Техника инструментирования кода и оптимизация кодовых строк при моделировании фазовых переходов на языке программирования C++

Е.В. Пальчевский <teelxp@inbox.ru>
А.Р. Халиков <khalikov.albert.r@gmail.com>
Уфимский государственный авиационный технический университет,
450000, Россия, г. Уфа, ул. К. Маркса, 12

Аннотация. В данной статье рассматриваются техника написания кода, с помощью которой можно сэкономить время для написания определенной программы, техника инструментирования кода на языке высокого уровня C++, приведены примеры алгоритмов написания кода для системных программ и рассмотрены оптимизированные компиляторы, за счет которых можно сократить время компиляции и ускорить работу программы. Даны определения языка программирования «C» и «C++», кодовой структуры, кодовой оптимизации. Также рассмотрена статистика использования языков программирования по популяризации. Описаны основные преимущества языка программирования «C++», в сравнении с «C». Произведено обозначение структуры исходного кода для более удобного использования. Приведены примеры реализации кодового алгоритма для дальнейшей работы с кодовыми строками. Приведены примеры внятных и кратких комментариев к написанным исходным строкам. Реализованы классовые конструкторы и деструкторы, целью которых является детальная оптимизированность и уменьшение использования блоковой оперативной памяти при моделировании фазовых переходов. Рассмотрены виды оптимизаций: ручная и автоматизированная. В ручной оптимизации описаны несколько подпунктов: peephole, внутрипроцедурная, локальная, межпроцедурная и оптимизационные циклы. Разработан структурированный кода с последовательной заменой логических выражений с проработанными различными типами данных с последующей оптимизацией потребления оперативной памяти персонального компьютера. Показана работа с определенными функциями и представлены практические примеры использования. Произведены расчеты потребляемой памяти до и после оптимизации кодовых строк, на различных компиляторах и серверном оборудовании, конфигурация которого также приведена в данной статье.

Ключевые слова: код; оптимизация кода; инструментирование кода; техника написания кода; C++; программирование

DOI: 10.15514/ISPRAS-2015-27(6)-6

Для цитирования: Пальчевский Е.В., Халиков А.Р. Техника инструментирования кода и оптимизация кодовых строк при моделировании фазовых переходов на языке программирования C++. Труды ИСП РАН, том 27, вып. 6, 2015 г., стр. 87-96. DOI: 10.15514/ISPRAS-2015-27(6)-6.

1. Введение

Не является секретом тот факт, что техника инструментирования кода – это одна из самых важных составляющих частей, которая позволяет выполнить поставленную задачу максимально качественно.

Актуальность выбранной темы подтверждается тем, что сфера программирования востребована, а язык *C++* занимает одну из лидирующих позиций, что подтверждает диаграмма, отображенная на рис.1.

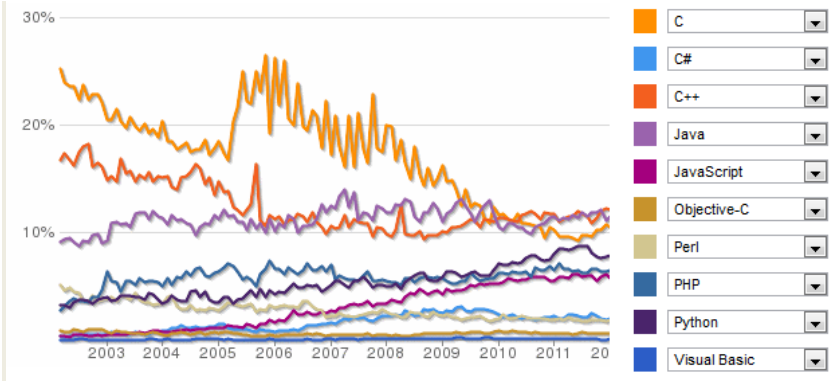

Рис. 1. Статистика языков программирования в 2003-2012 годах [1]

C++ - это язык программирования высокого уровня, который является компилируемым и предназначен для решения задач различного уровня. Этот язык очень широко используется в разработке программного обеспечения и является одним из самых популярных языков программирования. Область разработки включает в себя как написание консольных программ с выводом обычных сообщений, так и разработка крупных проектов, к примеру, операционная система *Windows*. Методики программирования на данном языке – это своеобразный структурированный алгоритм, за счет которого можно как ускорить работу кода, так и грамотно его разделить по частям, тем самым создавая удобства при программировании какого-либо проекта [2].

