О.А. Четверина, П.А. Степанов, к.ф.-м.н. М.И. Нейман-заде
(ЗАО «МЦСТ», ПАО «ИНЭУМ им. И.С. Брука)

O. Chetverina, P. Stepanov, M. Neiman-zade

АВТОМАТИЧЕСКАЯ НАПРАВЛЕННАЯ ОПТИМИЗАЦИЯ ПРОЦЕДУР

AUTOMATIC STATISTICAL SELECTION OF OPTIMIZATION SEQUENCE FOR PROCEDURE

При проведении преобразований кода оптимизирующим компилятором решающее значение имеет выбор последовательности настроек. В статье представлен подход, при котором компилятор проводит выбор оптимизирующей последовательности для каждой процедуры на основе статистической информации. Вводится функционал качества, позволяющий оценить эффективность выбираемой последовательности как с точки зрения производительности, так и с точки зрения затрачиваемого времени компиляции. Приведены результаты применения описанного механизма по сравнению с базовой компиляцией.

Optimizing compilers apply optimizations and code transformations in certain sequence to achieve performance improvement. Usually one best-on-average sequence for each level of optimization is used, which gives suboptimal result due to optimizations interference and ambiguity effect of some optimizations on different contexts. Some iterative approaches allow to choose better optimization sequence for each program or even each procedure; these approaches have serious limitations on usage. This paper presents a statical approach in which compiler chooses the optimizing sequence for each procedure on early compilation stage using statistical information. We introduce quality functional for assessing the effectiveness of selected sequence both in terms of performance and compilation time. Results gained with described technique comparing to baseline compilation modes demonstrate effectiveness of presented approach.

Ключевые слова: статистическая настройка оптимизации,

оптимизирующая линейка, уменьшение времени компиляции, функционал качества, классификация процедур.

Keywords: statistical performance tuning, optimization sequence, reducing compilation time, compilation quality functional, procedures classification.

Введение

Производительность современных вычислительных систем во многом зависит от проведения компилятором качественного планирования исполняемого кода с использованием оптимизирующих преобразований. Особенно значима оптимизация кода применительно к процессорам с VLIW-архитектурой, поскольку в этом случае производительность обеспечивается за счет статического планирования одновременного исполнения сразу нескольких команд в одном такте. При этом компилятор последовательно выполняет преобразования с использованием эвристических оценок, сформированных статистическим образом на некотором тестовом наборе задач. Проведенная таким образом оптимизация конкретной задачи может не дать максимально достижимого повышения производительности, вследствие чего вероятна необходимость в дополнительной ручной настройке набора и эвристик применяемых преобразований. Как правило, пользователь в полной мере не обладает информацией о работе оптимизаций, поэтому такой способ затратен по времени и не всегда приводит к оптимальным результатам, которых можно достичь, используя автоматическую настройку.

Основным методом реализации автоматической настройки являются итеративные решатели, которые можно разделить на две основные категории: многократная компиляция с последующим исполнением [1] и многократная компиляция с оценкой времени исполнения по спланированному коду [2]. Дополнительным достоинством обоих решений является возможность выбрать не общие эвристики для всей задачи, а наиболее подходящие для каждой процедуры.

К недостаткам этих методов стоит отнести требование исполнения задач на предста-

вительных данных и значительные затраты времени на компиляцию. Учет времени, наряду с производительностью итогового кода, особенно важен для компиляторов под VLIW-архитектуру в силу их сложности. Так, в процессе проведения базовой компиляции для архитектуры «Эльбрус» осуществляется более 300 этапов компиляции, и затрачиваемое время в среднем в 21 раз больше, чем в случае неоптимизирующей сборки. Для ряда задач такой расход времени становится неприемлемо большим при статической компиляции и недопустимым – в динамическом варианте, когда время компиляции добавляется к времени исполнения.

В своей работе авторами была поставлена задача повышения эффективности оптимизирующих преобразований за счет направленной по-процедурной компиляции, не обладающей недостатками итеративных подходов. Ее решение основывалось на статистическом выборе одной из имеющихся настроек оптимизации по совокупности доступных характеристик процедуры в начальной стадии компиляции. Соответствующий механизм был обозначен как *система направленной оптимизации процедур* (СНОП). В процессе работы было обнаружено, что аналогичная идея была высказана в [3], но к текущему моменту был представлен только подход к спецификации наборов эвристик без реализации самого автоматического выбора.

