УДК 004.023

ПОСТРОЕНИЕ ПРОГРАММНОГО КОДА В УСЛОВИЯХ ВЫБОРА МЕЖДУ МАКСИМАЛЬНОЙ ПРОИЗВОДИТЕЛЬНОСТЬЮ И ЭКОНОМИЕЙ ПАМЯТИ

Новиков А.П., Горовенко Л.А.

Армавирский механико-технологический институт (филиал)
ФГБОУ ВО «Кубанский государственный технологический университет»,
Армавир, e-mail: shura.novikov.1988@bk.ru, lgorovenko@mail.ru

Данная статья предназначена для аудитории читателей, занятых в сфере программирования игровых модификаций. Авторы статьи рассматривают вопросы построения программного кода сточки зрения выбора между максимальной производительностью и максимальной экономией памяти электронной вычислительной машины. Исследование построено на поиске компромисса между этими показателями. Рассматриваются также некоторые известные способы оптимизации, построенные на переходе от одного типа данных к другому. В статье авторы описывают проведённые ими эксперименты по преобразованию компьютерного программного кода, которые приводят к более мощному способу оптимизации памяти: хранению данных в ячейках с последующим использованием их через побитовый сдвиг. В качестве иллюстрации проведённого исследования приводится фрагмент программного кода и статистические данные о полученных результатах. В заключении статьи подводятся итоги исследования и даются общие рекомендации по оптимизации программного кода с целью минимизации используемого резерва памяти.

Ключевые слова: память компьютера, оптимизация памяти, побитовый сдвиг, производительность

THE BUILDING CODE IN TERMS OF A CHOICE BETWEEN MAXIMUM PERFORMANCE AND MEMORY SAVING

Novikov A.P., Gorovenko L.A.

Armavir Institute of Mechanics and Technology, the branch of Kuban State University of Technology,
Armavir, e-mail: shura.novikov.1988@bk.ru, lgorovenko@mail.ru

This article is intended for audience working in the field of programming game modifications. The authors examine the issues of building software code in terms of a choice between maximum performance and maximum saving in memory of an electronic computer. The study is based on finding a compromise between these indicators. We also consider some known methods of optimization constructed on the transition from one data type to another. In the article the authors describe their experiments on transformation of computer program code, which lead to more powerful optimization of memory storage in cells and then use them using the bitwise shift. As an illustration of the conducted research is a piece of software code and statistics on the results. In conclusion, the paper summarizes the research and provides General guidance on optimizing code to minimize the used memory reserve.

Keywords: computer memory, memory optimization, bitwise, shift, performance

Производительность программного обеспечения (ПО) является важным аспектом в разработке любого программного продукта. Актуальность этого вопроса объясняется постоянно возрастающей сложностью и значимостью программных средств.

Низкая производительность может обуславливать неэффективность работы пользователей и, как следствие, негативную реакцию на программный’ продукт, потерю прибыли и клиентуры, доли рынка; перерасход средств на модификации и перепроектирование. Более того, переработка кода с целью увеличения производительности с большой вероятностью может разрушить первоначальную структуру системы, сводя на нет преимущества от использования объектно-ориентированного подхода. Наконец, маловероятно, что переработанный код сможет довести производительность системы до уровня, на котором она могла бы быть в случае изначального проектирования с учетом вопроса производительности. В худшем случае, будет невозможно достичь поставленных целей простой модификацией исходного кода системы, что обусловит необходимость полного перепроектирования или остановки проекта.

Каждый программист, начиная работу над программным кодом той или иной программы понимает, что все программы должны быть правильными, но некоторые программы должны быть быстрыми. Если производительность является ключевым фактором, то недостаточно использовать эффективные алгоритмы и структуры данных. Нужно писать такой код, который компилятор легко оптимизирует и транслирует в быстрый исполняемый код.

Зачастую перед теми, кто занимается программированием в сфере игровых модификаций встает вопрос: Как стоит строить код: максимально производительно, или максимально экономично в отношении памяти [1, 5, 6, 7, 9, 11].

Большинство программистов решит, что нужно искать компромисс [1, 2, 3]. Но что делать в тех случаях, когда компромисс найти тяжело, либо вообще невозможно? Искать!

