УДК 004.432.2

ВЛИЯНИЕ ОПТИМИЗАЦИИ КОМПИЛЯТОРА GCC НА ЭФФЕКТИВНОСТЬ ПРОГРАММНОГО КОДА В ЯЗЫКЕ C++

Болотнов А.М., Нурисламова Э.А.

ФГОУ ВО «Башкирский государственный университет», Уфа, e-mail: BolotnovAM@mail.ru

Эффективность программного кода является важнейшей характеристикой языков программирования и компиляторов. При выполнении вычислительных алгоритмов язык C++ является одним из наиболее эффективных. Компиляторы GNU Compiler Collection являются наиболее доступными и эффективными инструментами генерации исполняемого кода для основных языков программирования. В данной работе проведено исследование влияния различных уровней оптимизации компилятора GCC на эффективность работы программ на языке C++. Тестирование проводилось на примерах шести стандартных алгоритмов, используемых в практике программирования. Для получения объективной оценки времени выполнения заданий были произведены расчеты на трех компьютерах с различной платформой. Цель данной работы заключалась в получении средних оценок влияния встроенных опций оптимизации компилятора GCC на время выполнения программ с различными алгоритмическими особенностями. Установлено, что время работы программ существенно зависит от используемых действительных типов. Тестирование проводилось по двум основным действительным типам: двойная точность и расширенная точность. Тип одинарная точность не был протестирован из-за того, что в вычислительных задачах использование этого типа ограничено малой длиной мантиссы. В работе получены средние оценки повышения эффективности программного кода при различных уровнях оптимизации компилятора. Показано, что максимальный эффект от применения оптимизации компилятора достигается для вещественных типов двойной точности. Для всех алгоритмов тестирования максимальное ускорение достигается при выборе ключа оптимизации -Ofast.

Ключевые слова: язык программирования C++, компиляторы GCC, свободное программное обеспечение, оптимизация программ, эффективность программного кода

IMPACT OF GCC COMPILER OPTIMIZATION ON SOFTWARE EFFICIENCY IN THE LANGUAGE C++

Bolotnov A.M., Nurislamova E.A.

Bashkir State University, Ufa, e-mail: BolotnovAM@mail.ru

Software efficiency is an essential feature of programming languages and compilers. The C++ language is one of the most computationally efficient for computational algorithms. GNU Compiler Collection is the most accessible and effective tool for generating executable code for core programming languages. In this paper, a study has been conducted on the impact of different levels of GCC compiler optimization on the effectiveness of programs written in the C++ language. Testing was based on the examples of six standard algorithms used in programming practice. In order to obtain an objective estimate of the time of the tasks, calculations were made on three different computing platforms. The objective of this work was to obtain average estimates of the impact of standard GCC compiler optimization options on the codes with different algorithmic features. It has been established that the code execution time is significantly dependent on the floating point variable types used. It has been established that the code execution time is significantly dependent on the floating point types used. Testing was conducted on two main types: double precision and enhanced precision. The single precision type has not been tested due to the fact that in computational tasks the use of this type is limited by low precision. The work includes average estimates of the effectiveness of code at different levels of compiler optimization. It is shown that the maximum effect of the use of compiler optimization is achieved for the real types of double precision. For all test algorithms, maximum acceleration is achieved when selecting the optimization key -Ofast.

Keywords: C++ programming language, GCC compilers, free software, program optimization, program code efficiency

В настоящее время при создании программных проектов используется большое количество интегрированных сред разработки приложений, компиляторов и языков программирования. Коллекция компиляторов GNU Compiler Collection (GCC) [1], распространяемая фондом свободного программного обеспечения на условиях лицензий GNU GPL и GNU LGPL, является наиболее доступным и универсальным инструментом генерации исполняемого кода для основных языков программирования: C, C++, Java, Fortran, Ada, Go, D. Отличительной особенностью коллекции компиляторов GCC является поддержка многих операционных систем и аппаратных платформ.

Эффективность генерируемого программного кода, в первую очередь – скорость выполнения проекта, является важнейшей характеристикой языков программирования и компиляторов. Одним из востребованных, универсальных и эффективных языков программирования, в том числе для реализации вычислительных алгоритмов, остается язык C++ [2, 3].

