УДК 519.683.8

ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ МЕТОДОВ ОПТИМИЗАЦИИ ИСХОДНОГО КОДА НА ПРИМЕРЕ ПРОГРАММЫ МОДЕЛИРОВАНИЯ РАСПИСАНИЯ ЗАНЯТИЙ

Димитриев А.П.

ФГБОУ ВО «Чувашский государственный университет им. И.Н. Ульянова», Чебоксары, e-mail: dimitrie1@yandex.ru

Статья посвящена исследованию эффективности методов оптимизации исходного кода на примере программы моделирования расписания занятий. Важнейшей задачей является исследование оптимизации исходного кода программного обеспечения при встраивании функций и перестроении системы вложенных циклов. В статье использована программа, предназначенная для имитационного моделирования расписания учебных занятий. Рассмотрены вычислительные задачи поиска в «глубину» и организации вложенных циклов. Исследование проведено с применением нескольких разновидностей процессоров компьютеров. С каждым процессором проведено по две серии экспериментов с различными комбинациями параметров. Получены результаты влияния оптимизации кода компилятором системы программирования Delphi на скорость вычислений. Встраивание функций применять не эффективно. Из рассмотренных методов оптимизации исходного кода программы эффективно применять изменения системы циклов. Полученные результаты можно обобщить на широкий класс задач программирования.

Ключевые слова: оптимизация исходного кода, программа, функция, цикл, рекурсия

RESEARCH OF EFFICIENCY OPTIMIZATION TECHNIQUES OF THE SOURCE CODE AT THE SAMPLE PROGRAM SIMULATION CLASS SCHEDULES

Dimitriev A.P.

Federal state budget educational institution of higher education «Chuvash State University named after I.N. Ulyanov», Cheboksary, e-mail: dimitrie1@yandex.ru

The article is devoted to the study of the efficacy some methods optimization of software source code, such as inlining functions, etc. Computational speed used as the main performance criterion. As an example, used a program designed for simulation schedule classes. Considered the computational problem of searching in the depth and organization of nested loops. The study was conducted using several kinds of computer processors that can found in currently. With each processor carried out two series of experiments with different combinations of parameters. Considered differences at the computing speed due to optimization that performed by the compiler of the programming system Delphi. Preliminary calculation products of numbers is not always effectively. The inlining of functions is not advisable to apply. The results can be generalized to a wide class of programming tasks.

Keywords: optimization source code, program, function, loop, recursion

Составление расписаний учебных занятий является одной из сложных задач автоматизации. В продаже и свободном распространении имеется большое количество программных продуктов для ее решения, таких как «Ректор-ВУЗ» [4], «FET» и др. Как правило, такие программы при оптимизации расписания опираются на критерий качества, выраженный в виде целевой функции. Имитационное моделирование расписания занятий [2] включает модель целевой функции при оптимизации.

Известны различные методы оптимизации расписания учебных занятий. Одним из известных методов является метод последовательного перебора. Однако уже при малом числе групп данный метод требует больших вычислительных затрат, так как это NP-полная задача. В этом случае актуальной является задача сокращения времени вычислений. Одним из подходов решения данной задачи служит применение методов оптимизации исходного кода программы [1].

Актуальность работы также связана с тем, что применение методов оптимизации кода важно не только в задаче составления расписаний, так как данные методы разработаны без привязки сугубо к вышеназванной задаче. Одним из таких методов оптимизации является встраивание кода какой-либо функции в вызывающий ее код. При этом сокращается время вычислений, благодаря тому, что не нужно сбрасывать и загружать заново очередь команд процессора и передавать параметры функции через стек. Для микропроцессоров Intel 80286 этот метод успешно применялся для решения задачи моделирования расписания. Важно выяснить, насколько встраивание функций эффективно для современной вычислительной техники. В настоящее время архитектура микропроцессоров претерпела существенные изменения, например конвейер команд и кэш-память стали объёмнее, а это влияет на время обработки последовательности команд.

Статья посвящена исследованию эффективности методов оптимизации исходного кода на примере программы моделирования расписания занятий. При анализе целесообразности применения тех или иных методов оптимизации необходимо учитывать два фактора.

Во-первых, одно из важнейших требований к программному обеспечению – его высокая производительность. Поэтому основной задачей данной работы является сравнение производительности тестовой программы при применении различных методов оптимизации кода. Во-вторых, сложность исходного кода, влияющая на затраты ресурсов при его создании и сопровождении (длительность разработки, требуемая квалификация разработчиков и т.д.).

Сложностью исходного кода иногда можно пренебречь ради достижения максимально возможной производительности, однако это допустимо лишь для сравнительно небольшого числа приложений – например для систем реального времени. Поэтому данный фактор также следует учитывать.

