|  |  |
| --- | --- |
| |  | | --- | | МИНОБРНАУКИ РОССИИ  Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |

Институт Информационных технологий

Кафедра МОСИТ

**ПРАКТИЧЕСКАЯ РАБОТА №2**

по дисциплине «Параллельное программирование»

**Студент группы** ИКБО-11-17 Алиев Ю.А \_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись студента)*

**Руководитель работы** Сыромятников В. П. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись руководителя)*

Москва, 2020

Оглавление

[**Тема работы** 2](#_Toc54222316)

[**Цель работы** 2](#_Toc54222317)

[**Постановка задачи** 2](#_Toc54222318)

[**Теоретическое введение** 2](#_Toc54222319)

[**Описание алгоритмов, используемых для решения задачи** 3](#_Toc54222320)

[**Текст исходного кода программы** 7](#_Toc54222321)

[**Тестирование** 8](#_Toc54222322)

[**Выводы** 14](#_Toc54222323)

[**Список используемых информационных источников** 15](#_Toc54222324)

# **Практическая работа №2**

Параллельное программирование с использованием основ технологии OpenMP.

# **Цель работы**

* Отработка принципов распараллеливания программ;
* Реализация простейшей многопоточной программы;
* Анализ эффективности параллельной программы по сравнению с последовательной программой.

# **Постановка задачи**

Составить программу последовательного и параллельного вычисления определённого интеграла методом прямоугольников. Провести контрольные прогоны программы для числа разбиений отрезка интегрирования n = 104, 105, 106, 107, 108 и установленных для параллельного варианта количествах потоков p = 4, 8, 16, 32 и 64 с вычислением времени выполнения и ускорения. Полученные результаты свести в сводную таблицу.

Построить графики изменения ускорения при последовательном и параллельных вычислениях в зависимости от числа разбиений отрезка интегрирования. Построить графики изменения ускорения при параллельных вычислениях в зависимости от количества используемых потоков. Вычислить показатели эффективности и стоимости параллельной реализации программы. Провести анализ полученных результатов. Сделать выводы о проделанной работе, основанные на полученных данных.

# **Теоретическое введение**

Метод прямоугольников — метод [численного интегрирования](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B8%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) функции одной переменной, заключающийся в замене подынтегральной функции на многочлен нулевой степени, то есть константу, на каждом элементарном отрезке. Если рассмотреть график подынтегральной функции, то метод будет заключаться в приближённом вычислении площади под графиком суммированием площадей конечного числа прямоугольников, ширина которых будет определяться расстоянием между соответствующими соседними узлами интегрирования, а высота — значением подынтегральной функции в этих узлах. [Алгебраический порядок точности](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D0%B8_%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B0) равен 0. (Для формулы средних прямоугольников равен 1.

1. – метод левый прямоугольников
2. – метод правых прямоугольников
3. – метод средних прямоугольников

В случае разбиения отрезка интегрирования на  n элементарных отрезков приведённые выше формулы применяются на каждом из этих элементарных отрезков между двумя соседними узлами. В результате, получаются составные квадратурные формулы;

1. – для левых прямоугольников
2. – для правых прямоугольников
3. – для средних прямоугольников

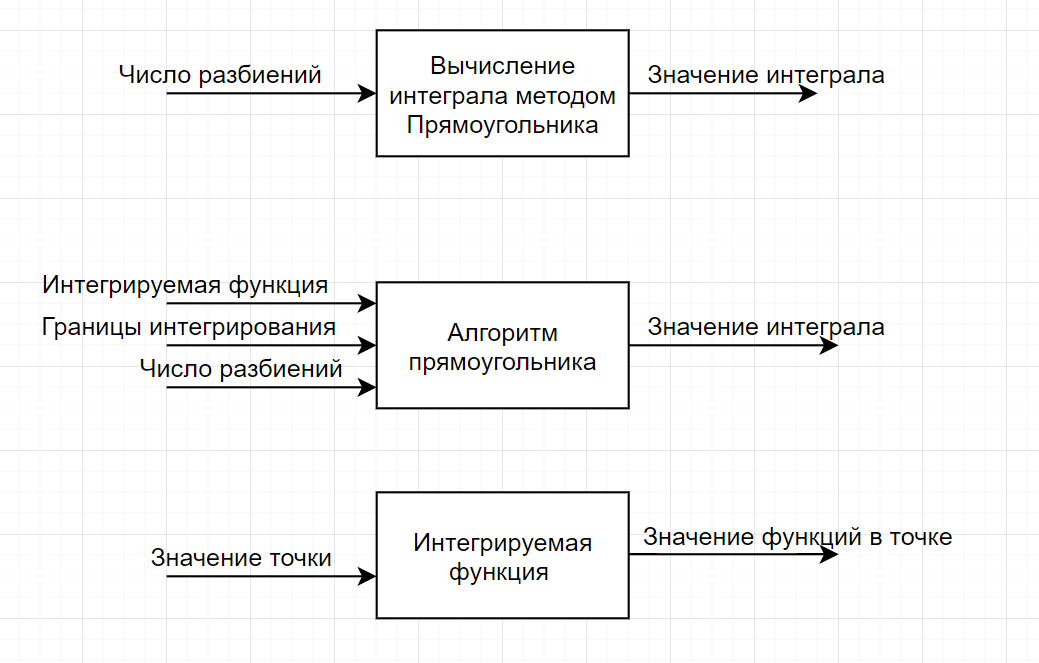
Ускорение параллельного алгоритма является его наиболее информативной характеристикой, которая показывает во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Ускорение параллельного алгоритма определяется величиной SN=T1/TN, где T1 – время выполнения алгоритма на одном процессоре, TN – время выполнения алгоритма на N процессорах.

