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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

Название: Параллельное программирование

Дисциплина: Анализ алгоритмов

Студент	ИУ7-54Б		С. Д. Параскун
	Группа	Подпись, дата	И. О. Фамилия
Преподаватель			Л. Л. Волкова
		Подпись, дата	И. О. Фамилия

Москва, 2021 г.

Содержание

Введение

В настоящее время компьютерные системы оперируют большими объемами данных. Над этими данными проводится большой объем различного рода вычислений. Для того, чтобы они выполнянлись быстрее, было придумано параллельное программирование.

Его суть заключается в том, чтобы относительно равномерно разделять нагрузку между потоками ядра. Каждое из ядер процессора может обрабатывать по одному потоку, поэтому когда количество потоков на ядро становится больше, происходит квантование времени. Это означает, что на каждый процесс выделяется фиксированная величина времени (квант), после чего в течение кванта обрабатывается следующий процесс. Таким образом создается видимость параллельности. Тем не менее, данная оптимизация может сильно ускорить вычисления.

Целью данной лабораторной работы является получение навыков параллельного программирования. Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить последовательный и параллельный варианты выбранного алгоритма;
- составить схемы данных алгоритмов;
- реализовать разработанные версии алгоритма;
- провести сравнительный анализ реализаций по затрачиваемым ресурсам (время и память);
- описать и обосновать полученные результаты.

1. Аналитический раздел

В данном разделе будут представлены описания последовательного и параллельного вариантов алгоритма вычисления определенного интеграла методом трапеций.

1.1 Вычисление определенного интеграла методом трапеций

Определенный интеграл - одно из основных математических понятий, применимое при вычислении площади под графиком заданной функции. Для его вычисления необходимо разбить отрезок, по которому будет интегрироваться функция, на множество маленьких отрезков и просуммировать значения функции. Это общий алгоритм вычисления определенного интеграла.

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

Чтобы повысить точность вычислений, можно использовать метод трапеций [1]. Его суть заключается в том, что на всех интервалах кроме крайних левого и правого вычисляется значение функции, суммируется, после прибавляется усредненное значение неучтенных интервалов и домножается на шаг интервалов.

Представим это в виде математической формулы. Если a и b - левая и правая границы интервалов, n - количество интервалов, $h = (b - a)/n$ - шаг аргумента функции, x_i - значение аргумента на шаге i, $f(x_i)$ - значение функции на шаге i. Тогда определенный интеграл может быть вычислен по следующей формуле:

$$I = h \cdot (\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)) \tag{1.2}$$

1.2 Параллельное вычисление определенного интеграла

Так как в алгоритме происходит разбиение исходного отрезка на несколько, и вычисления на них могут происходить независимо, целесообразно произвести распараллеливание по отрезкам интегрирования. Причем количество данных отрезков будем определять как количество создаваемых потоков. На каждом отрезке поток [2] будет вычислять значение определенного интеграла, и после того, как все потоки завершаться, будет произведено суммирование.

1.3 Вывод

В данном разделе были рассмотренны принципы работы последовательного и параллельного алгоритмов вычисления определенного варианта. Полученных знаний достаточно для разработки выбранных алгоритмов. На вход алгоритмам будут подаваться начало и конец интервала интегрирования, а также количество интервалов разбиения и исходная фукция. Реализуемое ПО будет работать в пользовательском режиме (вывод отсортированного массива тремя методами), а также в экспериментальном (проведение замеров времени выполнения алгоритмов).

2. Конструкторский раздел

В данном разделе будут спроектированы схемы алгоритмов, описаны используемые типы данных, а также произведена оценка памяти и описана структура ПО.

2.1 Схемы алгоритмов

На рисунках 2.1 - 2.3 представлены схемы рассматриваемых алгоритмов.

