**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**Кафедра «Информатика и программное обеспечение»**

**ЛАБОРАТОРНАЯ РАБОТА № 4**

**ПО ДИСЦИПЛИНЕ**

**«Архитектура вычислительных систем»**

**Тема: ТЕХНОЛОГИЯ OPENmp: этапы разработки параллельных программ на примере численного интегрирования**

Выполнил студент гр. О-21-ИВТ-1-по-Б

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шпаков В.В.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

Преподаватель

\_\_\_\_\_\_\_\_\_\_\_ к.ф.-м.н., доц. Дмитроченко О.Н.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

**Брянск 2024**

**Содержание**

[1. Цель работы 3](#_Toc155915190)

[2. Теоретические сведения 4](#_Toc155915191)

[2.1. Особенности параллельного программирования 4](#_Toc155915192)

[2.2. Некоторые методы вычисления определенного интеграла 4](#_Toc155915193)

[Формула прямоугольников 5](#_Toc155915194)

[Формула трапеций 5](#_Toc155915195)

[2.3. Применение параллельного программирования в численном интегрировании 6](#_Toc155915196)

[2.4. Автоматический выбор шага интегрирования 7](#_Toc155915197)

[3. Задания 9](#_Toc155915198)

[3.1. Задание 1 (Метод левых прямоугольников) 9](#_Toc155915199)

[3.2. Задание 2 (Метод правых прямоугольников) 11](#_Toc155915200)

[3.3. Задание 3 (Метод средних прямоугольников) 13](#_Toc155915201)

[3.4. Задание 4 (Метод трапеций) 14](#_Toc155915202)

[3.5. Задание 5 (Метод Симпсона) 16](#_Toc155915203)

[4. Контрольные вопросы 18](#_Toc155915204)

# Цель работы

Целью лабораторной работы является:

1. изучение этапов разработки параллельных программ и технологий преобразования традиционных алгоритмов для исполнения на многопроцессорных системах;
2. изучение основных методов численного интегрирования и механизма автоматического выбора шага интегрирования для достижения заданной точности вычисления;
3. приобретение практических навыков преобразования последовательных алгоритмов численного интегрирования в параллельные с использованием возможностей технологии OpenMP.

# Теоретические сведения

## Особенности параллельного программирования

Процесс разработки параллельных программ содержит все этапы более традиционного процесса разработки последовательных программ. Однако процесс разработки параллельных программ содержит три дополнительных этапа.

1. Определение параллелизма: анализ задачи с целью выделения подзадач, которые могут выполняться одновременно.
2. Выявление параллелизма: изменение структуры задачи таким образом, чтобы можно было эффективно выполнять подзадачи. Для этого часто требуется определить зависимость между подзадачами и организовать исходный код так, чтобы ими можно было эффективно управлять.
3. Выражение параллелизма: реализация параллельного алгоритма в исходном коде программы с помощью системы обозначений параллельного программирования. Системой обозначения параллельного программирования является язык параллельного программирования, например OpenMP.

Наиболее эффективны в распараллеливании итеративные алгоритмы. Потоки параллельной программы, реализующей итеративный алгоритм, работают совместно над решением одной задачи, описанной в теле цикла. Потоки взаимодействуют и синхронизируются с помощью разделяемых переменных, например, счетчика цикла.

## Некоторые методы вычисления определенного интеграла

При численном интегрировании предполагается вычисление значения определённого интеграла , основанное на том, что значение интеграла численно равно площади криволинейной трапеции, ограниченной осью абсцисс, графиком интегрируемой функции и отрезками прямых  и , где  и  — пределы интегрирования (см. рис. 1).

Численное интегрирование основано на замене интеграла суммой. Для этого отрезок  разбивается на  частей точками (узлами интегрирования) . Как правило, рассматривается равномерная сетка разбиения, т.е. , . Тогда определенный интеграл можно представить в виде суммы . Для построения общей формулы численного интегрирования на всем отрезке  становится достаточным построить формулу для нахождения интеграла на частичном отрезке .

