравним точность вычисленных значений в сравнении с последовательной программой. Для сравнения будет использована функция , значение интеграла от которой на промежутке [0;15] было вычислено в первой главе – оно равно 1125.

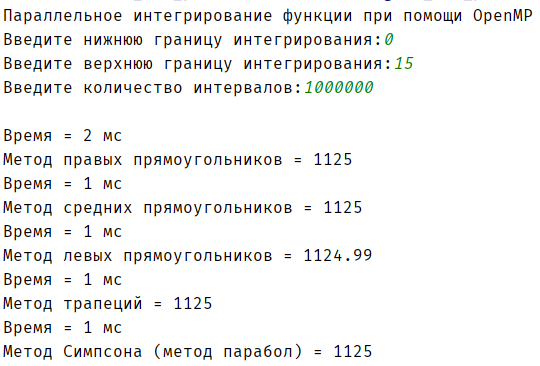
Рисунок 2.5 Результат последовательных вычислений

Рисунок 2.6 Результат параллельных вычислений (OpenMP)

Тестирование показало, что точность вычислений у параллельного и последовательного вариантов программы совпадают между собой, а также совпадают с теоретически рассчитанным значением. Выигрыш во времени у параллельного вычисления намного выше, особенно у метода Симпсона.

## 2.5 Реализация параллельной программы с использованием C++11 Threads (технология B)

Алгоритм программы совпадает с тем алгоритмом, который был применен для OpenMP (рисунок 2.4). Однако в C++11 Threads для решения проблемы «гонки данных» нет встроенных средств. Поэтому принято решение выделить отдельные переменные для каждого потока. Так как используется всего 4 потока, то объявление 4 дополнительных переменных не займёт много ресурсов компьютера.

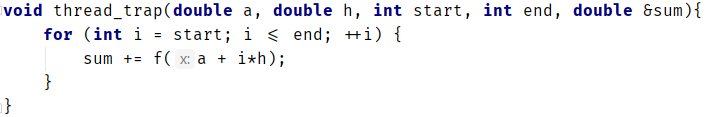
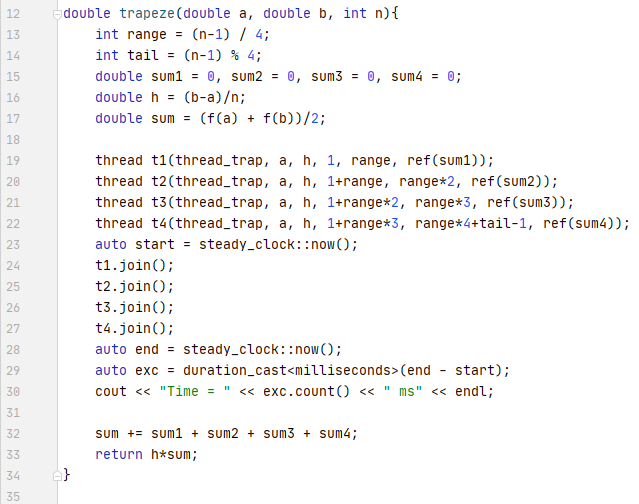
 С++11 Threads требует при объявлении потока указывать функцию, которую он будет выполнять, поэтому для каждого метода была написана собственная функция. Пример реализации метода трапеций при помощи C++11 Threads показан на рисунках 2.7 и 2.8.