Поскольку одной из решаемых проблем было уменьшение времени компиляции, при проведении работы также исследовалась возможность повышения скорости компиляции за счет построения последовательностей оптимизаций. В этой связи надо отметить, что на рассматриваемом в [4] тестовом пакете для компилятора gcc было показано сокращение времени компиляции в среднем в два раза без потери производительности за счет выбора для каждой процедуры минимальной достаточно эффективной последовательности преобразований. С целью наиболее полного регулирования процесса компиляции авторами реализован механизм, позволяющий задавать последовательность как оптимизирующих, так и аналитических этапов компиляции, и разработан набор последовательностей, позво-

ляющий эффективно спланировать код различных задач.

В разделе 2 вводится *функционал качества* и поясняется необходимость его использования для оценки фиксированного назначения последовательностей оптимизаций процедурам при решении статической задачи выбора. В разделе 3 описаны возможные подходы к решению задачи минимизации функционала с использованием статистических методов.

Разработанный механизм автоматического выбора последовательностей оптимизаций оказался достаточно эффективным как с точки зрения производительности, так и с точки зрения уменьшения времени, затрачиваемого на компиляцию. Результат применения СНОП приведен в разделе 4. На текущий момент предлагаемый механизм реализован в компиляторе для архитектуры «Эльбрус» с широким командным словом и заменил собой базовую оптимизацию.

1. Построение набора оптимизирующих последовательностей

В большинстве случаев при построении итерационных решателей используется набор подаваемых к моменту решения опций, регулирующих применение отдельных оптимизаций. Для удобной настройки не только набора, но и последовательности применяемых преобразований и аналитических этапов, в компиляторе для архитектуры «Эльбрус» был реализован механизм внешней передачи компилятору полной последовательности производимых компилятором действий, т.е. оптимизирующих линеек. Этот механизм является внутренним, и его применение пользователем не предполагается, тем не менее, он позволяет достаточно легко создавать и обновлять оптимизирующие последовательности, что значимо для использования при разработке и настройке компилятора.

Расчет на возможность статического выбора эффективной линейки из определенного набора связан с предположением о существовании неявной типизации процедур с точки зрения требований к оптимизации. В различных исследованиях [4, 5] предлагаются варианты автоматического построения набора последовательностей оптимизирующих преоб-

разований, но с точки зрения проблемы *автоматического* выбора они не очень удачны, т.к. не учитывают возможность различия в предсказаниях исполнения по исходному коду. Поскольку основная задача авторов состояла в том, чтобы обеспечить возможность автоматического выбора, было разработано небольшое число линеек, в которых последовательности и настройки преобразований сформированы с учетом оптимизации различных типов процедур. В качестве первых двух линеек были выбраны с некоторыми изменениями последовательности, использующиеся компилятором для базового уровня оптимизации (режима -O3) и для оптимизации с ограничением роста кода и спекулятивности (режима -O2). Еще две линейки были разработаны с помощью выделения наборов опций, наиболее часто применяемых с целью достижения пиковой производительности, с последующей доработкой оставшихся в линейках фаз для эффективной совокупной работы на тестовом пакете. По своей сути одна из полученных последовательностей позволяет качественно спланировать исполнение многоитерационных циклов с большой нагрузкой на память, а другая – циклов с большим количеством ветвлений. Список использованных для настройки оптимизаций и аналитических этапов приведен в табл. 1. При этом оптимизации в линейках могут быть по необходимости выключены, включены или на них изменены настройки применения. Кроме того, для разных линеек может отличаться последовательность осуществления преобразований.

Таблица 1

Список оптимизаций, используемых для настройки последовательностей в СНОП

Оптимизации		**Аналитические этапы**
•cache optimization •loop unswitching •group loop unswitching •split by index •split be condition •split by inequality •loop undercondition •loop undercase •vector canonization •vector idiom	•reroll •reverse •global copy propagation •move optimization •altexpel •nesting •vectorization •maw •peeling •unroll	•loop dependence analysis •forks •dependence analysis •value numbering •nodes sizes •unsigned to signed

Оптимизации		Аналитические этапы
•vector invariant removing •memory access widening •additional scalar replacement •loop lazy code motion •redundant loop load elimination •redundant loop store elimination •softpipe •interchange	•unroll_fuse •altexpel •regions+if conversion (superb-locking) •srtmd •dam •guess prefetch •list prefetch	

2. Функционал качества

Одной из основных проблем приведенного подхода является необходимость построения функционала в пространстве соответствий линеек процедурам, минимизация которого соответствует наилучшему выбору: $P = \{p_1,\ldots, p_n\}$ – набор рассматриваемых процедур, $L = \{l_1,\ldots, l_k\}$ – набор линеек, функционал $F(l(p_1,\ldots, l(p_n))\rightarrow\mathbb{R}$ определен в пространстве L^n, где $l{:}P\rightarrow L$.

