###### **МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

###### **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

###### **НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

###### **Факультет информационных технологий**

**Кафедра параллельных вычислений**

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ**

**Практическая работа №2**

Определение времени работы прикладных программ

студента 2 курса, группы 23201

Сорокина Матвея Павловича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.С. Матвеев

Новосибирск 2024

**СОДЕРЖАНИЕ**

[**ЦЕЛЬ** 3](#_Toc174976371)

[**ЗАДАНИЕ** 3](#_Toc174976372)

[**ОПИСАНИЕ РАБОТЫ** 4](#_Toc174976373)

[**ЗАКЛЮЧЕНИЕ** 4](#_Toc174976374)

[**ПРИЛОЖЕНИЯ** 5](#_Toc174976375)

# **ЦЕЛЬ**

1. Изучение основных функций оптимизирующего компилятора, некоторых примеров оптимизирующих преобразований и уровней оптимизации.
2. Получение базовых навыков работы с компилятором GCC.
3. Анализ влияния различных уровней оптимизации компилятора GCC на время выполнения программы.

# **ЗАДАНИЕ**

1. Написать программу на языке C или C++, которая реализует выбранный алгоритм из задания.
2. Проверить правильность работы программы на нескольких тестовых наборах входных данных.
3. Выбрать значение параметра N таким, чтобы время работы программы было порядка 30-60 секунд.
4. Программу скомпилировать компилятором GCC с уровнями оптимизации **-O0, -O1, -O2, -O3, -Os, -Ofast, -Og** под архитектуру процессора x86.
5. Для каждого из семи вариантов компиляции измерить время работы программы при нескольких значениях N.
6. Составить отчет по лабораторной работе.

# **ОПИСАНИЕ РАБОТЫ**

1. Реализовано задание 5 - алгоритм вычисления функции e^(x) с помощью разложения в ряд Маклорена по первым N членам данного ряда на языке С++.

Для измерения времени работы программы использовалась библиотечная функция clock\_gettime из библиотеки time.h. Время замерялось перед началом и после окончания работы функции, вычисляющей синус заданного угла.

Разность этих двух значений дает общее время выполнения функции. Для проверки точности измерений, код программы запускается несколько раз.

1. Код программы предоставлен (см. Приложение 1), также предоставлен bash-скрипт, компилирующий и запускающий программу с конкретными знаениями x и N, записывающий время вычисления экспоненциальной функции с различными уровнями оптимизации компилятора GCC в report.csv файл (см. Приложение 2), после чего запускает create\_table.sh, визуализирующий информацию из report.csv (см. Приложение 3).

# **ЗАКЛЮЧЕНИЕ**

В ходе данной лабораторной работы мы познакомились с различными методами измерения работы программ и научились пользоваться ими на практике.

В ходе лабораторной работы было установлено, что уровни оптимизации компилятора GCC оказывают влияние на время выполнения программы. Было замечено, что с увеличением уровня оптимизации время выполнения программы уменьшается.

Стоит обратить внимание на то, что наиболее распространенным ключом оптимизации является **-O2**, поскольку:

1. Ключ активирует большинство оптимизаций, которые улучшают производительность и уменьшают размер кода, не добавляя экстремальных и потенциально нестабильных оптимизаций.
2. Он улучшает скорость выполнения программы, при этом сохраняя её стабильность.

# **ПРИЛОЖЕНИЯ**

**Приложение 1:Исходный код программы ExponentCalculation.cpp**

|  |
| --- |
| #include <iostream>  #include <time.h>  #include <cstdlib> // for atoi and atof  #include <cmath> // for pow  long double ExpCalculation(long double x, long long n) {      long double exp = 1.0;      long double term = 1.0;      for (long long i = 1; i < n; i++) {          term \*= x / i;          exp += term;      }      return exp;  }  int main(int argc, char \*argv[]) {      if (argc != 3) {          std::cerr << "Usage: <degree> <number of terms>" << std::endl;          return 0;      }      struct timespec start, end;      long double x = atoll(argv[1]);      long long n = atoll(argv[2]);      std::cout << "x = " << x << ", n = " << n << std::endl;      int runs = 3;      double time\_total = 0;      for (long long i = 0; i < runs; i++) {          clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);          long double exp = ExpCalculation(x, n);          clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);          double taken\_time = (end.tv\_sec - start.tv\_sec) + (end.tv\_nsec - start.tv\_nsec) / 1e9;          std::cout << "e^(" << x << ") = " << exp << std::endl;          std::cout << "Run №" << i + 1 << " took " << taken\_time << " seconds to complete" << "\n" << std::endl;          time\_total += taken\_time;      }      std::cout << "Average time: " << time\_total / runs << " seconds" << std::endl;      return 0;  } |