Если брать в сравнение с языком «*C*», то можно отметить следующие основные нововведения:

- в данном языке программирования, включена поддержка объектно-ориентированного программирования через классы, что позволяет строить абстрактные типа данных, к примеру, структуры и перечисления, интерфейсы, метаклассы, подпроцессы и характеризовать их своими функциями и особенностями;
- имеется поддержка программирования через шаблоны функций и классов, если называть иначе, то «обобщённое программирование», которое дает возможность описания алгоритмов и данных так, что можно не менять самого описания и есть возможность применить это описание к различным типам данных;
- появилась возможность обрабатывать исключения, что позволяет описывать реакцию программы на ошибки времени выполнения и другие проблемы, называемые «исключениями», появление которых может возникнуть при компиляции программы и могут привести к невозможной ее дальнейшей отладке;
- виртуальная функция или виртуальный метод. Используется для переопределения в классах-наследниках для того, чтобы вызов для конкретной реализации метода будет определен во время исполнения самой программы;
- встраиваемые функции «*inline*», которые используются для ускорения программы. Сам вызов функции занимает намного больше времени, чем код, который будет без написания данных функций. И если использовать встроенные функции для замены вызовов, то компиляция программы будет ускорена в 5-6 раз;
- перегрузка имен функций, которая позволяет определять множество функций с одинаковым именем и с точным типом возвращаемого значения.

2. Техника написания кода на языке программирования C++

Для того чтобы код функционировал правильно – нужно тщательно продумывать логику будущей программы.

Нужно разработать алгоритм программы, за счет которого будет написан будущий код. Грамотный подход составления алгоритма программы – это написание будущей программы псевдокодом, который использует ключевые слова языков программирования и опускает специфический синтаксис.

2.1 Обозначение структуры кода

Кодовая структура – это неотъемлемая составляющая всего кода, за счет которой достигается «понимание» программы [3].

Работать с определенными функциями очень удобно, так как есть разделяющие компоненты связанности, что упрощает структурированность всей программы. Но более удобно работать с такими вещами, как публичные классы с определенной точностью символов и отличающимися названиями. В качестве примера можно привести два публичных класса с конструкторами и деструкторами [4]:

- *class UfaClass*: *public ParentClass* // Название класса «*UfaClass*», в котором имеется публичный родительский класс «*ParentClass*».

```
  {
    public:
  UfaClass(); // конструктор
  ~UfaClass(); // деструктор
protected:
  // Доступ элементов секции из родительского класса.
private:
  // элементы доступны по умолчанию из класса.
};
```

и второй точно такой же публичный класс, с точно таким же названием:

- *class UfaClass*: *public ParentClass* // Название класса «*UfaClass*», в котором имеется публичный родительский класс «*ParentClass*».

```
  {
    public:
  UfaClass(); // конструктор
  ~UfaClass(); // деструктор
protected:
  // Доступ элементов секции из родительского класса.
private:
  // элементы доступны по умолчанию из класса.
};
```

Несмотря на комментарии в самом коде, можно запутаться даже в самой небольшой структуре кода программы. Именно поэтому рекомендуется использовать разные заголовки классов и публичных функций [5].