В рассматриваем случае числовая оценка нужна для установки соотношения между увеличением производительности и допустимым при этом замедлении компиляции. В то же время при решении задачи автоматического выбора введение функционала даже необходимо с целью оценки изменения только применительно к производительности. Причина заключается в том, что автоматический выбор предусматривает использование характеризующих процедуру параметров, значения которых могут совпадать для разных процедур. Для иллюстрации этой проблемы в табл. 2 приведен пример, при котором выбор оптимальной линейки для двух из трех процедур с одинаковыми характеризующими параметрами является неоптимальным с точки зрения совокупного времени исполнения. В качестве подходящего функционала в этом случае может быть использована сумма времен исполнения процедур при компиляции с использованием соответствующих линеек.

Выбор функционала для одновременного учета времени компиляции и исполнения зависит от предъявляемых требований к оптимизации. При этом кроме достижения оптимума выбора в точке минимума функционал должен регулировать и стоимость ошибки

отклонения от него. Для обеспечения этих требований был построен функционал следующего вида:

$$F(l(p_1 0,...,l(p_n)) = \left(\sum_i exe(p_i, l(p_i))\right)^r \left(\sum_i comp(p_i, l(p_i))\right), \qquad (1)$$

где: $exe(p_i, l)$ – время исполнения процедуры p_i при компиляции линейкой l; $comp(p_i, l)$ – время компиляции процедуры p_i при использовании линейки l. Значение r позволяет регулировать уровень оптимизации. Для регулирования на рассматриваемом наборе в базовом режиме (-O3) было эвристически подобрано оптимальное значение $r = 7$. При этом значении параметра замедление исполнения на всем пакете на 1% по сравнению с минимально возможным допускается при ускорении компиляции на 7%, замедление на 2% – при ускорении на 15%, замедление на 5% – при ускорении на 43%, замедление на 20% – при ускорении компиляции в 5 раз и так далее.

Таблица 2

Выбор оптимизирующей линейки для процедур с одинаковыми характеристиками

Процедуры / Линейки	**Proc1**	**Proc2**	**Proc3**	**Суммарное время исполнения**
Line1	1000	500	1000	2500
Line2	900	400	2000	3300

3. Автоматический выбор эффективной последовательности

Дальнейшее решение проблем автоматического выбора было разбито на две части:

1) выделение показательного набора характеризующих процедуры параметров, которые доступны на раннем этапе компиляции;

2) построение решателя, выбирающего по имеющимся характеристикам одну из оптимизационных линеек таким образом, чтобы на заданном наборе процедур минимизировался функционал качества.

В качестве параметров рассматривались различные характеристики графа управления и количество различных типов операций в процедуре. Для выделения независимых

параметров рассчитывались матрицы корреляции, и проводилась требуемая нормировка. В результате был сформирован следующий набор параметров: средняя плотность процедуры, под которой подразумевается среднее оценочное количество выполнений каждой операции процедуры; максимальная глубина вложенности циклов; количество вызовов в процедуре; число операций в процедуре; среднее количество операций в узлах управляющего графа процедуры, которое указывает на степень его ветвлений. Остальные характеристики являются долей различных видов операций: операций с плавающими вычислениями, операций чтения по указателю, операций вычисления адреса.

При выборе автоматического решателя было проведено исследование существующих статистических методов кластеризации и классификации. Стоит отметить, что для всех статистических методов эффективность решателя ограничивается качеством выбранных характеристик, в то же время без фиксации решателя или способа разбиения пространства параметров невозможно оценить значимость характеристик. Особенности рассматриваемой задачи заключаются в следующем:

- используется более двух вариантов значения выбора – несколько возможных линеек;
- требование минимизации функционала означает разную возможную стоимость ошибки при выборе неоптимального решения даже для отдельно взятой процедуры;
- построенные характеризующие параметры различны по диапазону и распределению, при этом их качественную нормировку для формирования метрики невозможно провести без оценки влияния на эффективность решения.

Наиболее часто используются для задач классификации методы, позволяющие повысить вероятность выбора оптимального решения, такие как метод опорных векторов (SVM, support vector machine) и Байесовские сети [6]. В случае неоптимального выбора они дают возможность учесть зависимость веса ошибки от значимости события, но не учитывают возможность разной стоимости ошибки для промаха и не работают с общими

функционалами.

Были также изучены возможности использования методов кластеризации [7], которые позволяют провести разбиение пространства без обучения при наличии разделяемых плотных участков. В результате был разработан промежуточный метод разбиения пространства, аналогичный кластеризации с ядрами, при котором уменьшается расстояние до минимума функционала.