Эффективность (efficiency) использования параллельным алгоритмом процессоров при решении задачи определяется соотношением E = (n \* p) / (n + 2 \* p^2) (величина эффективности определяет среднюю долю времени выполнения алгоритма, в течение которой процессоры реально задействованы для решения задачи). Она вычисляется как Sn/nmax, где nmax – количество процессоров в системе.

Стоимость (cost) вычислений – метод, стоимость которого является пропорциональной времени выполнения наилучшего последовательного алгоритма. Вычисляется по формуле Cp = p\*Tp (число процессоров \* время выполнения задачи)

# **Описание алгоритмов, используемых для решения задачи**

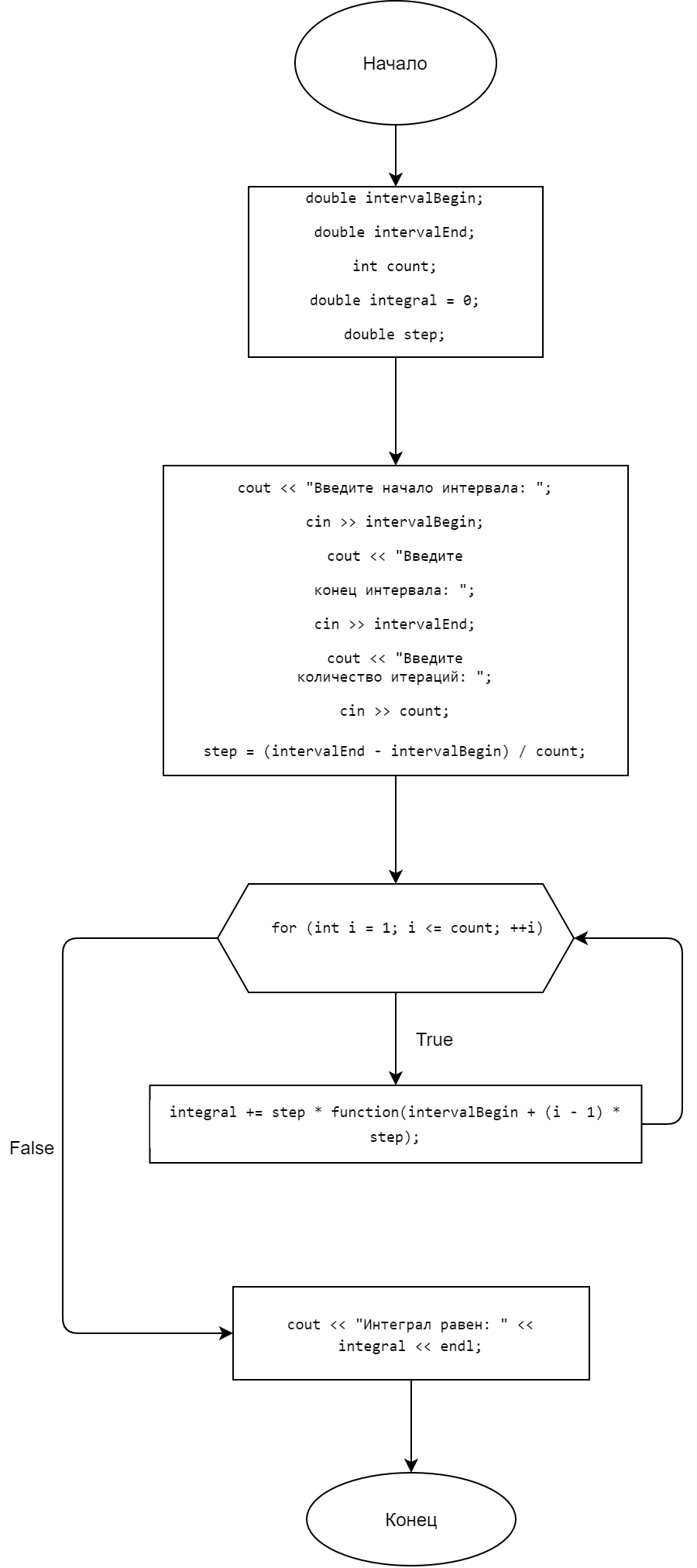
Для выполнения данной практической работы было решено провести декомпозицию общего процесса на более мелкие составляющие (рисунок 1).