**Приложение 2: bash-скрипт для компиляции и запуска программы, записи результата в файл report.csv – compile\_and\_run.sh**

|  |
| --- |
| #!/bin/bash  input="ExponentCalculation.cpp"  optimization\_keys=("-O0" "-O1" "-O2" "-O3" "-Os" "-Ofast" "-Og")  n=("4000000000" "4500000000" "5000000000")  x=90  echo "Optimixation level, N value, Time taken (seconds)" > report.csv  for i in "${optimization\_keys[@]}"; do      for j in "${n[@]}"; do          g++ $input $i -std=c++11          output=$(./a.out $x $j | grep "Average time:" | awk '{print $3}')          echo "$i, $j, $output" >> report.csv      done  done  echo "Successfully generated report.csv" |

**Приложение 3: bash-скрипт для вывода в табличном формате данных из report.csv - create\_table.sh**

|  |
| --- |
| #!/bin/bash  if [ ! -f report.csv ]; then      echo "report.csv not found!"      exit 1  fi  # Вывод заголовка таблицы  echo -e "Оптимизация\t N\t\t Время выполнения (сек)"  # -F',': запятая в качестве разделителя столбцов  # NR>1: пропускаем первую строку  awk -F',' 'NR>1 { printf "%s\t\t%s\t%s\t%s\n", $1, $2, $3, $4 }' report.csv |

**Приложение 4: Таблица с результатами измерений**

|  |  |  |
| --- | --- | --- |
| Оптимизация | Значение N | Время выполнения (с) |
| -O0 | 4000000000 | 35.1887 |
| -O0 | 4500000000 | 39.8341 |
| -O0 | 5000000000 | 43.9038 |
| -O1 | 4000000000 | 8.04768 |
| -O1 | 4500000000 | 9.01514 |
| -O1 | 5000000000 | 9.90664 |
| -O2 | 4000000000 | 7.84239 |
| -O2 | 4500000000 | 8.87028 |
| -O2 | 5000000000 | 9.99184 |
| -O3 | 4000000000 | 7.96013 |
| -O3 | 4500000000 | 8.82066 |
| -O3 | 5000000000 | 9.77772 |
| -Os | 40000000000 | 8.39481 |
| -Os | 45000000000 | 9.47931 |
| -Os | 50000000000 | 10.3841 |
| -Ofast | 40000000000 | 7.8265 |
| -Ofast | 45000000000 | 8.91898 |
| -Ofast | 50000000000 | 10.0095 |
| -Og | 40000000000 | 7.99961 |
| -Og | 45000000000 | 8.88734 |
| -Og | 50000000000 | 9.85844 |

**Приложение 5: график зависимости времени от оптимизации компиляции и параметра N для разных уровней оптимизации**

***A graph with a red line

Description automatically generated***

**Приложение 6: Скрипт для визуализации графика зависимости времени от зависимости компиляции на языке Python**

|  |
| --- |
| import matplotlib.pyplot as plt  n\_values = [4000000000, 4500000000, 5000000000]  times\_o0 = [35.1887, 39.8341, 43.9038]  times\_o1 = [8.04768, 9.01514, 9.90664]  times\_o2 = [7.84239, 8.87028, 9.99184]  times\_o3 = [7.96013, 8.82066, 9.77772]  times\_os = [8.39481, 9.47931, 10.3841]  times\_ofast = [7.8265, 8.91898, 10.0095]  times\_og = [7.99961, 8.88734, 9.85844]  plt.figure(figsize=(12, 6))  plt.plot(n\_values, times\_o0, 'r-o', label='-O0')  plt.plot(n\_values, times\_o1, 'g-o', label='-O1')  plt.plot(n\_values, times\_o2, 'b-o', label='-O2')  plt.plot(n\_values, times\_o3, 'y-o', label='-O3')  plt.plot(n\_values, times\_os, 'c-o', label='-Os')  plt.plot(n\_values, times\_ofast, 'm-o', label='-Ofast')  plt.plot(n\_values, times\_og, 'k-o', label='-Og')  plt.xlabel('N value')  plt.ylabel('Time taken (seconds)')  plt.legend()  plt.grid(True)  plt.show() |