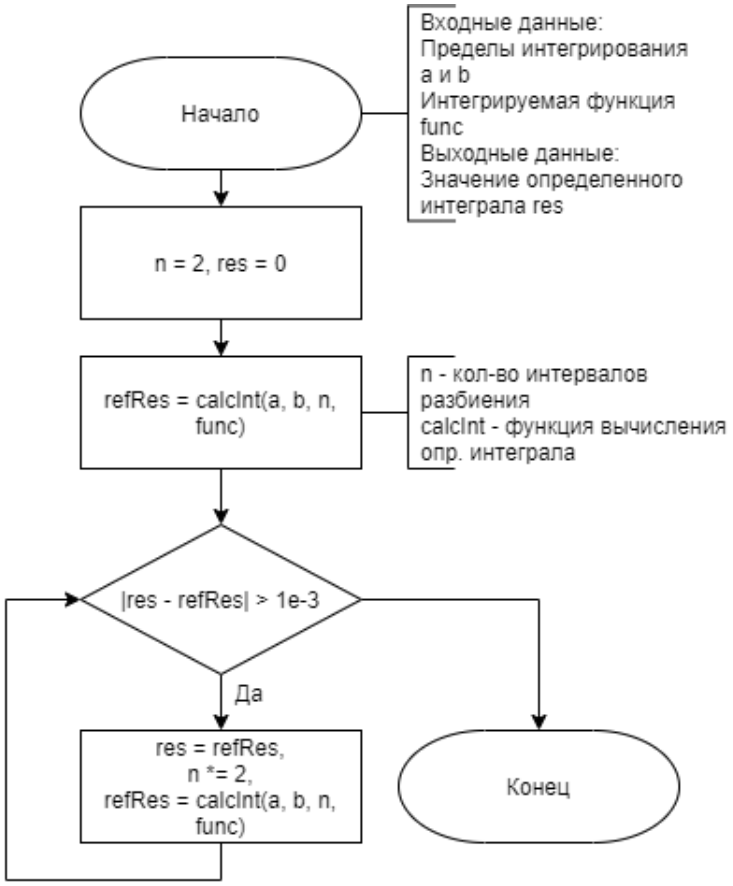

Рисунок 2.1 – Схема алгоритма последовательного вычисления значения определенного интеграла

В данном алгоритме используется функция вычисления определенного интеграла (рисунок 2.2).