В рассматриваемом случае теоретическую задачу разбиения можно сформулировать следующим образом.

Пусть: $P = \{p_1,\ldots, p_n\}$ – набор рассматриваемых процедур; H – пространство параметров, т.е. построено отображение $Ch: P \to H$; $L = \{l_1,\ldots, l_k\}$ – набор линеек – и задан функционал $F(l(p_1,\ldots, l(p_n)) \to \mathbb{R}$, определенный на множестве L^n, где $l: P \to L$. Разбиением на кластеры пространства H будем называть выделение областей в H с одновременным выбором постоянного для каждой области отображения $l(Ch^{-1}(H))$. Требуется построить такое разбиение на m кластеров, на котором достигается минимум $F(l(p_1,\ldots, l(p_n)) \to \min$ по всем таким разбиениям. Для обеспечения статистической значимости разбиения дополнительно потребуем, чтобы для каждой процедуры p существовала область, содержащая не менее q элементов, использование на которой линейки $l(p)$ не приводит к увеличению значения функционала по сравнению с базовым. Под базовым подразумевается значение функционала на линейке -O3, являющейся оптимальной в среднем.

Практическая реализация выбора кластера должна также учитывать дальнейшее его использование в компиляторе с учетом требования минимизации времени компиляции, т.е. определение принадлежности процедуры с набором параметров кластеру не должно иметь большую вычислительную сложность, поэтому в качестве областей были выбраны прямоугольные параллелепипеды.

Предлагаемый алгоритм выделения областей для фиксации значений линеек выглядит следующим образом:

1. Методом покоординатного спуска вычисляется точка, на которой функционал качества достигает минимума $F_{\min}$. Точка минимума в рассматриваемом случае оказалась достаточно устойчивой, т.к. из всех точек с константным выбором линейки для всех процедур она была достигнута не более чем за три шага.

2. Рассчитывается *матрица ошибок*, т.е. отклонения функционала от минимального значения для всех процедур и всех линеек:

$$\Delta(p_i, l_k) = \log\left(\frac{F(l_{\min(p_1)}, (l_{\min(p_2)}, ..., l_k, ..., (l_{\min(p_n)})}{F_{\min}}\right),$$

где l_k – назначение линейки для p_i. Стоит отметить, что в каждой строке построенной матрицы есть нулевой элемент, поскольку $\Delta(p_i, l_{\min(p_i)}) = 0$. Для определенности всем процедурам назначается базовая линейка, значение F в этой точке обозначается как F_{start}.

3. Далее следует итерационный алгоритм до выделения m кластеров:

А. Выбирается не помеченная флагом процедура p с максимальной ошибкой на назначенной линейке, обозначим $l_{\text{cur}} = l_{\min(p)}$.

Б. Строится прямоугольный параллелепипед в H, содержащий $Ch(p)$, на котором назначение линейки позволяет максимально уменьшить значение функционала по сравнению с текущим. Было опробовано несколько способов построения такого параллелепипеда, наиболее эффективным оказался двоичный поиск границ между $Ch(p)$ и граничными значениями параметров в H. Дополнительно на каждом шаге производится проверка того, что в текущей области не менее q точек из $Ch(p)$. Если не удается построить область с количеством точек не менее q, на которой значение функционала уменьшается по сравнению с текущим, то процедура помечается флагом и осуществляется переход в А.

В. Если полученное при выделении области изменение функционала больше, чем $t*F_{\text{start}}$, где $t = 0{,}01$ – эвристический параметр, то процедурам в выделенной области назначается линейка l_{cur}, а границы сформированной области запомина-

ются как кластер. Снимаются флаги с ранее помеченных процедур, и осуществляется переход в А.

Заметим, что построенные кластеры могут иметь пересечения. Для того чтобы определить принадлежность процедуры кластеру, выбирается последний построенный кластер, в который укладываются характеризующие эту процедуру параметры.

В приведенном алгоритме для определения эффективности построенного кластера используется эвристический параметр t. В теории можно было бы построить кластеры для каждой процедуры, а не для процедуры с максимальной ошибкой, и выбрать кластер, максимально уменьшающий значение F, но у этого решения слишком большая вычислительная сложность. В качестве альтернативы был взят не связанный с ограничением эффективности кластера подход, при котором на каждом шаге строятся кластеры для нескольких процедур с наибольшими ошибками, а из них оставляется оптимальный. Результат такого разбиения оказался практически идентичен приведенному.