Многие современные компиляторы, в том числе из коллекции GCC, содержат встроенные средства оптимизации исходного кода программы, направленные в том числе на повышение скорости работы приложений. Для объективной оценки влияния различных уровней оптимизации компиля-

тора на повышение эффективности исполняемого кода необходим сравнительный анализ результатов работы нескольких программ, отличающихся алгоритмическими особенностями и протестированных на компьютерах с различными платформами.

Цель исследования в данной работе состоит в получении усредненных оценок повышения эффективности программного кода для различных уровней оптимизации компилятора MinGW-64 GCC [4] при выполнении на трех компьютерах с различной платформой шести тестовых алгоритмов, реализованных на языке C++. Так как время работы программных приложений существенно зависит от используемых базовых действительных типов (одинарная, двойная, расширенная точность), тестирование проводилось для двух основных типов: с двойной точностью (double) и с расширенной точностью (long double). Действительный тип с одинарной точностью (float) не тестировался ввиду того, что в вычислительных задачах применение указанного типа ограничено недостаточно широким диапазоном представления чисел и малой длиной мантиссы.

Материалы и методы исследования

Компилятор для языка программирования C++ из коллекции GCC допускает применение следующих уровней оптимизации, в порядке увеличения скорости работы программы: *-O*1, *-O*2, *-O*3 и *-Ofast*. Информацию о флагах оптимизации компилятора GCC можно получить с помощью команды man gcc. Конкретный перечень оптимизирующих процедур по каждому из перечисленных уровней оптимизации приведен в [1, 4]. Отметим, что включение опций оптимизации несколько увеличивает время компиляции программ; их использование может оказаться нецелесообразным в проектах, выполняющихся за короткое время.

Результаты ускорения выполнения программ с использованием ключей оптимизации во многом зависят от используемой платформы, поэтому в данном исследовании тестовые расчеты проводились независимо на трех персональных компьютерах с различными характеристиками: ПК1 – IBM PC 32 bit X86 Intel CPU 2.8 ГГц RAM 2 Гб; ПК2 – Note Book 64 bit AMD Phenom(tm) II Quad-Core 2.0 ГГц RAM 4 Гб; ПК3 – IBM PC 64 bit Intel(R) Core(TM) i5-3470 3.2 ГГц RAM 8 Гб.

Введем следующие обозначения. Пусть

i – порядковый номер задачи (1, 2, …, 6);

j – порядковый номер ПК (1, 2, 3);

m – тип данных (1 – double, 2 – long double);

n – уровень оптимизации (0 – без оптимизации; 1 – *O*1; 2 – *O*2; 3 – *O*3; 4 – *Ofast*).

Тогда t_{ijmn} – время выполнения i-й задачи на j-м компьютере для m-го действительного типа данных с n-м уровнем оптимизации компилятора. Для объективности оценки времени выполнения алгоритмов значение параметра t_{ijmn} вычислялось как среднее арифметическое из 10 повторных расчетов, после чего фиксировалось время выполнения i-й задачи (в секундах): t_{i1mn} – для первого ПК, t_{i2mn} – для второго, и t_{i3mn} – для третьего.

Очевидно, что средний коэффициент ускорения выполнения i-й задачи на трех ПК может быть вычислен по формуле

$$K_{imn} = \frac{1}{3}\sum_{j=1}^{3} t_{ijm0} / t_{ijmn};\ i = 1, 2, \ldots, 6;$$

$$m = 1, 2;\ n = 0, 1, 2, 3, 4, \qquad (1)$$

где t_{ijm0} – время работы алгоритма без оптимизации.

Принятые обозначения использованы в представленных ниже результатах проведенных расчетов. Тестирование проводилось на примерах типичных вычислительных алгоритмов, которые широко встречаются в практике программирования. Все полученные значения приведены с округлением до трех значащих цифр.

Результаты исследования и их обсуждение

Тест-1: итерационный алгоритм. В программе реализован метод последовательных приближений решения краевой задачи для уравнения Пуассона с нелинейными граничными условиями [5, 6]. Особенностью данного алгоритма является отсутствие массивов; задействованы функции пользователя с многократными вызовами; число итераций фиксировано.