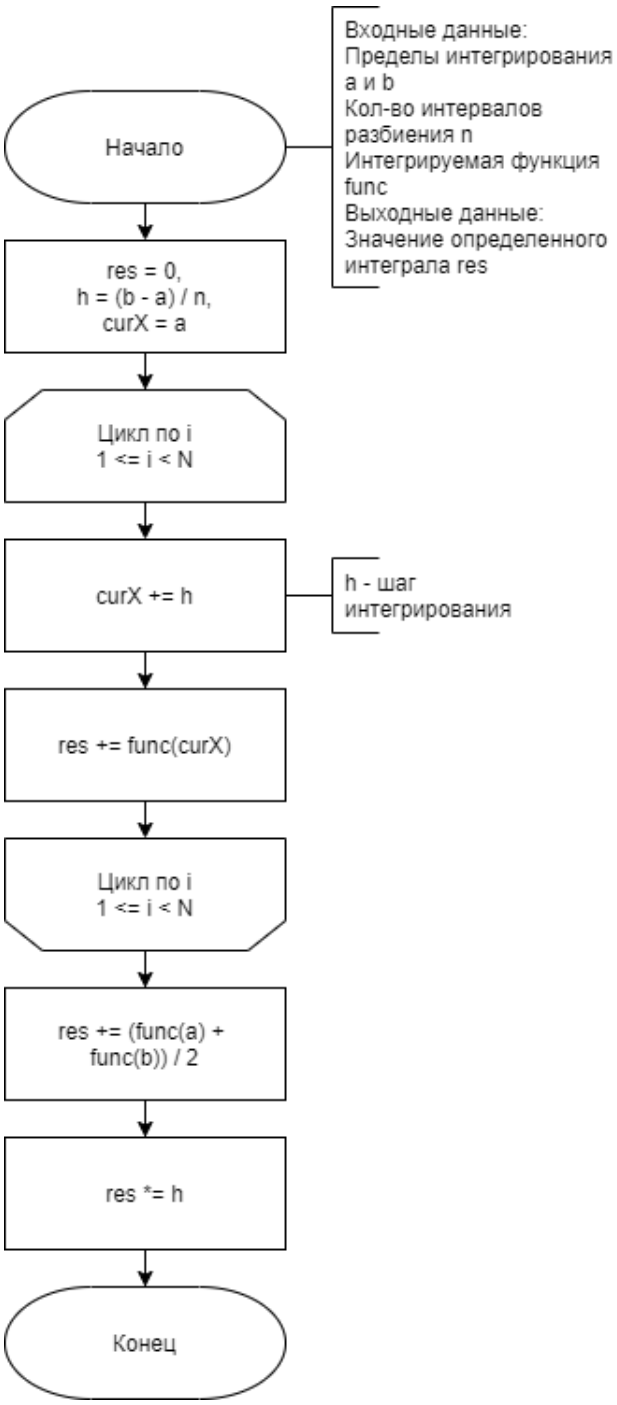

Рисунок 2.2 – Схема алгоритма вычисления значения определенного интеграла

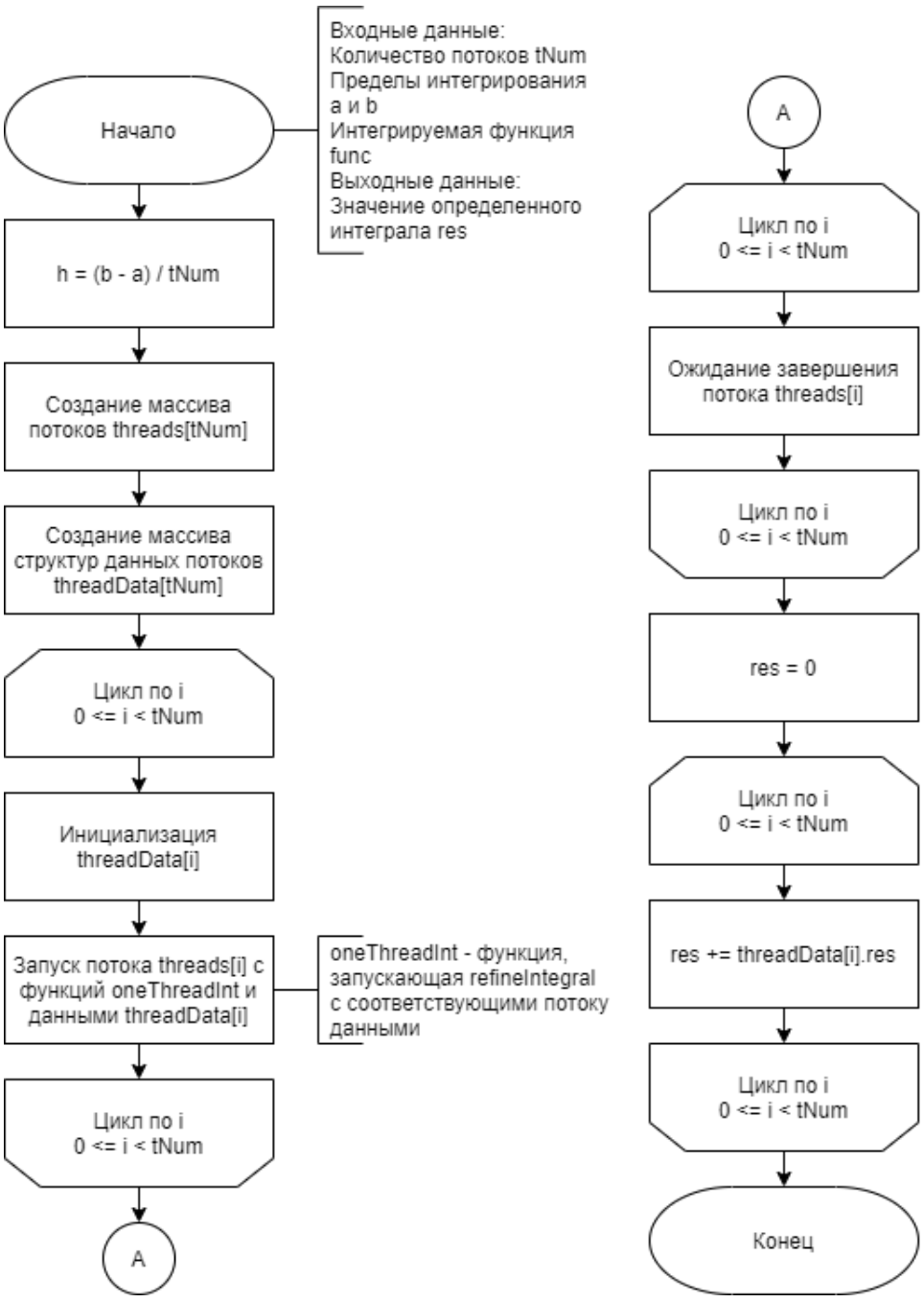

Рисунок 2.3 – Схема алгоритма параллельного вычисления значения определенного интеграла

2.2 Используемые типы данных

При реализации алгоритмов будут использованы следующие структуры данных:

- границы интервалов интегрирования, аргументы и значения функций - вещественные числа double;
- потоки - массив типа pthread_t;
- данные потоков - массив типа threadData.

Тип threadData описан в листинге ниже.

Листинг 2.1 – Тип данных потока

```
typedef double(*integralFunc)(double);

struct threadData{
    int ind;
    double a;
    double h;
    integralFunc func;
    double res;
};
```

2.3 Оценка памяти

Рассмотрим затрачиваемый объем памяти для рассмотренных алгоритмов.