Имитационное моделирование расписания учебных занятий и методы исследования

В имитационном моделировании расписания учебных занятий задача состоит в оптимальном расположении объектов по дискретным интервалам времени, причем имеется n объектов и m интервалов [2]. Вектор расположений объектов имитирует расписание. У каждого объекта по два целочисленных параметра, исходя из значений которых на основании расписания однозначно вычисляется значение целевой функции.

Первый эксперимент состоит в запуске полного перебора вариантов расположения объектов для $n = 10$, $m = 20$ с оцениванием каждого варианта. Для этого используется прием программирования, называемый рекурсией, которая определяется как вызов подпрограммой самой себя. Рекурсия реализует следующую схему поиска в «глубину», начинающую работу со второго уровня вложенности рекурсии при взаимно однозначном соответствии между объектами и уровнями: объект текущего уровня в цикле от 1 до m смещается вправо на один временной интервал, и для каждого такого смещения вызывается рекурсивная функция, выполняющая аналогичные смещения с вызовами самой себя и наращиванием уровня, пока уровень не достигнет числа n. При каждом смещении объекта n-го уровня вместо вызова рекурсии производится вычисление целевой функции, причем для повышения быстродействия вычисляется только ее изменение и складывается со старым значением. Поиск завершается при последнем в цикле возврате из рекурсии на втором уровне. Всего оценивается $m^{(n-1)} = 20^9 = 5{,}12 \cdot 10^{11}$ вариантов, так как перемещения первого объекта программой не рассматриваются в силу особенностей задачи.

На основе данной схемы поиска предлагается первая серия экспериментов, в которой рассматривается метод встраивания рекурсивных функций. Некоторые рекурсии, называемые «хвостовыми», компиляторы при включенном режиме оптимизации умеют оптимизировать самостоятельно. Особенность этих рекурсий в том, что вызов функцией самой себя является последней командой перед возвратом из функции. В данной работе, как следует из исходного кода, рекурсии хвостовыми не являются, поэтому оптимизацию рекурсий автоматизировать не удается [3] и используется оптимизация кода вручную.

В данной серии экспериментов рассматриваются четыре комбинации методов оптимизации исходного кода:

1. Обычные вызовы рекурсии, которые рассмотрены выше. При вычислении целевой функции здесь и далее применяется такая эвристика, как предварительное вычисление массива произведений параметров объектов размером $n \times n$.

2. Вызовы рекурсии с заполнением массивов памяти, имитирующих локальные переменные рекурсивной функции. Это позволяет сравнить получаемую скорость работы программы со скоростью, когда рекурсии действительно будут встроены, поскольку в этом случае также потребуется заполнение указанных массивов. Необходимость данных массивов обусловлена тем, что для каждого уровня вложенности рекурсии требуются свои адреса памяти для хранения локальных переменных. В программной реализации индексы массивов соответствуют уровням вложенности.

3. Встраивание рекурсивных вызовов. Для этого используются вышеописанные массивы и две команды безусловного перехода GOTO, в начале подпрограммы, содержащей текст встроенной рекурсии, и ближе к завершению подпрограммы, но не в самом конце.

4. Обычные вызовы рекурсии без предварительного вычисления массива произведений целых чисел, используемых в целевой функции. Цель – определить, не лишено ли смысла вычисление такого массива в качестве метода оптимизации кода, так как в некоторых процессорах произведение целых чисел вычисляется всего за один такт.

Вторая серия экспериментов связана с методом встраивания обычных (нерекурсивных) функций. Для этого по 100 000 раз запускается функция, вычисляющая значение целевой функции для псевдослучайных моделей расписаний при $n = 40$, $m = 100$.

Рассматриваются четыре комбинации других методов оптимизации исходного кода: *A* – обычный вызов функций; *B* – встраивание функций, т.е. вместо вызова функции в вызывающем модуле используется текст, эквивалентный тексту функции; *C* – встраивание функции, при котором ее внутренний вычислительный цикл становится внешним, т.е. рассматривается метод перестроения системы вложенных циклов. В данном случае вложены три цикла, из которых два внешних сначала выполнялись от 1 до $n - 1$, а внутренний от 1 до m, а теперь цикл от 1 до m стал внешним, а два цикла от 1 до $n - 1$ стали внутренними; *D* – встраивание функции, при котором ее внутренний цикл (от 1 до m) помещен между двумя циклами от 1 до $n - 1$.

Для получения обобщенных результатов предлагается рассмотреть работу тестовой программы на процессорах различных годов выпуска (которые связаны с тактовой частотой), моделей и типов. Исследование проведено с применением персональных компьютеров с процессорами $П_1$, $П_2$, $П_4$, $П_5$ (табл. 1) и ноутбука с процессором $П_3$.