Одним из направления уменьшений расходуемой памяти является оптимизация структур данных.

Допустим, нам нужно иметь в памяти 1000 ячеек содержащих двоичный ноль или единицу. 1000 целочисленных переменных (int) займет в памяти 1.95 Мб, в то время как 1000 переменных логического типа (bool) займет в памяти примерно 490 Кб. Этот метод перехода от использования одного типа данных к использованию другого приводит к экономии трёх четвертей используемой памяти [4, 8, 10]. Неплохо! Но, тут же возникает вопрос, а можно ли сэкономить ещё больше?

Еще одним направлением экономии памяти является использование более экономных типов данных.

Проведённые нами эксперименты по преобразованию компьютерного программного кода привели нас к ещё более мощному способу оптимизации памяти: хранение данных в ячейках с целью дальнейшего использования их через побитовый сдвиг.

В этом случае, на 1000 переменных мы потратим порядка 80 Кб оперативной памяти, что почти в 25 раз меньше, по сравнению с той же тысячью целочисленных переменных типа int.

Результаты проведённого сравнительного анализа представлены в табл. 1.

Таблица 1

Сравнительные анализ количества переменных различного типа на фиксированный объём оперативной памяти

Фиксированный объём памяти	1950 кб = 1.95 мб		
Тип переменной	int	bool	bit's
Количество переменных	1000	4000	>24 000

В качестве эксперимента, каждый вид переменных был записан, использован и перезаписан 10 000 раз с целью определения скорости работы программы, реализующей одну и ту же задачу посредством использования различных типов переменных. Результаты эксперимента представлены в табл. 2:

Таблица 2

Результаты эксперимента

Тип данных	Время (усл. ед.)
int	~190
bool	~170
bit's	~240

Как показал эксперимент, bit-овые переменные оказались «медленнее» остальных. Почему? Потому, что для выполнения побитовых операций генерируются дополнительные инструкции в секции кода, которые идут параллельно с нашими bit-ами. А, как известно, любая операция в той или иной степени «загружает» машину, что, естественно, сказывается на её быстродействии. Проведённый нами эксперимент показал также, что не все программные двигатели имеют многопоточность и выполняют все операции последовательно.

Приведём фрагменты программного кода, иллюстрирующие проведённое нами исследование.

Итак, создаем:

```
#define MAX_USERS (1000)
enumBits:(<<= 1) //не забываем,
не более 32
{
EnumOne = 1,
EnumTwo, ...
};
newBits:UserBits[MAX_USERS];
```

Узнаем значение и обнуляем весь массив пользователя:

```
UserBits[userid] &EnumOne;
UserBits[userid] = Bits:0;
```

Устанавливаемячейкезначение 1 (true)

```
UserBits[userid] |= EnumOne;
```

Устанавливаемячейкезначение 0 (false)

```
UserBits[userid] &= ~EnumOne;
```

Меняем значение в ячейке (при 0 ставим 1, и наоборот)

```
UserBits[userid] ^= EnumOne;
```

Таким образом, для грамотной работы по оптимизации памяти, прежде всего, необходимо понимать, как эта работа «устроена изнутри», какие действия происходят, когда компилятор преобразует наши скупые синтаксические конструкции в машинный код.

Современный процессор скрывает огромную вычислительную мощь. Но, чтобы получить к ней доступ, нужно писать

программы в определённом стиле. Решить какие трансформации и к какой части кода применить– это и есть рецепт написания быстрого кода.

В оптимизации есть несколько важных моментов. Оптимизация должна быть естественной. Оптимизированный фрагмент кода должен легко вливаться в программу, не нарушая логики ее работы. Он должен легко вводиться в программу, изменяться или удаляться из нее.

Оптимизация должна приносить существенный прирост производительности. Оптимизированная программа должна работать минимум на 20%-30% эффективней, чем ее неоптимизированный аналог, иначе оптимизация теряет смысл. Зачем мучиться и вносить изменения в уже готовый код, если этоне дастпрактически никакого результата?

При этом разработка (и отладка) критических областей не должна увеличивать время разработки программы более чем на 10%-15%.