*Рисунок 1. Декомпозиция общей задачи на подзадачи*

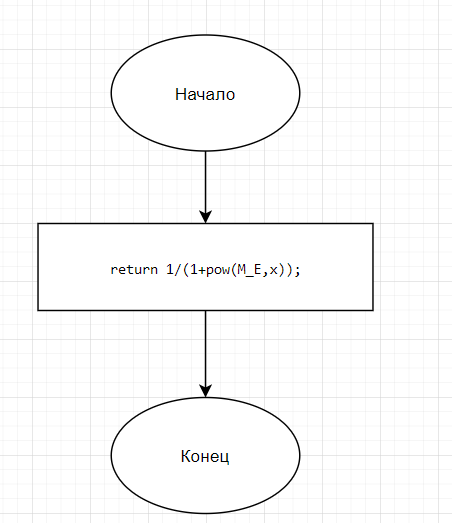
После проведения декомпозиции опишем использование всех алгоритмов, используемых в решении общей задачи.

Во-первых, ниже продемонстрирован рисунок общего алгоритма работы программы (рисунок 2).



*Рисунок 2. Схема алгоритма работы программы*

Во-вторых, продемонстрирован рисунок алгоритма вычисления значения функции в точке (рисунок 4).



*Рисунок 4. Схема алгоритма вывода элементов массива на экран*

# **Текст исходного кода программы**

|  |
| --- |
| #include <iostream>  #include <math.h>  #include <omp.h>  using namespace std;  double function(double x);  int main() {  double intervalBegin;  double intervalEnd;  int count;  double integral = 0;  double step;  cout << "Введите начало интервала: ";  cin >> intervalBegin;  cout << "Введите конец интервала: ";  cin >> intervalEnd;  cout << "Введите количество итераций: ";  cin >> count;  step = (intervalEnd - intervalBegin) / count;  #pragma omp parallel for  for (int i = 1; i <= count; ++i) {  integral += step \* function(intervalBegin + (i - 1) \* step);  }  cout << "Интеграл равен: " << integral << endl;  int a;    cin >> a;  return 0;  }  double function(double x) {  return 1/(1+pow(M\_E,x));  } |
|  |

Листинг файла practice.cpp. Исходный код программы.

# **Тестирование**

На основе выполненных тестов были составлены сводные таблицы, на основе которых составлены графики, указанные в основной задаче.

*Таблица 1. Результаты работы сортировки без использования методов распараллеливания*

|  |  |  |
| --- | --- | --- |
| № тестирования | Количество разбиений отрезков, n | Время выполнения работы программы, сек. |
| 1 | 10000 | 0.005 |
| 2 | 100000 | 0.009 |
| 3 | 1000000 | 0.020 |
| 4 | 10000000 | 0.097 |
| 5 | 100000000 | 3.462 |

*Таблица 2. Результаты работы сортировки с использованием методов распараллеливания при количестве потоков p = 4*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № тестирования | Количество элементов массива, n | Время выполнения программы, сек | Ускорение (Sn) | Эффективность | Стоимость |
| 1 | 10000 | 0.033 | 0.09 | 0.008 | 0.132 |
| 2 | 100000 | 0.014 | 0.71 | 0.004 | 0.056 |
| 3 | 1000000 | 0.022 | 3.81 | 0.006 | 0.088 |
| 4 | 10000000 | 0.245 | 7.61 | 0.061 | 0.98 |
| 5 | 100000000 | 2.317 | 3.59 | 0.579 | 9.268 |

*Таблица 3. Результаты работы сортировки с использованием методов распараллеливания при количестве потоков p = 8*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № тестирования | Количество элементов массива, n | Время выполнения программы, сек | Ускорение (Sn) | Эффективность | Стоимость |
| 1 | 10000 | 0.020 | 0.15 | 0.0025 | 0.16 |
| 2 | 100000 | 0.053 | 0.19 | 0.006625 | 0.424 |
| 3 | 1000000 | 0.06 | 1.4 | 0.0075 | 0.48 |
| 4 | 10000000 | 0.240 | 7.78 | 0.03 | 1.92 |
| 5 | 100000000 | 2.068 | 4.03 | 0.2585 | 16.544 |