При последовательном вычислении память используется на:

- входные параметры функции - 2 * sizeof(double) + sizeof(double *);
- возвращаемое значение - sizeof(double);
- вспомогательные переменные - sizeof(int) + sizeof(double);
- память, используемая функцией calcInt, а именно:

    - входные параметры функции - 2 * sizeof(double) + sizeof(int) + sizeof(double *);
    - возвращаемое значение - sizeof(double);
    - вспомогательные переменные - 3 * sizeof(double).

В случае параллельного вычисления для каждого потока помимо описанной выше используемой памяти выделяется структура threadData, размер которой равен sizeof(double) + sizeof(int) + sizeof(double *). Также внутри функции, создающей потоки выделяется память:

- возвращаемое значение - sizeof(double);
- массив потоков - $tNum$ * sizeof($pthread\_t$);
- вспомогательные переменные - 2 * sizeof(double).

Таким образом, для параллельного вычисления используется в $tNum$ раз больше памяти, где $tNum$ - количество потоков.

2.4 Структура ПО

ПО будет состоять из следующих модулей:

- главный модуль - из него будет осуществляться запуск программы и выбор соответствующего режима работы;
- модуль интерфейса - в нем будет описана реализация режимов работы программы;
- модуль, содержащий реализации алгоритмов.

2.5 Вывод

На основе полученных в аналитическом разделе знаний об алгоритмах были спроектированы схемы алгоритмов, выбраны используемые типы данных, проведена оценка затрачиваемого объема памяти, а также описана структура ПО.

3. Технологический раздел

В данном разделе будут приведены требования к ПО, средства его реализации и листинга кода алгоритмов, а также рассмотрены тестовые случаи.

3.1 Требования к ПО

Программное обеспечение должно удовлетворять следующим требованиям:

- ПО принимает вещественные пределы интегрирования и указатель на интегрируемую функцию;
- ПО возвращает вещественное число - результат интегрирования.

3.2 Средства реализации

Для реализации ПО был выбран компилируемый язык C. В качестве среды разработки - QtCreator. Оба средства были выбраны из тех соображений, что навыки работы с ними были получены в более ранних курсах.

3.3 Листинги кода

Листинги 1 - 3 демонстрируют реализацию алгоритмов.

Листинг 3.1 – Алгоритм последовательного вычисления значения определенного интеграла

```
double refineIntegral(double a, double b, integralFunc func)
{
    int n = 2;
    double refRes = calculateIntegral(a, b, n, func);
    double res = 0;
    while (fabs(res - refRes) > EPS)
    {
        res = refRes;
```

```
        n *= 2;
        refRes = calculateIntegral(a, b, n, func);
    }
    return res;
}
```

Листинг 3.2 – Алгоритм вычисления значения определенного интеграла

```
double calculateIntegral(double a, double b, int n, integralFunc func)
{
    double result = 0;
    double h = (b - a) / n;
    double curX = a;
    for (int i = 1; i < n; i++)
    {
        curX += h;
        result += func(curX);
    }
    result += (func(a) + func(b)) / 2;
    result *= h;
    return result;
}
```

Листинг 3.3 – Алгоритм параллельного вычисления значения определенного интеграла

```
double integralWithThreads(int tNum, double a, double b, integralFunc func)
{
    double h = (b - a) / tNum;
    pthread_t threads[tNum];
    struct threadData threadData[tNum];
    for (int i = 0; i < tNum; i++)
    {
        threadData[i].ind = i;
        threadData[i].a = a;
        threadData[i].h = h;
        threadData[i].func = func;
        threadData[i].res = 0;
        pthread_create(&(threads[i]), NULL,\
                       oneThreadIntegral, &threadData[i]);
    }

    for (int i = 0; i < tNum; i++)
        pthread_join(threads[i], NULL);

    double res = 0;
    for (int i = 0; i < tNum; i++)
        res += threadData[i].res;
```

```
23      return res;
24  }
```

3.4 Тестирование ПО

В таблице 3.1 приведены тестовые случаи для алгоритмов интегрирования, причем функция $f = x^2 + 4x - 21$. Случай 1 - совпадение левого и правого пределов, случай 2 - левый предел больше правого, случаи 3-4 - значение интеграла ненулевое.

Таблица 3.1 – Тестовые случаи

№	Пределы интегрирования	Результат
1	0 0	0
2	-1 -8	Некорректные пределы интегрирования
3	10 100	350910.000
4	-1 3	-58.666

3.5 Вывод

На основе схем из конструкторского раздела были написаны реализации требуемых алгоритмов, а также проведено их тестирвание.