Рисунок 2.7 Функция, которую будет выполнять поток

Рисунок 2.8 Часть параллельной программы на C++11 Threads

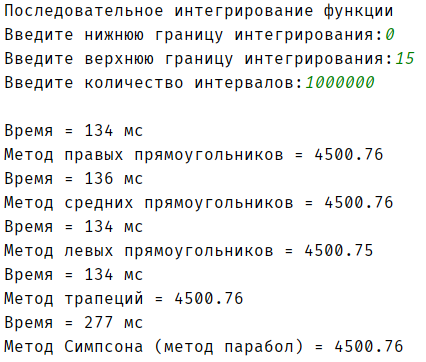
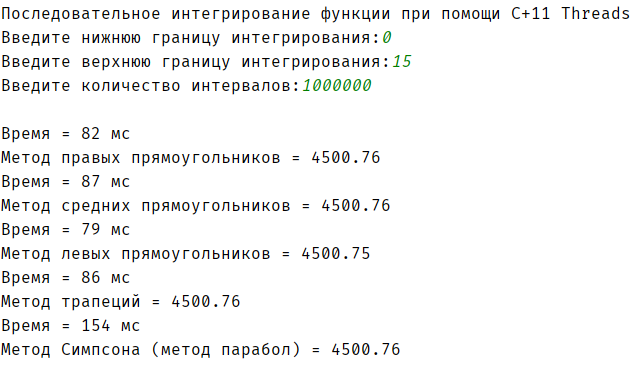
 Для тестирования программы изменим функцию на более сложную: заменим на и сравним время и значения, полученные в ходе вычисления последовательной программы и программы с применением C++11 Threads.

Рисунок 2.9 Результат последовательных вычислений

Рисунок 2.10 Результат параллельных вычислений (C++11 Threads)

Результат тестирования показывает, что точность вычислений совпадает, а это означает отсутствие гонок данных. Также видно, что скорость вычислений возросла в 1,8 раз.

# Анализ эффективности параллельных алгоритмов и разработанного программного обеспечения

## Теоретическое исследование эффективности параллельного алгоритма

Методы численного интегрирования имеют линейную сложность О(n). это означает, что при увеличении количества интервалов n в X раз увеличивается и время выполнения функции в Х раз. Ниже будут приведены формулы, в которых за n принимается количество отрезков интегрирования, а p – количество потоков.

Количество операций на реализацию последовательного алгоритма будет иметь зависимость:

,

Тогда время, затраченное на реализацию последовательного алгоритма будет равно:

Количество операций, затраченных на выполнение параллельного алгоритма будет равно:

Время, затраченное на выполнение параллельного алгоритма равно:

Коэффициент ускорения параллельных вычислений находят по формуле:

А коэффициент эффективности распараллеливания по формуле:

Теоретические исследования показывают, что коэффициент ускорения в лучшем случае будет равен , так как используется 4 потока на 4-ядерном процессоре, а коэффициент эффективности будет равен .

## Разработка методики оценки эффективности программного обеспечения

Для оценки разработанных программ будет использоваться коэффициент ускорения и эффективности. Формулы расчёта данных характеристик были описаны в предыдущем параграфе. Для измерения времени было задействовано два способа: профайлер *gprof* и стандартная библиотека *chrono*. Профайлер *gprof* входит в пакет установки компилятора MinGW. Чтобы им воспользоваться, необходимо собрать программу, передав компилятору параметр *–pg*. Тогда команда для сборки будет выглядеть следующим образом:

gcc –g –pg rectangle.hpp trapeze.hpp simpson.hpp main.cpp –o MP\_int.exe

Затем необходимо запустить программу, после этого в каталоге появится файл *gmon.out*. Данный файл содержит данные, собранные профайлером. Чтобы его прочитать, надо ввести команду

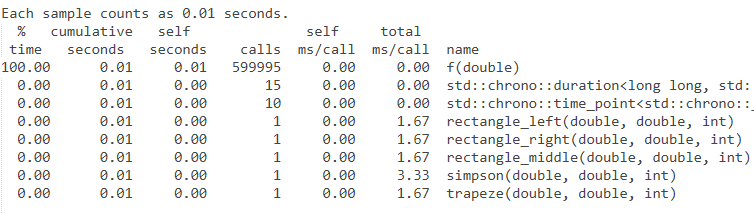
gprof MP\_int.exe gmon.out

Рисунок 3.1 Результат работы профайлера gprof

## Произведение эксперимента на эффективность

Эксперимент проводился на компьютере с 4-х ядерным процессором i5-8300H и 8 ГБ оперативной памяти. Для каждого метода интегрирования были замерены следующие характеристики: время выполнения последовательного и двух параллельных вычислений, ускорение и эффективность для каждой технологии распараллеливания. На таблицах 1-5 представлены результаты эксперимента.