Помимо рассмотренных в нашей статье способов оптимизации, мы также предлагаем известную базовую стратегию оптимизации программного кода, а именно:

● Выбирайте эффективные алгоритмы и структуры данных. Никакой компилятор не заменит плохие алгоритмы или структуры данных на хорошие.

● Избегайте блокировщиков оптимизации, чтобы помочь компилятору генерировать эффективный код. Избавьтесь от ненужных вызовов функций. Если возможно, вынесите вычисления за пределы цикла. Избавьтесь от ненужных запросов к памяти. Введите временные переменные для хранения промежуточных результатов.

● Используйте низкоуровневую оптимизацию. Применяйте раскрутку циклов, чтобы уменьшить накладные расходы на цикл.

Список литературы

1. Бондар М.Д., Часов К.В. Оверклокинг или повышение производительности процессора // Развитие природоохранной системы и экологии города: атериалы региональной научно-практической молодежной интернет-конференции. 2017. С. 154-156.

2. Горовенко Л.А. Логическое программирование и искусственный интеллект // Научный потенциал вуза - производству и образованию. Сборник трудов научно-практической конференции профессорского-преподавательского состава Армавирского механико-технологического института (филиала) ГОУ ВПО «Кубанский Государственный технологический университет». Том. 2. – Армавир: Издательство АФЭИ, 2005. – С. 303-304.

3. Горовенко Л.А. Логическое программирование как средство решения задач искусственного интеллекта // Современные проблемы математики и информатики: Сборник научных трудов. Вып. 1 / Сост. Н.Г. Дендеберя, С.Г. Манвелов. – Армавир: редакционно-издательский центр АГПУ, 2004. – С. 56-57.

4. Горовенко Л.А. Математические методы компьютерного моделирования физических процессов: учебное пособие / Л. А. Горовенко. – Армавир: РИО АГПУ, 2016. – 104 с.

5. Горовенко Л.А. Опыт создания обучающих программ // Нормативные технологии диагностики в современной экономике и обществе. Материалы межвузовской научно-практической конференции. / Под ред. А.И. Шарнова. Ст. Отрадная: Изд-во ОГИ, 2001. – С. 201-205.

6. Горовенко Л.А. Построение архитектуры интеллектуальных обучающих систем нового поколения // Конкурентный потенциал вуза в условиях рынка образовательных услуг: теория и практика общественного опыта. Материалы межвузовской научно-практической конференции (24-26 мая 2002 г.). – Армавир: РИО АФЭИ, 2002 . – С. 37-40.

7. Горовенко Л.А. Педагогические аспекты эффективности применения автоматизированных обучающих систем с элементами искусственного интеллекта // Конкурентный потенциал вуза в условиях рынка образовательных услуг: теория и практика общественного опыта. Материалы межвузовской научно-практической конференции (24-26 мая 2002 г.). – Армавир: РИО АФЭИ, 2002 . – С. 33-37.2002 . – С. 33-37.

8. Горовенко Л.А., Манин М.П. Применение математического аппарата методов оптимизации в задачах моделирования электросбережения // Сборник докладов, отмеченных наградами XXI научной конференции студентов и аспирантов АМТИ, посвященной 70-летию Победы в Великой Отечественной войне. Армавир: ООО «Редакция газеты «Армавирский собеседник», подразделение Армавирская типография», 2015. – С. 88 – 92.

9. Горовенко Л.А., Шарнова В.А. Технология применения комбинаторного анализа в играх с угадыванием числа // Сборник докладов по материалам юбилейной XX студенческой научной конференции АМТИ, Армавир: ОАО «Армавирское полиграфпредприятие», 2014. – С.101-106.

10. Горовенко Л.А., Коврига Е.В. Теория и практика компьютерного моделирования физических процессов: учебное пособие / Л.А. Горовенко. – Армавир: РИО АГПУ, 2017. – 132 с.

11. Шарнова В.А., Горовенко Л.А. Технология применения методов комбинаторного анализа в играх с угадыванием числа // «Международный студенческий научный вестник»: 2015. – № 5–4. – С. 586-587.