*Таблица 4. Результаты работы сортировки с использованием методов распараллеливания при количестве потоков p = 16*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № тестирования | Количество элементов массива, n | Время выполнения программы, сек | Ускорение (Sn) | Эффективность | Стоимость |
| 1 | 10000 | 0.002 | 1.5 | 0.000125 | 0.032 |
| 2 | 100000 | 0.056 | 0.18 | 0.0035 | 0.896 |
| 3 | 1000000 | 0.052 | 1.62 | 0.00325 | 0.832 |
| 4 | 10000000 | 0.176 | 10.59 | 0.011 | 2.816 |
| 5 | 100000000 | 2.097 | 3.98 | 0.1310625 | 33.552 |

*Таблица 5. Результаты работы сортировки с использованием методов распараллеливания при количестве потоков p = 32*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № тестирования | Количество элементов массива, n | Время выполнения программы, сек | Ускорение (Sn) | Эффективность | Стоимость |
| 1 | 10000 | 0.025 | 0.12 | 0.00078125 | 0.8 |
| 2 | 100000 | 0.004 | 2.5 | 0.000125 | 0.128 |
| 3 | 1000000 | 0.021 | 4.0 | 0.00065625 | 0.672 |
| 4 | 10000000 | 0.274 | 6.81 | 0.0085625 | 8.768 |
| 5 | 100000000 | 2.127 | 3.92 | 0.06646875 | 68.064 |

*Таблица 6. Результаты работы сортировки с использованием методов распараллеливания при количестве потоков p = 64*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № тестирования | Количество элементов массива, n | Время выполнения программы, сек | Ускорение (Sn) | Эффективность | Стоимость |
| 1 | 10000 | 0,045 | 0.067 | 0.0007 | 2.88 |
| 2 | 100000 | 0,029 | 0.345 | 0.0004 | 1.856 |
| 3 | 1000000 | 0,054 | 1.56 | 0.0008 | 3.456 |
| 4 | 10000000 | 0,167 | 11.16 | 0.0026 | 10.688 |
| 5 | 100000000 | 1,719 | 4.85 | 0.0269 | 110.016 |

*Рисунок 7. Зависимости ускорения при различных потоках от количества элементов массива*

На основе данного рисунка можно сделать вывод о том, что ускорение происходит на больших величинах (в районе 10000000 разбиений) больше всего при выполнении алгоритма на 64 и 16 потоков. Самое же низкое ускорение достигается при использовании 32 потоков.

*Рисунок 8. Графики зависимости времени выполнения при различных потоках от количества элементов массива*

На основе данного рисунка можно сделать вывод о том, что самое долгое затраченное время на вычисление интеграла при разбиении отрезка от 10000 до 1000000000 выполнение происходит у не распараллеленного алгоритма. При использовании потоков время выполнения намного меньше и все примерно равны, где наименьшее время выполнения у 64 потоков.

*Рисунок 9. Графики зависимости эффективности выполнения при различных потоках от количества элементов массива*

На основе данного рисунка можно сделать вывод о том, самая высокая эффективность достигается при 4 поточном вычислений значений. Усредненная эффективность лучше всего достигается при выполнении кода при 16 потоках. Худший вариант – использование 64 потоков.

*Рисунок 10. Графики зависимости стоимости выполнения при различных потоках от количества элементов массива*

На основе данного рисунка можно сделать вывод о том, самая высокая стоимость выполнения достигается при использовании 64 потоков, самая маленькая – при использовании 4 потоков.

# **Выводы**

В соответствии с полученными данными построены различные графики зависимости, к которым сделаны соответствующие логические выводы.

1. Наилучшая стоимость при 64 потоках
2. Наилучшая эффективность при 64 потоках
3. Наилучшее время при 4 потоках
4. Наилучшее ускорение при 64 потоках
5. Наихудшая стоимость при 4 потоках
6. Наихудшее эффективность при 64 потоках
7. Наихудшее время при 64 потоках
8. Наихудшее ускорение при 4 потоках

# **Список используемых информационных источников**

1. Сыромятников В. П. Курс лекций по дисциплине «Параллельное программирование». – РТУ МИРЭА, 2020-2021 г.

2. Руководство по использованию модуля OpenMp [Электронный ресурс]. – Режим доступа: <http://ccfit.nsu.ru/arom/data/openmp.pdf> (дата обращения 18.10.2020)

3. Официальные характеристики процессора Intel Core i5-6300HQ [Электронный ресурс]. – Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/80815/intel-core-i5-4590-processor-6m-cache-up-to-3-70-ghz.html> (дата обращения 18.10.2020)

4. Официальный сайт дистрибутива MVS Code [Электронный ресурс]. – Режим доступа: <https://code.visualstudio.com> (дата обращения 18.10.2020)