Тест-2: сортировка одномерного массива. В качестве теста используется простейший алгоритм обменами (метод пузырька) для упорядочивания одномерного массива с 200000 элементами действительного типа двойной и расширенной точности.

На рис. 1 представлены результаты, полученные в первых двух тестах.

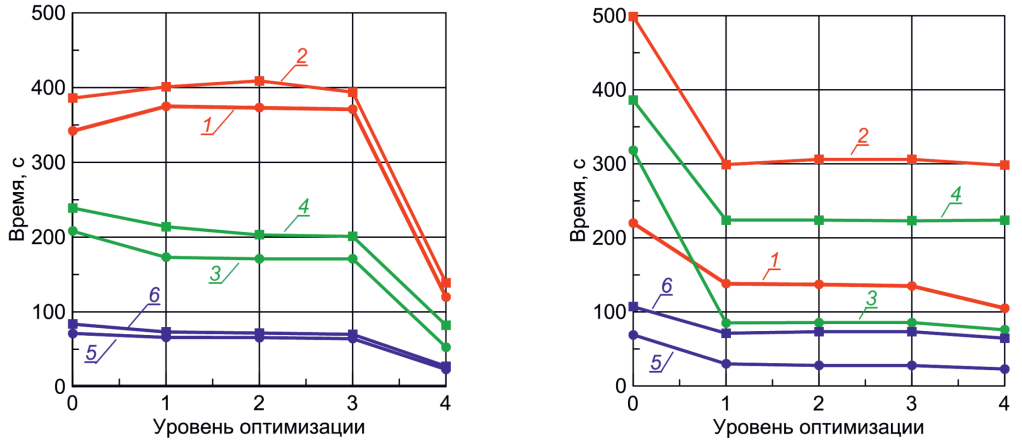
Как видно из рис. 1, в первом тесте применение оптимизации компилятора с первого по третий уровни не дают заметного эффекта и только при четвертом уровне (*-Ofast*) среднее время выполнения программы сокращается более чем в 2,5 раза. Для второго алгоритма ситуация кардинально отличается: здесь уже при первом уровне оптимизации значение среднего коэффициента ускорения превышает 2,5 для типа double и до-

стигает 1,7 для типа long double. Включение же второго и третьего уровней оптимизации не приводит к заметным изменениям по сравнению с первым уровнем. Эффект от применения оптимизации четвертого уровня наиболее значителен для типа double (K = 3,1); для типа long double аналогичный показатель гораздо ниже (K = 1,69).

Тест-3: перемножение матриц (вариант А). В данном тесте реализован алгоритм перемножения двух квадратных матриц (1000*1000) с элементами двойной и расширенной точности в его классическом виде «строка – на – столбец».

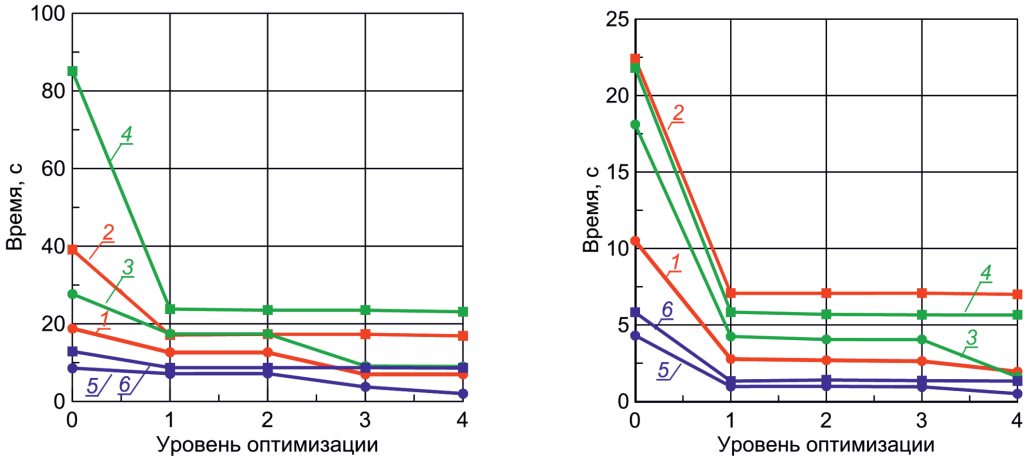
Тест-4: перемножение матриц (вариант Б). Решается та же задача. Отличие от предыдущего алгоритма заключается в том, что в данном случае учитывается расположение элементов двумерных массивов в оперативной памяти компьютера. Компилятор GCC для языка C++ размещает двумерные массивы в памяти ПК по строкам, поэтому в данном примере для ускорения работы алгоритма транспонируется вторая матрица (строки и столбцы меняются местами), после чего матрицы перемножаются по схеме «строка – на – строку».