Формула прямоугольников

Метод заключается в приближённом вычислении площади под графиком суммированием площадей конечного числа прямоугольников, ширина которых определяется расстоянием между соответствующими соседними узлами интегрирования, а высота — значением подынтегральной функции в этих узлах.

**Формула левых прямоугольников.** Пусть, , т.е. осуществляется аппроксимация левой кусочно–линейной интерполяцией (см. рис. 2). Тогда:



Таким образом, .

**Формула правых прямоугольников.** Аналогичным образом в случае , , получаем (см. рис. 3):

.

**Формула средних прямоугольников.** Функция на отрезке заменяется на ее значение в середине отрезка (см. рис. 4), т.е. , , . Тогда, получаем:



Формула трапеций

Площадь криволинейной трапеции, заменяется на площадь прямоугольной трапеции (рис. 5), т.е.



Тогда для всего отрезка получаем:



При аппроксимации интеграла , функцию  на отрезке  заменяют параболой, проходящей через точки , ,  (см. рис. 6). Тогда:

,

где , , .

Формула Симпсона для вычисления интеграла на отрезке :



## Применение параллельного программирования в численном интегрировании

Аналогом программы «Hello world» в учебниках по параллельному программированию является программа реализации алгоритма вычисления значения определенного интеграла функции на отрезке  с помощью формулы средних прямоугольников. Так аналитически может быть показано, что , сравнение численного значения определенного интеграла с числом  позволяет проверить корректность программы реализации параллельного алгоритма.

Рассмотрим процесс преобразования последовательной программы вычисления значения определенного интеграла в параллельную.

* **Определение параллелизма.** В последовательной программе имеется единственный цикл, который требуется распараллелить.
* **Выявление параллелизма.** Итерации цикла полностью независимы, если не считать значения зависимой переменной  и накопительной переменной .
* **Выражение параллелизма.** Используем для этого директивы, опции и функции библиотеки OpenMP.

Так как переменная  используется как временное хранилище для результатов вычислений в итерации цикла, ее необходимо сделать локальной для каждого потока с помощью опции *private.* Разделяемой потоками программы, выполняющими итерации цикла, является переменная-счетчик *i*, однако в OpenMP она автоматически становится локальной закрытой переменной для каждого потока.

Накопительная переменная *sum* используется для суммирования. Это классический пример редукции, поэтому можно применить опцию *reduction*.

После добавления этих опций к конструкции *parallel for*, получим программу вычисления определенного интеграла, распараллеленную с помощью технологии OpenMP.

С помощью директивы *include* был также подключен заголовочный файл *openmp.h*. Это позволяет определить типы и библиотечные подпрограммы времени выполнения, которые иногда требуются при написании программ с помощью технологии OpenMP. В программе эти компоненты технологии не использовались, но хороший стиль программирования предполагает добавление подключаемого файла *openmp.h*.

## Автоматический выбор шага интегрирования

При вычислении значения определенного интеграла от функций, заданных аналитически, необходимо обеспечить заданную точность вычисления .

Точность вычисления можно повысить двумя способами:

1. использовать более точный метод вычисления определенного интеграла;
2. увеличить число узлов, соответственно уменьшить шаг интегрирования .

На практике обычно заданная точность вычисления достигается вторым из указанных способов. Вычисляется значение определенного интеграла  с выбранным числом узлов , затем значение определенного интеграла  с удвоенным числом узлов. Если результаты отличаются более чем на требуемое значение точности, число узлов вновь удваивается. Вычисление заканчивают, когда , принимая, что , т. е. последнее вычисленное приближенное значение интеграла  отличается от точного значения  не больше чем на заданную точность.

# Задания

В каждом варианте задания указана функция и отрезок, на котором необходимо вычислить значение определенного интеграла . Необходимо написать параллельную программу вычисления определенного интеграла с использованием различных методов численного интегрирования: по формулам левых, правых и средних прямоугольников, формуле трапеций и формуле Симпсона.