2.2 Внятные и краткие комментарии к написанному коду

Для более быстрой ориентации в коде, нужно оставлять комментарии после каждого написанного класса или функции, что показано в следующем примере, в котором рассмотрено выравнивание данных в оперативной памяти:

- *struct M*{ //
  char m; // целочисленный (символьный) тип данных
  long int p; // целочисленный тип данных «*p*»
  int n;
  };

3. Оптимизация кода на языке С++

Кодовая оптимизация – это преобразования кода за счет различных методов с целью улучшения различных атрибутов и характеристик. Основными целями кодовой оптимизации являются три составляющие:

- ускорение компиляции кода;
- ускорение работы самой программы;
- уменьшение объема кода;
- уменьшение потребления системных ресурсов.

Кодовая оптимизация может производить двумя способами:

- ручной способ. Программист сам отыскивает нужные участки кода и производит оптимизацию;
- автоматизированный способ, в котором оптимизацию выполняют «оптимизирующие компиляторы». В качестве примера можно привести компилятор «*VTune Amplifier*», который умеет анализировать код полностью и показывать время компиляции того или иного участка кода.

Пример скриншота программы приведен на рис.3.

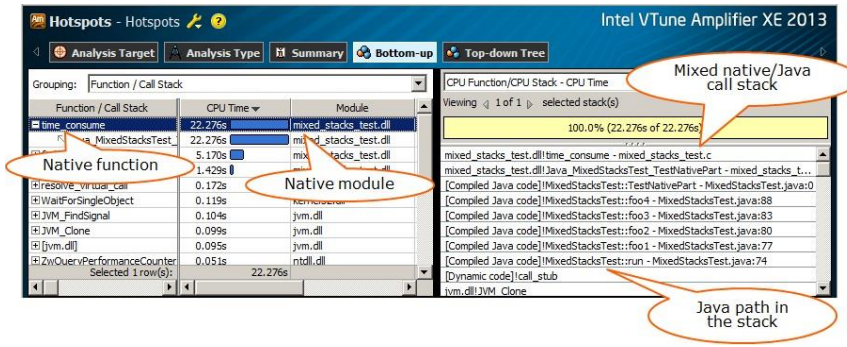

Рис. 3. Пример работы программы с компилятором «VTune Amplifier» [6].

Существуют несколько типов ручной оптимизации:

- *peephole*-оптимизация, суть которой состоит в том, что идет рассмотрение «инструкций», чтобы сделать вывод: можно ли сделать с ними какую-нибудь трансформацию для кодовой оптимизации [7];
- внутрипроцедурная оптимизация – это глобальная оптимизация, которая выполняется целиком и только в рамках одной единицы трансляции: функции. Данная оптимизация эффективнее, к примеру, чем локальная и позволяет достигать оптимизационных эффектов в разы больше, за счет больше затребованной информации и системных ресурсов, которые понадобятся для различных вычислений;
- локальная оптимизация охватывает рассмотрение информационный базовый блог, строго за один шаг. В самих базовых блоках нет переходов к потокам управления, что значит следующее: данная оптимизация экономит время и оперативную память, но теряет информацию к следующему шагу оптимизации;
- оптимизационные циклы (оптимизация циклов) – это одна из самых важных оптимизаций в самой программе, так как большие количество циклов замедляет как работу программы, так и компиляцию кода. Если объединять несколько циклов в один, то компилятору будет легче компилировать, за счет уменьшения вызовов функций и объема кода;
- межпроцедурная оптимизация. С помощью данного метода оптимизации можно анализировать сразу весь код, без определенного поиска нужных блоковых частей. Встраивание копии тела функций, с помощью данного метода, позволяет сэкономить системные ресурсы, которые связаны именно с вызовом функций.

Можно привести пример «выравнивания данных» с оптимизационными методами. Выравнивание данных – это способ выравнивания данных в оперативной памяти вычислительной техники (компьютера), который помогает разместить данные таким образом, что доступ к ним будет ускорен в несколько раз.