На рис. 2 представлены полученные результаты.



а) *б)*

Рис. 1. Зависимости времени выполнения программ от уровня оптимизации компилятора в тесте-1 – (а) и в тесте-2 – (б): 1, 2 – ПК1; 3, 4 – ПК2; 5, 6 – ПК3; для базовых действительных типов double – 1, 3, 5 и long double – 2, 4, 6



а) *б)*

Рис. 2. Зависимости времени выполнения программ от уровня оптимизации компилятора в тесте-3 – (а) и в тесте-4 – (б). Остальные обозначения те же, что и в подписи к рис. 1

Сравнивая результаты данных тестов, отметим, что абсолютное время выполнения задачи по второму алгоритму для всех ПК сократилось в 3–6 раз. Отметим особо результаты вычислений на ПК2 для типа double: при четвертом уровне оптимизации (ключ -*Ofast*) время выполнения алгоритма сократилось более чем в 11 раз (18,1 с без оптимизации и 1,6 с с четвертым уровнем оптимизации). При максимальном уровне оптимизации средний коэффициент ускорения превышает значение 3,0 для типа double и 2,5 – для типа long double.

Тест-5: метод Гаусса. Численное решение систем линейных алгебраических уравнений (СЛАУ) является актуальной задачей для многих разделов вычислительной математики. Метод исключения Гаусса – один из наиболее известных и часто используемых прямых методов решения СЛАУ [7]. В данном примере тестируется алгоритм решения системы $A*x = b$ методом Гаусса с выбором ведущего элемента по столбцу. Здесь A – матрица ($N*N$); x и b – векторы из N элементов. Тестирование проводилось для СЛАУ при $N = 1400$.

Тест-6: метод Гивенса. В данном примере решается та же система линейных алгебраических уравнений, что и в предыдущем тесте, но методом вращений (Гивенса). Метод вращений является наиболее устойчивым к вычислительной погрешности [8, 9], хотя и требует больших усилий при реализации алгоритма.

Результаты полученных расчетов для тестов 5, 6 представлены на рис. 3.

Сравнивая результаты тестов 5 и 6, приведенные на рис. 3, отметим, что абсолютное время решения СЛАУ методом Гивенса в среднем в 2–2,5 раз больше, чем решение той же задачи методом Гаусса. В тесте 5 при максимальном уровне оптимизации средний коэффициент ускорения оказался равным 3,44 для типа double, и 1,93 – для типа long double; при этом для типа long double уровни оптимизации 1–3 приводят к близким значениям ускорения (1,67–1,70). В тесте 6 средний коэффициент ускорения при четвертом уровне оптимизации достигает значения 5,2 для базового типа double и 2,7 – для типа long double.

Обобщая проведенные исследования, приведем гистограмму распределения коэффициентов ускорений при выполнении шести тестовых алгоритмов с различными уровнями оптимизации компилятора (рис. 4).