Параллельная программа должна осуществлять динамический выбор шага разбиения отрезка интегрирования для вычисления значения определенного интеграла при различной точности: , , .

**Задание.** Вывести на экран время вычисления значения определенного интеграла различными методами и общее количество выполненных итераций в процессе вычисления.

Вариант 23.

Отрезок интегрирования [1,3]

## Задание 1 (Метод левых прямоугольников)

*Листинг 1*

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <ctime>  #include <cmath>  void computeIntegral(double a, double b, double epsilon, double timeStart)  {  int count = 0;  double presentF = 0, nextF = 0, height = 0;  while (1)  {  if (!count)  {  count = 1;  height = (b - a) / count;  #pragma omp parallel for reduction(+:presentF)  for (int i = 0; i < count; i++)  {  double x = a + (i \* height);  presentF += sqrt(1 + 0.5 \* x + log(x) \* log(x));  }  presentF \*= height;  }  else  {  count \*= 2;  nextF = 0;  height = (b - a) / count;  #pragma omp parallel for reduction(+:nextF)  for (int i = 0; i < count; i++)  {  double x = a + (i \* height);  nextF += sqrt(1 + 0.5 \* x + log(x) \* log(x));  }  nextF \*= height;  if (fabs((nextF - presentF)) <= epsilon)  {  std::cout << "Точность: " << epsilon << " знаков" << std::endl;  std::cout << "Общее количество итераций: " << count << std::endl;  std::cout << "Время вычисления значения: " << (omp\_get\_wtime() - timeStart) << " секунды" << std::endl;  break;  }  else  {  presentF = nextF;  }  }  }  }  int main()  {  double a = 1.0;  double b = 3.0;  double epsilon[] = { 1e-5, 1e-6, 1e-7 };  setlocale(LC\_ALL, "Russian");  srand(time(NULL));  for (int i = 0; i < 3; ++i)  {  std::cout << "Метод левых прямоугольников:" << std::endl;  double timeStart = omp\_get\_wtime();  computeIntegral(a, b, epsilon[i], timeStart);  std::cout << std::endl;  }  return 0;  } |