Программа разработана в среде Turbo Delphi и составлена таким образом, что даже в случае многоядерной архитектуры для вычислений используется только одно ядро процессора. Включение оптимизации выполнялось с помощью опции Optimization группы Code generation в настройках компилятора.

Результаты встраивания рекурсий и предварительных действий

Эффективность применяемых методов можно выразить индексом производительности для разных процессоров $\lambda_1 = 1/(fT)$ – числом вычислений (в данном случае переборов вариантов) в секунду, которое привносит 1 МГц процессора (табл. 2). Здесь f – тактовая частота процессора, МГц; T – среднее время вычислений при переборе вариантов, с. Время вычислений определено с помощью диспетчера задач как суммарное время работы центрального процессора и в табл. 2 не внесено.

Таблица 1

Обозначение	Название процессора	Частота, МГц
$П_1$	x86 Family 6 Model 37 Stepping 5 GenuineIntel	3345
$П_2$	AMD A8-3850 APU	2900
$П_3$	AMD E2-6110 APU	1500
$П_4$	x86 Family 6 Model 15 Stepping 2 GenuineIntel	1607
$П_5$	Intel(R) Core(TM)2 Duo CPU E4600	2400

Таблица 2

$\lambda_1 \cdot 10^6$ для разных процессоров и методов оптимизации при $n = 10$, $m = 20$

Процессор	Комбинация параметров			
	1	2	3	4
$П_1$	44,0	35,6	34,8	44,0
$П_2$	45,4	31,9	31,3	42,1
$П_3$	46,3	35,3	35,7	46,6
$П_4$	44,4	37,0	35,8	42,6
$П_5$	43,4	37,2	35,9	42,5
$П_1^*$	42,7	35,6	34,8	45,3
$П_2^*$	24,1	20,3	20,3	23,8
$П_3^*$	31,3	25,6	25,9	31,7
$П_4^*$	34,0	30,1	29,6	31,4
$П_5^*$	33,1	29,3	28,9	30,6

Примечание. * Без оптимизации кода в настройках компилятора (очевидно, ее использование влияет на эффект от применения приемов оптимизации исходного кода на основе «искусства» программирования).

На основе данных в табл. 2 можно сделать ряд выводов. Так как программисты, стремясь повысить производительность программного обеспечения, включают в компиляторе режим оптимизации, сначала рассмотрим результаты, где оптимизация включена.

В исходном варианте индекс производительности у всех процессоров почти одинаков. Заполнение массивов уменьшает λ_1 на (22 ± 8)%. Это заметнее на новых процессорах и особенно на процессорах AMD. Встраивание рекурсивных вызовов в дополнение к заполнению массивов обычно немного уменьшает λ_1. В итоге сравнение встраивания рекурсий и вызова рекурсий показывает, что λ_1 уменьшается на (24 ± 7)%.

Таким образом, необходимости встраивания рекурсивных вызовов нет. Тем более что при встраивании значительно усложняется исходный код. Это отрицательно сказывается на всех фазах жизненного цикла исходного кода [5], увеличивая трудоемкость его разработки и дальнейшего сопровождения. Данное обстоятельство становится особенно весомым при значительных перерывах в работе с исходным кодом и при передаче его на сопровождение третьим лицам [6].

Если оптимизацию выключить, производительность снижается меньше: 11–18% и 13–19% соответственно. В среднем, как видно в табл. 2, λ_1 уменьшается на 23%. Для новых процессоров Intel в среднем нет снижения производительности, для старых в среднем это 22%, а для новых процессоров AMD – 30%. Однако при обычном вызове рекурсий и выключенной оптимизации наблюдается снижение производительности для всех процессоров.

Кроме того, можно сделать вывод, что на новых процессорах Intel предварительно произведения целых чисел можно не вычислять. Вычислять их рекомендуется лишь на старых процессорах.

Результаты встраивания функций и перестройки системы циклов

Второй индекс производительности для разных процессоров (табл. 3) $\lambda_2 = 1/(fT)$ – число однократных вычислений значений целевой функции в секунду, которое привносит 1 МГц процессора, где T – время 10^5 вычислений значений целевой функции, с.

С помощью расчетов из табл. 3 можно заключить следующее. Встраивание функций почти всегда снижает производительность, за исключением новых процессоров Intel с оптимизацией кода средствами компилятора. Снижение производительности достигает 44–46%, причем оно значительно заметнее при использовании процессоров AMD.

У метода встраивания функций есть и другой недостаток. Так как программа использует одну и ту же функцию во многих строках, ее встраивание приводит к значительному увеличению объема программы. Таким образом, встраивание даже нерекурсивной функции фактически лишено смысла.