В качестве первого кода, возьмем структуру «*one*» и зададим в ней следующие параметры:

- *struct one*{
  char m; // целочисленный (символьный) тип данных
  long int p; // целочисленный тип данных «*p*»
  int n;
  };

данная структура занимает 24 байта. Узнать это можно с помощью вывода «*sizeof(A)*». А если логически поменять местами целочисленный тип данных «*p*» с «*n*», то получится следующая структура «*two*»:

- *struct two*{

char m; // целочисленный (символьный) тип данных
int n;
long int p; // целочисленный тип данных «*p*»
};

Таким образом, структура будет занимать уже 16 байт, а не 24. Связанно это с тем, что на 64-х битной операционной системе, считывание памяти происходит участками, которые равны 8 байтам. Пример считывания для данной программы:

Под первую структуру подсчет памяти будет вестись следующим образом:

1 байт выдается под структуру «*one*», остальные 7 байт остаются пустыми;

нужно добавить 8 байт под целочисленный тип данных «*p*»;
и под «*n*» выделяется 4 байта и 4 остаются пустыми.
(1+7) + 8 + (4+4) = 24 байта, что не совсем рационально.

Под вторую структуру подсчет будет вестись другим образом:

1 байт выдается под структуру «*two*», остальные 3 байта – пустые и нужно прибавить 4 байта под тип данных «*n*». Сокращение произошло за счет изменения структуры целочисленных типов данных;

Выделение 8 байт происходит для целочисленного типа данных «*p*».
(1+3+4) + 8 = 16 байт, что выглядит более рационально.

Данная оптимизация напрямую влияет на скорость компиляции кода. Результаты тестирования приведены в таблице 1.

Табл. 1 – Тестирование кодовых структур на скорость компиляции

Кодовая структура	Общая скорость компиляции, мс.	Параметр «*debug*», мс.	Параметр «*realese*», мс.
«*one*»	14	8	6
«*two*»	12	7	5

В качестве теста использовали программу для моделирования кинетики упорядочения бинарных сплавов по вакансионному механизму диффузии состава *AnBm* в модели твердых сфер [8].

Тестирование проводилось на машине (конфигурация):

- *CPU* 2 x *Intel Xeon* 5660 (в сумме 12 ядер / 24 потока по 2.8*GHz*, 12*Mb Cache*, 6.40 *GT/s*);
- *RAM* 32*Gb DDR*3-10600 *ECC REG*;
- *Intel S3710 SSDSC2BA012T401*.

Компилятор, на котором производилось тестирование: Microsoft Visual Studio 2010.

Таким образом, при смене логических выражений числовых типов данных, можно оптимизировать и ускорить структуру «two» и снизить объем потреблении оперативной памяти.

4. Заключение

В данной статье была рассмотрена статистика языков программирования по популярности, после обсуждения которой был выяснен тот факт, что язык программирования *C++* является одним из самых популярных языков программирования высокого уровня. После обсуждения статистики были рассмотрены базовые аспекты языка программирования *C++* с примерами, а также детально рассмотрены методики программирования на *C++*, которые позволяют сделать код более удобным и практичным. Приведены примеры оптимизирующих компиляторов, за счет которых можно оптимизировать код в автоматическом режиме, а также рассмотрены виды ручной оптимизации, за счет которых можно ускорить работу программы и компиляцию. Приведены примеры структурированного кода с заменой логических выражений различных типов данных для оптимизации потребления оперативной памяти персонального компьютера и ускорения компиляции программы с разными параметрами для двух структур.

Список литературы:

[1]. Материалы с ресурса HABRAHABR: http://habrahabr.ru/ [Электронный ресурс] // Хабрахабр

[2]. Стивен Прата. Язык программирования C++ - Вильямс, 2012.

[3]. Медведев В.И. Особенности объектно-ориентированного программирования на C++/CLI, C# и Java - РИЦ «Школа», 2010.

[4]. Литвиненко Н.А. Технология программирования на C++. Win32 API-приложения - РИЦ «Школа», 2010.

[5]. Страуструп Бьярн. Программирование. Принципы и практика использования C++ - «Вильямс», 2011.