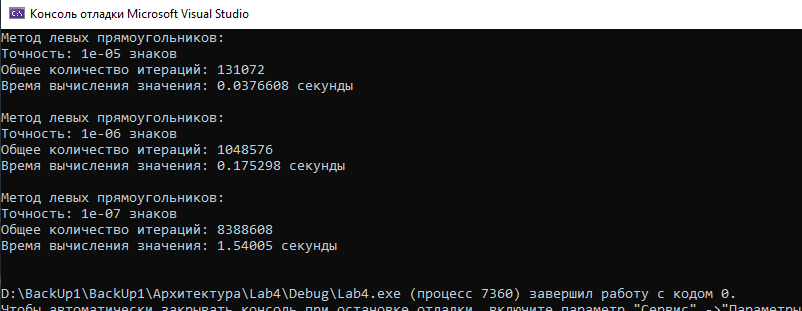


Рис. . Результат выполнения

## Задание 2 (Метод правых прямоугольников)

*Листинг 2*

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <cmath>  #include <ctime>  void computeIntegral(double a, double b, double epsilon, double timeStart);  int main()  {  double a = 1.0;  double b = 3.0;  double epsilon[] = { 1e-5, 1e-6, 1e-7 };  setlocale(LC\_ALL, "Russian");  for (int i = 0; i < 3; ++i)  {  std::cout << "Метод правых прямоугольников:" << std::endl;  double timeStart = omp\_get\_wtime();  computeIntegral(a, b, epsilon[i], timeStart);  std::cout << std::endl;  }  return 0;  }  void computeIntegral(double a, double b, double epsilon, double timeStart)  {  int count = 0;  double presentF = 0, nextF = 0, height = 0;  while (1)  {  if (!count)  {  count = 1;  height = (b - a) / count;  #pragma omp parallel for reduction(+:presentF)  for (int i = 0; i < count; i++)  {  double x = a + ((i + 1) \* height);  presentF += sqrt(1 + 0.5 \* x + log(x) \* log(x)); // ваша функция  }  presentF \*= height;  }  else  {  count \*= 2;  nextF = 0;  height = (b - a) / count;  #pragma omp parallel for reduction(+:nextF)  for (int i = 0; i < count; i++)  {  double x = a + ((i + 1) \* height);  nextF += sqrt(1 + 0.5 \* x + log(x) \* log(x)); // ваша функция  }  nextF \*= height;  if (fabs((nextF - presentF)) <= epsilon)  {  std::cout << "Точность: " << epsilon << " знаков" << std::endl;  std::cout << "Общее количество итераций: " << count << std::endl;  std::cout << "Время вычисления значения: " << (omp\_get\_wtime() - timeStart) << " секунды" << std::endl;  break;  }  else  {  presentF = nextF;  }  }  }  } |

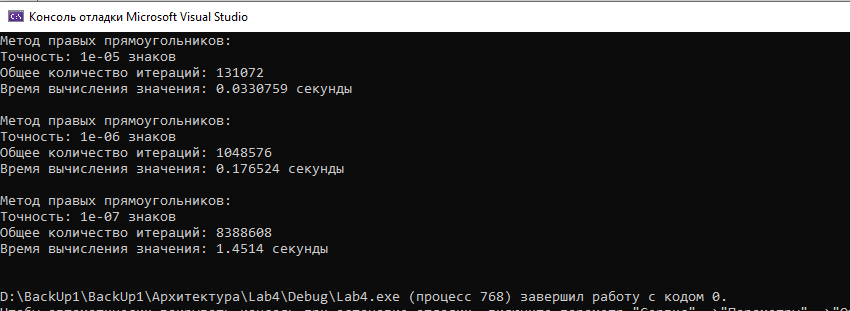


Рис. .Результат выполнения

## Задание 3 (Метод средних прямоугольников)

*Листинг 4*

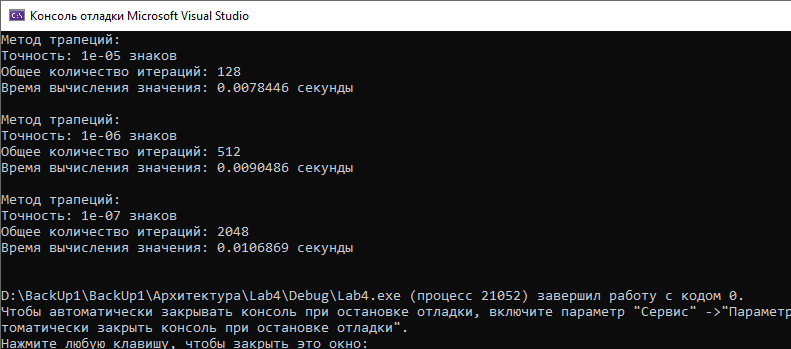
|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <cmath>  #include <ctime>  void computeIntegral(double a, double b, double epsilon, double timeStart);  int main()  {  double a = 1.0;  double b = 3.0;  double epsilon[] = { 1e-5, 1e-6, 1e-7 };  setlocale(LC\_ALL, "Russian");  for (int i = 0; i < 3; ++i)  {  std::cout << "Метод средних прямоугольников:" << std::endl;  double timeStart = omp\_get\_wtime();  computeIntegral(a, b, epsilon[i], timeStart);  std::cout << std::endl;  }  return 0;  }  void computeIntegral(double a, double b, double epsilon, double timeStart)  {  int count = 0;  double presentF = 0, nextF = 0, height = 0;  while (1)  {  if (!count)  {  count = 1;  height = (b - a) / count;  #pragma omp parallel for reduction(+:presentF)  for (int i = 0; i < count; i++)  {  double x = a + (i \* height) + (height / 2);  presentF += sqrt(1 + 0.5 \* x + log(x) \* log(x)); // ваша функция  }  presentF \*= height;  }  else  {  count \*= 2;  nextF = 0;  height = (b - a) / count;  #pragma omp parallel for reduction(+:nextF)  for (int i = 0; i < count; i++)  {  double x = a + (i \* height) + (height / 2);  nextF += sqrt(1 + 0.5 \* x + log(x) \* log(x)); // ваша функция  }  nextF \*= height;  if (fabs((nextF - presentF)) <= epsilon)  {  std::cout << "Точность: " << epsilon << " знаков" << std::endl;  std::cout << "Общее количество итераций: " << count << std::endl;  std::cout << "Время вычисления значения: " << (omp\_get\_wtime() - timeStart) << " секунды" << std::endl;  break;  }  else  {  presentF = nextF;  }  }  }  } |