Рассмотрим, как влияет на производительность изменение структуры вложенных циклов. Первый вариант модификации структуры цикла сильно снижает λ_2: в пределах 1,5–3,4 раза при оптимизации средствами компилятора и в 1,9–2,5 раза без нее. Второй вариант модификации у процессоров Intel тоже снижает λ_2, но не так сильно: от 12 до 30% при оптимизации средствами компилятора и от 15 до 23% без нее. У AMD, напротив, увеличивает соответственно от 13 до 44% и от 32 до 47%. Следовательно, первоначальный вариант модификации структуры циклов в основном можно считать наиболее удачным.

Таблица 3

λ_2 для разных процессоров при $n = 40$, $m = 100$

Процессор	Комбинация параметров							
	A	*B*	*C*	*D*	*A**	*B**	*C**	*D**
$П_1$	1,99	1,99	0,79	1,76	1,43	1,33	0,59	1,13
$П_2$	1,95	1,33	0,59	1,92	0,94	0,73	0,39	0,96
$П_3$	1,48	1,14	0,74	1,29	0,85	0,59	0,41	0,87
$П_4$	2,46	2,37	0,71	1,65	1,45	1,42	0,56	1,09
$П_5$	2,45	2,31	0,73	1,67	1,44	1,39	0,60	1,10

Примечание. * Без оптимизации кода в настройках компилятора.

Снижение производительности из-за отказа от оптимизации средствами компилятора в среднем составляет 37 % (от 28 до 41 %). При этом оно наименьшее в первом варианте модификации циклов и меньше у процессоров Intel.

В целом по двум сериям экспериментов снижение производительности из-за отказа от оптимизации средствами компилятора в среднем равно 30 %. Следовательно, оптимизацию при компиляции следует включать. Исключением являются процессоры Intel, приведенные в табл. 1. На них при отключенной оптимизации в компиляторе и отказе от предварительного вычисления произведений можно достичь производительности примерно на 3 % выше, чем в исходном варианте.

Заключение

Исследование эффективности методов оптимизации исходного кода на примере программы моделирования расписания занятий показывает, что целесообразно применять изменения системы циклов. Предварительное вычисление произведений целых чисел эффективно не всегда. Встраивание функций применять неэффективно. Это в равной степени относится как к обычным, так и к рекурсивным функциям. Как показывают эксперименты, это приводит к снижению производительности в среднем на 18 %.

Важно применение оптимизации компилятора. Полученные результаты можно использовать и для других задач, так как такие конструкции, как вложенные циклы и функции, применяются в программировании практически всегда.

Список литературы

1. Гордеев А.В., Молчанов А.Ю. Системное программное обеспечение: учебник для вузов. – СПб.: Питер, 2003. – 736 с.

2. Димитриев А.П. Критерий прекращения поиска решений при дискретной оптимизации расписаний // Современные проблемы науки и образования. – 2015. – № 2–2. – 8 с.

3. Касперски К. Техника оптимизации программ. Эффективное использование памяти. – СПб.: БХВ-Петербург, 2003. – 464 с.

4. Ректор-Программа Расписание: Продукты [Электронный ресурс]. – Режим доступа: http://rector.spb.ru/ (дата обращения: 15.04.16).

5. Фадеев С.Г. Жизненный цикл исходного кода программного обеспечения // Современные проблемы науки и образования. – 2014. – № 6. – С. 175.

6. Фадеев С.Г. Работа с чужим исходным кодом // Современные тенденции развития науки и технологий. – 2015. – № 2–2. – С. 152–156.

References

1. Gordeev A.V., Molchanov A.Yu. Sistemnoye programmnoye obespecheniye: Uchebnikdlyavuzov [System software: Textbook for high schools]. St. Petersburg, Piter, 2003. 736 p.

2. Dimitriev A.P. Sovremennye problemy nauki i obrazovaniya, 2015, no. 2–2. 8 p.

3. Kasperski K. Tekhnika optimizatsii programm. Effektivnoye ispolzovaniye pamyati (Technique optimization programs. Efficient use of memory). St. Petersburg, BKHV-Peterburg, 2003. 464 p.

4. *Rektor-ProgrammaRaspisaniye: Produkty*(Rector-Program Schedule: Products) Available at: http://rector.spb.ru/ (accessed 15 April 2016).

5. Fadeev S.G. Sovremennye problemy nauki i obrazovaniya, 2014, no. 6, pp. 175.

6. Fadeev S.G. Sovremennye tendentsii razvitiya nauki i tekhnologii, 2015, no. 2–2, pp. 152–156.