[6]. Материалы с ресурсов INTEL: http://www.intel.com/ [Электронный ресурс] // INTEL

[7]. Каретин И.И., Макаров В.А. Энергосберегающая оптимизация кода за счет использования отключаемых компонентов процессора / Труды Института системного программирования РАН Том 19. 2010 г. Стр. 187-194.

[8]. Халиков А.Р., Искандаров А.М. Моделирование кинетики упорядочения бинарного сплава по вакансионному механизму диффузии в модели твердых сфер / Известия высших учебных заведений. Физика. 2012 г. №12. Стр. 87.

Technique the Instrumentation a Code and Optimization of Code Lines in Modeling Phase Transitions on the Programming Language C++

E.V. Palchevsky <teelxp@inbox.ru>
A.R. Khalikov <khalikov.albert.r@gmail.com>
Ufa State Aviation Technical University, 450000, Russia, Ufa, st. Marx, 12

Abstract. This article discusses the technique of writing code that you can use to save time to write a specific program, instrumenting code technique on high-level language C ++, examples Coding algorithms are provided for system software and reviewed optimized compilers, by which it is possible to reduce compile time and speed program work. The definitions of the programming language «C» and «C ++», code structure, code optimization. It is also considered the use of statistics to promote programming languages. The basic advantages «C ++» programming language, in comparison with the «C». Produced designation of source code structure for better usability. Examples of the implementation of the code of the algorithm for further work with the code strings. Examples of clear and concise written comments to the source lines. Implemented class constructors and destructors, which aim to detailed optimization and reduction of the use of block RAM in the modeling of phase transitions. The types of optimizations: manual and automatic. In manual optimization described a few paragraphs: peephole, vnutriprotsedurnaya, local, and interprocedural optimization cycles. A structured code with sequential replacement of logical expressions with well developed various types of data for further optimization of consumption of RAM PC. It is shown that work with specific functions and provides practical examples of usage. Manufactured memory consumption calculations before and after optimization of the code lines at various compilers and server hardware, the configuration of which is also given in this article.

Keywords: code; code optimization; code instrumentation; techniques of writing code; C++; programming.

DOI: 10.15514/ISPRAS-2015-27(6)-6

For citation: Palchevsky E.V., Khalikov A.R. Technique the Instrumentation a Code and Optimization of Code Lines in Modeling Phase Transitions on the Programming Language C++. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 6, 2015, pp. 87-96 (in Russian). DOI: 10.15514/ISPRAS-2015-27(6)-6

References

[1]. Materials with resource HABRAHABR: http://habrahabr.ru/ [Electronic resource] // Habrahabr

[2]. Steven Pratt. The programming language C++ - Williams 2012.

[3]. Medvedev V.I. Features of object-oriented programming in C++ / CLI, C# and Java - RIP "School", 2010.

[4]. Litvinenko N.A. Programming Technology in C++. Win32 API-application - RIP "School", 2010.

[5]. Bjarne Stroustrup 5. Programming. Principles and practice of using C ++ - «Williams» 2011.

[6]. Materials with resources INTEL: http://www.intel.com/ [Electronic resource] // INTEL

[7]. Karetin I.I., Makarov V.A. Jenergosberegajushhaja optimizacija koda za schet ispol'zovanija otkljuchaemyh komponentov processora [Energy optimization of code by using disconnected components CPU] / Trudy ISP RAN [The Proceedings of ISP RAS], Volume 19, 2010, pp. 187-194. (in Russian)

[8]. Khalikov A.R., Iskandarov A.M. . Modelirovanie kinetiki uporjadochenija binarnogo splava po vakansionnomu mehanizmu diffuzii v modeli tverdyh sfer [Modeling the kinetics of ordering of the binary alloy of the vacancy mechanism of diffusion in the model of hard spheres] / Izvestija vysshih uchebnyh zavedenij. Fizika [Proceedings of the higher educational institutions. Physics]. 2012, №12. Pp. 87. (in Russian)