Рис. .Результат выполнения

## Задание 4 (Метод трапеций)

*Листинг 4*

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <cmath>  #include <ctime>  void computeIntegral(double a, double b, double epsilon, double timeStart);  int main()  {  double a = 1.0;  double b = 3.0;  double epsilon[] = { 1e-5, 1e-6, 1e-7 };  setlocale(LC\_ALL, "Russian");  for (int i = 0; i < 3; ++i)  {  std::cout << "Метод трапеций:" << std::endl;  double timeStart = omp\_get\_wtime();  computeIntegral(a, b, epsilon[i], timeStart);  std::cout << std::endl;  }  return 0;  }  void computeIntegral(double a, double b, double epsilon, double timeStart)  {  int count = 0;  double presentF = 0, nextF = 0, height = 0;  while (1)  {  if (!count)  {  count = 1;  height = (b - a) / count;  #pragma omp parallel for reduction(+:presentF)  for (int i = 0; i < count; i++)  {  double x1 = a + i \* height;  double x2 = a + (i + 1) \* height;  presentF += (sqrt(1 + 0.5 \* x1 + log(x1) \* log(x1)) + sqrt(1 + 0.5 \* x2 + log(x2) \* log(x2))) / 2;  }  presentF \*= height;  }  else  {  count \*= 2;  nextF = 0;  height = (b - a) / count;  #pragma omp parallel for reduction(+:nextF)  for (int i = 0; i < count; i++)  {  double x1 = a + i \* height;  double x2 = a + (i + 1) \* height;  nextF += (sqrt(1 + 0.5 \* x1 + log(x1) \* log(x1)) + sqrt(1 + 0.5 \* x2 + log(x2) \* log(x2))) / 2;  }  nextF \*= height;  if (fabs((nextF - presentF)) <= epsilon)  {  std::cout << "Точность: " << epsilon << " знаков" << std::endl;  std::cout << "Общее количество итераций: " << count << std::endl;  std::cout << "Время вычисления значения: " << (omp\_get\_wtime() - timeStart) << " секунды" << std::endl;  break;  }  else  {  presentF = nextF;  }  }  }  } |

