МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ

КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ордена Трудового Красного Знамени федеральное государственное

бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Информатика»

Лабораторная работа №8

**«Разработка параллельных программ с применением технологии OpenMP»**

по дисциплине

**Информационные технологии и программирование**

|  |  |
| --- | --- |
|  | Выполнил: студент гр. БЭИ2202  Бояркин Е. В.  Вариант №3  Проверил: зав. кафедры «Информатика» Халабия Р. Ф. |

Москва, 2024 г.

Содержание

[1 Постановка задачи 3](#_Toc184165327)

[2 Алгоритм решения задачи 5](#_Toc184165328)

[3 Результаты работы программы и их сравнение 6](#_Toc184165329)

[4 Листинги программы 7](#_Toc184165330)

[6 Вывод 10](#_Toc184165331)

# 1 Постановка задачи

Цель задачи: получить навыки распараллеливания существующих последовательных программ на языке C++. Оценить производительность параллельной программы при использовании разного количества ядер процессора.

Задание:

Провести эксперимент по анализу ускорения вычислений на многоядерном процессоре. Работу провести в следующей последовательности.

1. Составить и отладить программу, решающую задачу, на языке С++ в обычном, последовательном режиме вычислений. Доказать, что программа выполняется верно.
2. Продумать декомпозицию задачи (функциональную или по данным).
3. Продумать алгоритм распараллеливания и коммуникации.
4. Найти функции библиотеки параллельного программирования, наиболее подходящие к выбранному алгоритму распараллеливания задачи.
5. Отладить программу, работающую в параллельном режиме.
6. Добавить в программы функции оценки времени выполнения из библиотеки OpenMP.
7. Увеличить объем вычислений в задачах (количество сигналов, размер массивов, временную сложность вычисления функций). Функцию можно задать как сумму других функций, например разложить синус рядом Тейлора, где n = 1000.
8. Провести несколько экспериментов по оценке времени выполнения задачи с различным количеством потоков.
9. Проанализировать и оценить результаты численных экспериментов.
10. Сделать выводы о влиянии на результат различных факторов (архитектуры вычислительной системы, эффективности программы, выбранных алгоритмов, сложности вычислений, времени коммуникаций).

Вариант 3. Разработайте программу методом параллельного программирования для задачи вычисления определенного интеграла с использованием метода прямоугольников.

# 2 Алгоритм решения задачи

Шаг 1. Подготовить пределы интегрирования

Шаг 2. Определить количество разбиений интервала

Шаг 3. Создать функцию для вычисления значения в точке

Шаг 4. Настроить количество потоков для параллельных вычислений

Шаг 5. Зафиксировать начальное время выполнения

Шаг 6. Разделить интервал интегрирования на равные части

Шаг 7. Для каждой части интервала выполнить следующие действия:

Шаг 7.1. Вычислить точку внутри текущей части

Шаг 7.2. Применить созданную функцию к этой точке

Шаг 7.3. Добавить полученное значение к общей сумме

Шаг 8. Умножить общую сумму на размер одной части интервала, чтобы получить приблизительное значение интеграла

Шаг 9. Зафиксировать конечное время выполнения

Шаг 10. Рассчитать общее затраченное время

Шаг 11. Вывести рассчитанное значение интеграла

Шаг 12. Вывести затраченное время на выполнение программы

# 3 Результаты работы программы и их сравнение

Тесты проводились на усложнённой программе на трёх машинах:

1. AMD Ryzen 5 3600X 4.4GHz 6 ядер, 12 потоков, 32GB DDR4 RAM
2. INTEL(R) XEON(R) GOLD 6548Y+ 2.5–4.1GHz 64 ядра, 128 потоков, 128GB DDR5 RAM
3. 2 AMD EPYC 7742 2.25–3.4GHz 64 ядра, 128 потоков (суммарно 128 ядер, 256 потоков), 2TB DDR4 RAM

Для всех машин тесты проводились с n = 1 000 000 000 при параллельном вычислении и 100 000 000 при однопоточном. Результаты, занесённые в таблицу умножены на соответствующую разницу в n, так как была установлена линейная зависимость между разными n.

Результаты запуска приведены в таблице 1.

| Таблица 1 — Результаты запуска программы | | |
| --- | --- | --- |
| Машина | В одном потоке | Параллельно |
| 1 | *1191,61 секунд* | 203,553 секунд |
| 2 | *3042,18* *секунд* | 60,297 секунд |
| 3 | *3593,32* *секунд* | 35,6199 секунд |

# 4 Листинги программы

Однопоточный вариант:

|  |
| --- |
| и #include <iostream>  #include <functional>  #include <omp.h>  namespace  {  double rectangleMethod(const double a, const double b, const int n, const std::function<double(double)>& func)  {  const double h = (b - a) / n;  double sum = 0.0;  for (int i = 0; i < n; ++i)  {  const double x = a + i \* h;  sum += func(x);  }  return h \* sum;  }  }  int main()  {  constexpr double a = 0.0;  constexpr double b = 1.0;  constexpr int n = 1000000000;  auto func = [](const double x)  {  return x \* x;  };  const double startTime = omp\_get\_wtime();  const double result = rectangleMethod(a, b, n, func);  const double endTime = omp\_get\_wtime();  std::cout << "Integral value: " << result << "\n";  std::cout << "Elapsed time: " << endTime - startTime << " seconds" << "\n";  return 0;  } |

Параллельный вариант:

|  |
| --- |
| #include <iostream>  #include <functional>  #include <omp.h>  namespace  {  double rectangleMethod(const double a, const double b, const int n, const std::function<double(double)>& func)  {  const double h = (b - a) / n;  double sum = 0.0;  #pragma omp parallel for reduction(+:sum)  for (int i = 0; i < n; ++i)  {  const double x = a + i \* h;  sum += func(x);  }  return h \* sum;  }  }  int main()  {  constexpr double a = 0.0;  constexpr double b = 1.0;  constexpr int n = 1000000000;  auto func = [](const double x)  {  return x \* x;  };  omp\_set\_num\_threads(256);  const double startTime = omp\_get\_wtime();  const double result = rectangleMethod(a, b, n, func);  const double endTime = omp\_get\_wtime();  std::cout << "Integral value: " << result << "\n";  std::cout << "Elapsed time: " << endTime - startTime << " seconds" << "\n";  return 0;  } |

Параллельный усложнённый вариант:

|  |
| --- |
| #include <iostream>  #include <functional>  #include <omp.h>  namespace  {  double rectangleMethod(const double a, const double b, const int n, const std::function<double(double)>& func)  {  const double h = (b - a) / n;  double sum = 0.0;  #pragma omp parallel for reduction(+:sum)  for (int i = 0; i < n; ++i)  {  const double x = a + i \* h;  sum += func(x);  }  return h \* sum;  }  }  int main()  {  constexpr double a = 0.0;  constexpr double b = 1.0;  constexpr int n = 1000000000;  auto func = [](const double x)  {  double sum = 0.0;  double term = x;  for (int i = 1; i <= 1000; ++i)  {  sum += term;  term \*= -x \* x / (2 \* i \* (2 \* i + 1));  }  return sum;  };  omp\_set\_num\_threads(256);  const double startTime = omp\_get\_wtime();  const double result = rectangleMethod(a, b, n, func);  const double endTime = omp\_get\_wtime();  std::cout << "Integral value: " << result << "\n";  std::cout << "Elapsed time: " << endTime - startTime << " seconds" << "\n";  return 0;  } |

# 6 Вывод

В ходе выполнения задания была разработана программа на языке C++, демонстрирующая разницу между запуском операций в одном потоке и параллельно в нескольких.