Таблица 1

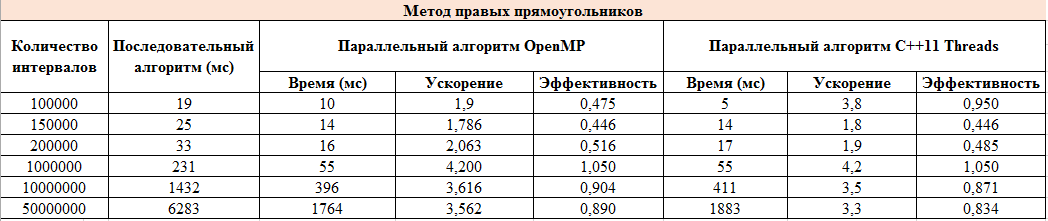
Результат эксперимента для метода правых прямоугольников

Таблица 2

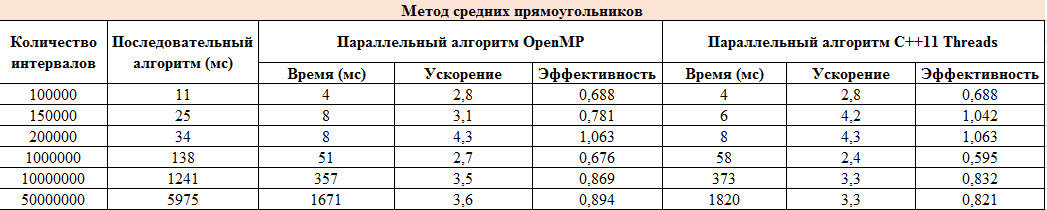
Результат эксперимента для метода средних прямоугольников

Таблица 3

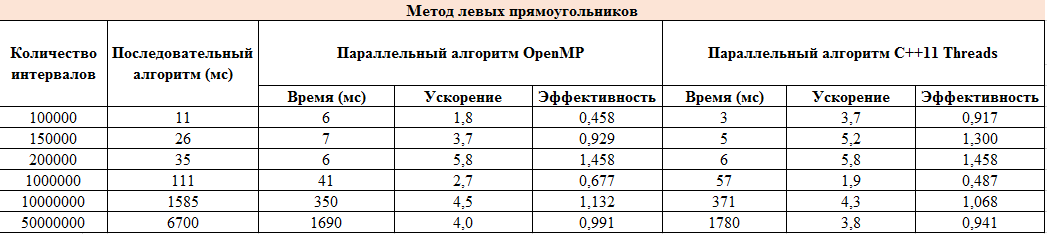
Результат эксперимента для метода левых прямоугольников

Таблица 4

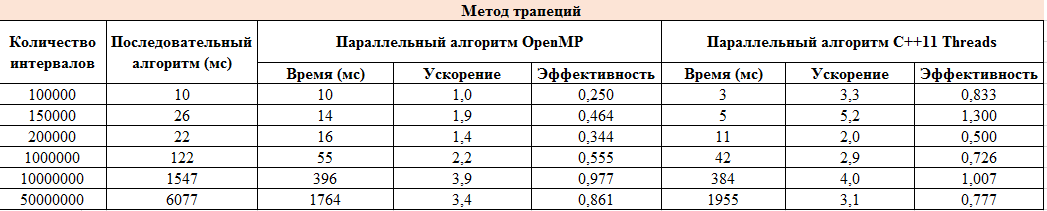
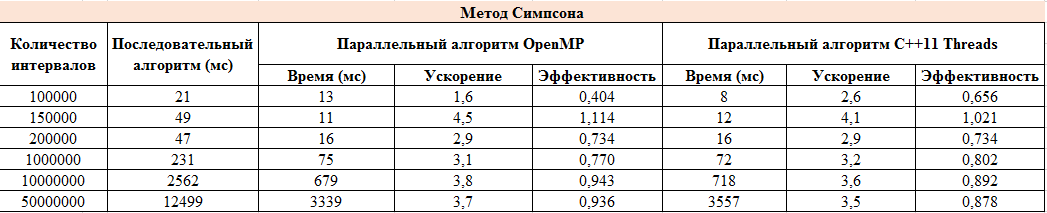
Результат эксперимента для метода трапеций

Таблица 5

Результат эксперимента для метода Симпсона

На рисунках 3.2-3.4 показаны графики зависимостей времени от количества интервалов для последовательной программы и двух параллельных.