  
Рис. .Результат выполнения

## Задание 5 (Метод Симпсона)

*Листинг 5*

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <cmath>  #include <ctime>  void computeIntegral(double a, double b, double epsilon, double timeStart);  int main()  {  double a = 1.0;  double b = 3.0;  double epsilon[] = { 1e-5, 1e-6, 1e-7 };  setlocale(LC\_ALL, "Russian");  for (int i = 0; i < 3; ++i)  {  std::cout << "Метод Симпсона:" << std::endl;  double timeStart = omp\_get\_wtime();  computeIntegral(a, b, epsilon[i], timeStart);  std::cout << std::endl;  }  return 0;  }  void computeIntegral(double a, double b, double epsilon, double timeStart)  {  int count = 0;  double presentF = 0, nextF = 0, height = 0;  while (1)  {  if (!count)  {  count = 1;  height = (b - a) / count;  #pragma omp parallel for reduction(+:presentF)  for (int i = 0; i < count; i++)  {  double x1 = a + i \* height;  double x2 = a + (i + 1) \* height;  double x3 = (x1 + x2) / 2;  presentF += (sqrt(1 + 0.5 \* x1 + log(x1) \* log(x1)) + 4 \* sqrt(1 + 0.5 \* x3 + log(x3) \* log(x3)) + sqrt(1 + 0.5 \* x2 + log(x2) \* log(x2))) / 6;  }  presentF \*= height;  }  else  {  count \*= 2;  nextF = 0;  height = (b - a) / count;  #pragma omp parallel for reduction(+:nextF)  for (int i = 0; i < count; i++)  {  double x1 = a + i \* height;  double x2 = a + (i + 1) \* height;  double x3 = (x1 + x2) / 2;  nextF += (sqrt(1 + 0.5 \* x1 + log(x1) \* log(x1)) + 4 \* sqrt(1 + 0.5 \* x3 + log(x3) \* log(x3)) + sqrt(1 + 0.5 \* x2 + log(x2) \* log(x2))) / 6;  }  nextF \*= height;  if (fabs((nextF - presentF)) <= epsilon)  {  std::cout << "Точность: " << epsilon << " знаков" << std::endl;  std::cout << "Общее количество итераций: " << count << std::endl;  std::cout << "Время вычисления значения: " << (omp\_get\_wtime() - timeStart) << " секунды" << std::endl;  break;  }  else  {  presentF = nextF;  }  }  }  } |

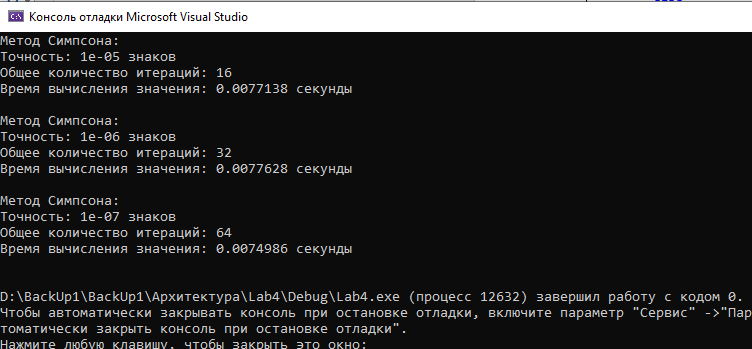


Рис. . Результат выполнения

# Контрольные вопросы

1. **Дополнительные этапы при разработке параллельных программ:**

Определение параллелизма

Выявление параллелизма

Выражение параллелизма

1. **Влияние зависимостей между подзадачами на разработку:**

Требуется учет синхронизации и взаимодействия

Изменения в структуре алгоритма и кода

1. **Пример обозначений для параллельных алгоритмов:**

Директивы OpenMP, например, #pragma omp parallel

1. **Постановка задачи численного интегрирования:**

Приближенное вычисление определенного интеграла

1. **Особенности задачи численного интегрирования для параллелизации:**

Независимые вычисления на подинтервалах

1. **Разбиение задачи численного интегрирования на подзадачи:**

Разделение интервала интегрирования на подинтервалы

1. **Связи между подзадачами и OpenMP:**

Зависимости могут быть учтены с использованием директив #pragma omp barrier

1. **Опции директивы параллельного цикла в OpenMP:**

private и reduction для изоляции и объединения результатов

1. **Достижение точности в численном интегрировании:**

Увеличение числа шагов разбиения

1. **Влияние количества шагов на точность:**

Увеличение шагов улучшает точность

1. **Зависимость времени интегрирования от порядка формулы:**

Высший порядок может требовать больше ресурсов, но дает более точные результаты.