4. Исследовательский раздел

В данном разделе будет проведен сравнительный анализ алгоритмов по времени и затрачиваемой памяти.

4.1 Технические характеристики

Тестирование выполнялось на устройстве со следующими характеристиками:

- Операционная система Windows 10 [3]
- Память 8 Гб
- Процессор Intel Core i3 7020U, 2.3 ГГц [4]

Во время проведения эксперимента устройство не было нагружено сторонними задачами, а также было подключено к блоку питания.

Замеры времени осуществлялись с помощью утилиты библиотеки windows.h. Результат был умножен на $1e6$, чтобы были получены микросекунды.

Листинг 4.1 – Замер времени в микросекундах

```
{
    LARGE_INTEGER frequency;
    LARGE_INTEGER start;
    LARGE_INTEGER end;
    double interval;
    QueryPerformanceFrequency(&frequency);
    QueryPerformanceCounter(&start);
    res = refineIntegral(a, b, func1);
    QueryPerformanceCounter(&end);
    interval = (double) (end.QuadPart - start.QuadPart)\
            / frequency.QuadPart * MCS;
}
```

При проведении эксперимента была отключена оптимизация командой компилятору -o0 [5].

4.2 Оценка времени работы алгоритмов

В таблице 4.1 представлены замеры времени работы алгоритмов на интервалах интегрирования длинной от 10 до 100. Каждое значение было получено усреднением по 100 замерам. Максимальное количество запускаемых потоков было выбрано 16, так как процессор содержит 4 логических ядра [4].

Таблица 4.1 – Временные замеры работы алгоритмов

L	Последоват.	1 п.	2 п.	4 п.	8 п.	16 п.
10	831.67	802.52	315.80	95.74	93.05	127.32
20	6419.24	6195.73	2307.03	681.79	559.40	675.64
30	26243.15	24761.44	9448.83	2136.75	1662.55	1955.23
40	53126.95	49403.12	18451.95	5417.52	4310.91	5041.34
50	111396.90	110161.73	36901.99	10837.60	8203.71	9651.27
60	115938.46	104935.93	75092.23	17391.07	13456.72	15761.32
70	114874.66	99674.98	123571.19	30062.82	23765.29	28051.00
80	126957.68	99492.35	149183.09	44076.26	36326.88	43359.88
90	142689.09	98890.41	149129.21	72630.98	60292.21	70491.01
100	168881.42	99556.42	200810.29	91269.81	78802.60	93109.37

4.3 Вывод

По результатам исследования можно сделать вывод, что самым быстрым для процессора является использование 8 потоков. Если количество потоков увеличить до 16 время увеличивается, так как за квант процессорного времени процессы не успевают обработаться, из-за чего увеличивается объем "распаковок"и "запаковок"используемых процессором данных.

Заключение

В ходе выполнения данной лабораторной работы были выполнены следующие задачи:

- изучены последовательный и параллельный варианты алгоритма вычисления определенного интеграла;
- составлены схемы данных алгоритмов;
- реализованны разработанные версии алгоритмов;
- проведен сравнительный анализ алгоритмов по затрачиваемым ресурсам (время и память);
- описаны и обоснованы полученные результаты.

В результате исследований можно прийти к выводу, что использование параллельного алгоритма вычисления определенного интеграла по времени является в 10 раз оптимальнее последовательного (при использовании 8 потоков). По памяти же многопоточная реализация оказывается более затратной, так как выделяется память под массив указателей на потоки и массив с данными для каждого потока отдельно.

Список литературы

[1] Вычисление определенного интеграла методом трапеции [Электронный ресурс]. Режим доступа: http://www.cleverstudents.ru/integral/method_of_trapezoids.html (дата обращения: 28.10.2021).

[2] Потоки и работа с ними [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/standard/threading/threads-and-threading (дата обращения: 28.10.2021).

[3] Windows 10 [Электронный ресурс]. Режим доступа: https://www.microsoft.com/ru-ru/windows/windows-10-specifications (дата обращения: 18.10.2021).

[4] Процессор Intel Core i3-7020u [Электронный ресурс]. Режим доступа: https://www.intel.ru/content/www/ru/ru/products/sku/122590/intel-core-i37020u-processor-3m-cache-2-30-ghz/specifications.html (дата обращения: 18.10.2021).

[5] ISO/IEC 9899:1999 [Электронный ресурс]. Режим доступа: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf (дата обращения: 22.10.2021).