Рисунок 3.2 Зависимость времени от количества интервалов для последовательной программы

Рисунок 3.3 График зависимости времени от количества интервалов для программы с OpenMP

Рисунок 3.4 График зависимости времени от количества интервалов для программы с C++11 Threads

Измерения проводились для количества интервалов в диапазоне от 100 000 до 50 000 000. На график видно, что использование OpenMP и C++11 Threads значительно сокращает время вычисления значения определённого интеграла.

На рисунках 3.5-3.9 представлены графики ускорения для каждого метода численного интегрирования.

Рисунок 3.5 Показатели ускорения для метода правых прямоугольников

Рисунок 3.6 Показатели ускорения для метода средних прямоугольников

Рисунок 3.7 Показатели ускорения для метода левых прямоугольников

Рисунок 3.8 Показатели ускорения для метода трапеций

Рисунок 3.9 Показатели ускорения для метода Симпсона

## Анализ эффективности

По данным графикам видно, что при количестве интервалов меньше 200 000 быстрее выполняет вычисления программа с использованием технологии C++11 Threads, но потом начинает проигрывать на большем количестве интервалов. Это обусловлено тем, что у OpenMP уходит больше времени на открытие потоков, однако скорость обмена данными и взаимодействия между потоками незначительно быстрее, чем у C++11 Threads.

# Заключение

Целью курсовой работы являлось повышение скорости нахождения определённого интеграла при помощи распределения вычислений между несколькими потоками. Эта цель была достигнута при помощи двух технологий: OpenMP и C++11 Threads.

В результате выполнения курсовой работы было создано три программы: первая программа вычисляет интеграл последовательно, вторая программа – при помощи OpenMP, третья программа – при помощи C++11 Threads. Каждая из программ вычисляет значение интеграла пятью методами численного интегрирования: методы правых, средних, левых прямоугольников, метод трапеций и метод Симпсона.

Также было проведено тестирование с целью анализа эффективности разработанных программ. Результаты показывают, что для вычислений определённого интеграла с количеством интервалов до 200 000 выгоднее использовать программу с C++11 Threads. Если количество интервалов находится в промежутке от 200 000 до 1 000 000, то обе программы имеют одинаковую производительность. Если же количество превышает 1 000 000, то с небольшим преимуществом вырывается вперёд программа с OpenMP.

# Список используемой литературы

* 1. Антонов А.С. Параллельное программирование с использованием OpenMP: учебное пособие / А.С. Антонов. – М.: Изд-во МГУ, 2009. - 77 с.
  2. А.А. Амосов, Ю.А. Дубинский, Н.В. Копченова. Вычислительные методы для инженеров. М.: Высш. шк. 1994
  3. Bjarne Stroustrup. The C++ Programming Language. Fourth Edition. Addison Wesley, 2013
  4. Метод Симпсона (парабол), формула для оценки погрешности, примеры, решения. [Электронный ресурс]. Режим доступа: http://www.cleverstudents.ru/integral/method\_of\_parabolas.html
  5. Эхтер Ш., Робертс Дж. Многоядерное программирование. — СПб.: Питер, 2010. — 316 с: ил. — (Серия «Библиотека программиста»).
  6. Введение в технологию OpenMP (применение с использованием языка С++). [Электронный ресурс]. Режим доступа: http://www.quizful.net/post/cpp-openmp
  7. Параллельная обработка данных – лекция 5. Message Passing Interface. [Электронный ресурс]. Режим доступа: https://parallel.ru/vvv/mpi.html
  8. C++ Multithreading – Threading in C++. [Электронный ресурс]. Режим доступа: https://www.tutorialcup.com/cplusplus/multithreading.htm#How-to-create-a-thread
  9. Оценка сложности алгоритмов / Хабр. [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/104219/

**Приложение**

Исходный код разработанной программы размещён на платформе GitHub и доступен по ссылке <https://github.com/Spec163/mkm_course_work>