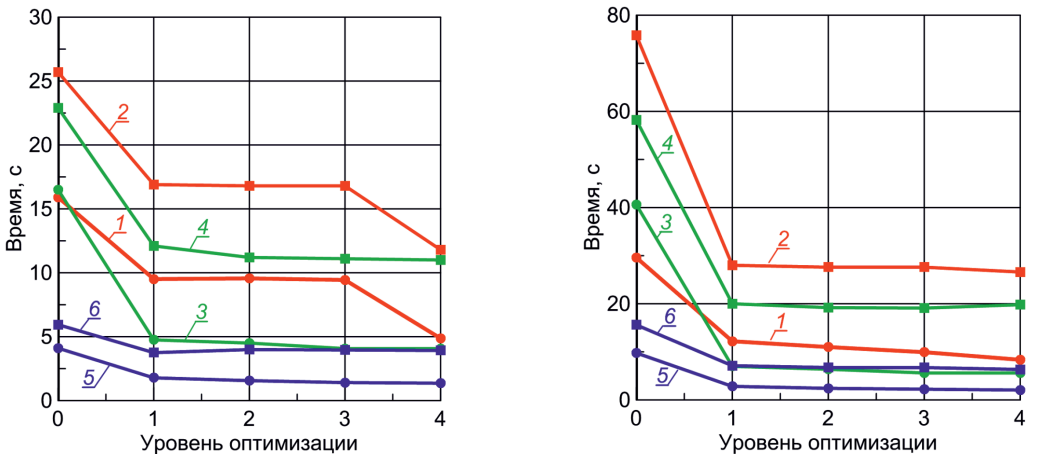
По результатам проведенных расчетов получены итоговые оценки на основании соотношения

$$K^0_{mn} = \frac{1}{6}\sum_{i=1}^{6} K_{imn};\ m = 1, 2;\ n = 0, 1, 2, 3, 4, \quad (2)$$

которые определяют средние значения коэффициентов ускорения в шести тестовых алгоритмах при различных ключах оптимизации компилятора для двух действительных типов. Результаты, полученные по формуле (2), представлены в таблице

Средние значения ускорений

n	0	1	2	3	4
K^0_{1n}	1	2,60	2,73	3,08	4,46
K^0_{2n}	1	2,20	2,21	2,22	2,60



а) *б)*

Рис. 3. Зависимости времени выполнения программ от уровня оптимизации компилятора в тесте-5 – (а) и в тесте-6 – (б). Остальные обозначения те же, что и в подписи к рис. 1

Рис. 4. Коэффициенты ускорения при выполнении программ в шести тестах с выбором ключей оптимизации: 0 – нет оптимизации, 1 – ключ оптимизации -O1, 2 – ключ оптимизации -O2, 3 – ключ оптимизации -O3, 4 – ключ оптимизации -Ofast

Заключение

Применение средств оптимизации программного кода всех уровней, имеющихся в компиляторе GCC, приводит, как правило, к различному ускорению работы приведенных алгоритмов. Максимальный эффект, как и следовало ожидать, наблюдается при выборе ключа оптимизации четвертого уровня (*-Ofast*). Наибольшее среднее значение коэффициента ускорения (4,46) получено в алгоритмах с базовым действительным типом double; среднее значение аналогичного показателя для типа long double достигает значения 2,6.

Список литературы

1. GCC, the GNU Compiler Collection. [Электронный ресурс]. URL: https://gcc.gnu.org/ (дата обращения: 18.11.2019).

2. Бьерн Страуструп. Язык программирования C++. Специальное издание. Пер. с англ. М.: Издательство Бином, 2017. 1136 с.

3. Димитриев А.П. Исследование эффективности методов оптимизации исходного кода на примере программы моделирования расписания занятий // Фундаментальные исследования. 2016. № 5–1. С. 28–32.

4. MinGW, Minimalist GNU for Windows. [Электронный ресурс]. URL: http://www.mingw.org (дата обращения: 18.11.2019).

5. Болотнов А.М. Компьютерное моделирование потенциальных электрических полей в электролитах на основе интервальных вычислений // Современные проблемы науки и образования. 2014. № 2. [Электронный ресурс]. URL: http://science-education.ru/ru/article/view?id=12937 (дата обращения: 18.11.2019).

6. Болотнов А.М., Хисаметдинов Ф.З. Определение сопротивления изоляции трубопровода по результатам измерений разности потенциалов «грунт – труба» // Вестник Башкирского университета. 2017. Т. 22. № 1. С. 20–24.

7. Каханер Д., Моулер К., Нэш С. Численные методы и программное обеспечение: Пер. с англ. М.: Мир, 2001. 575 с.

8. Амосов А.А., Дубинский Ю.А., Копченова Н.В. Вычислительные методы для инженеров: учеб. пособие. М.: Высш. шк., 1994. 544 с.

9. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. 4-е изд. М.: БИНОМ. Лаборатория знаний, 2006. 636 с.