4. Результаты применения СНОП при компиляции

Задача классификации решалась с использованием функционала (1) при значении параметра $r = 7$. С помощью описанного алгоритма выделения областей для фиксации значений линеек было построено 10 классов с требованием наличия не менее $q = 10$ элементов в каждом. На рис. 1 показан результат применения СНОП, представленный в сравнении с базовым режимом компиляции.

Для задач с целыми вычислениями (пакет CINT) было получено среднее ускорение времени исполнения на 2,9% и времени компиляции на 26,7%. Основной эффект достигался за счет сокращения объема работы, увеличивающей размер кода и повышающей спекулятивность оптимизаций на больших процедурах с большой степенью ветвлений. Задачи пакета CFP, т.е. задачи с плавающими вычислениями, в среднем ускорились на 8,2% по исполнению, причем среднее время компиляции сократилось на 6,7%. В этом случае основное ускорение связано с полученной возможностью применить оптимизации

работы памяти, которые увеличивают время планирования, но позволяют существенно уменьшить количество блокировок по чтению.

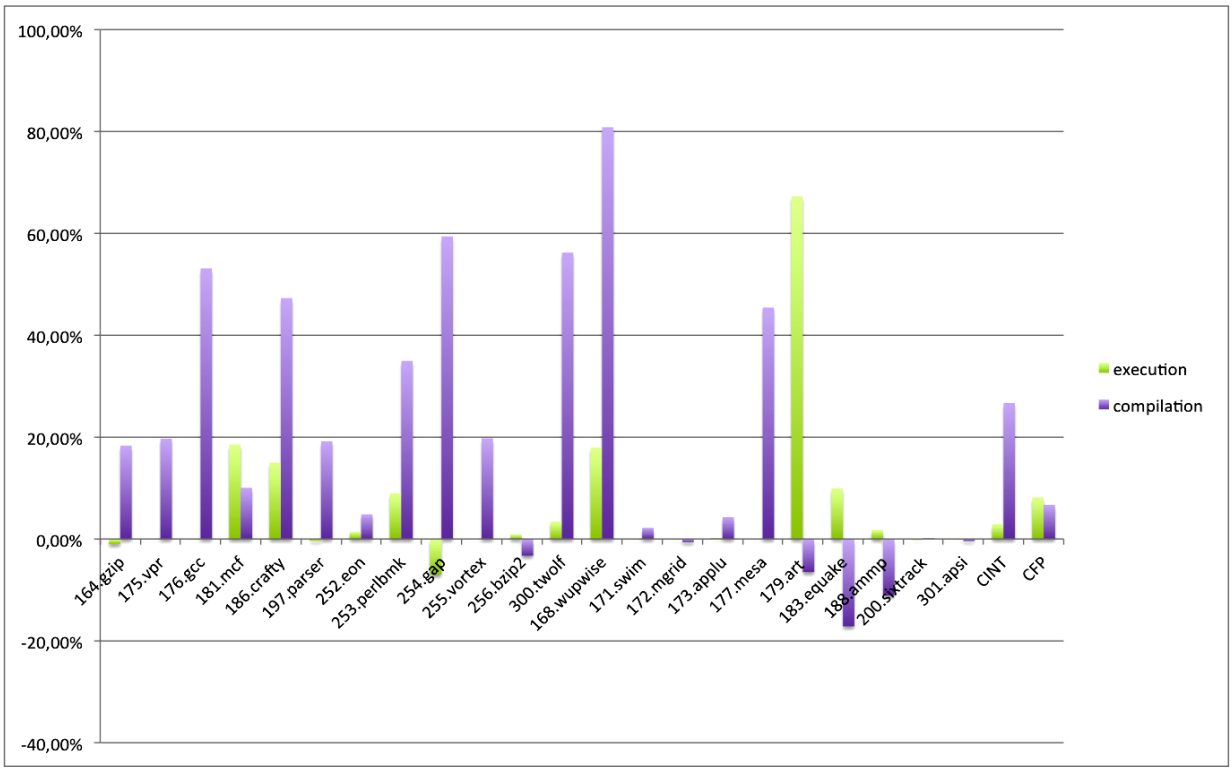

Рис. 1. Сравнение режима компиляции СНОП с базовым режимом компиляции: execution – время исполнения, compilation – время компиляции, CINT – среднее для задач с целыми вычислениями, CFP – среднее для задач с плавающими вычислениями

Заключение

В статье описана разработанная авторами система направленной оптимизации процедур, выбирающая эффективную последовательность оптимизаций для каждой процедуры на раннем этапе компиляции за счет статистической классификации. Предложенная система и разработанные для нее методы были реализованы и применены в компиляторе для процессора Эльбрус с архитектурой VLIW. Достигнутые ускорение исполнения и уменьшение времени компиляции дали основание для внедрения СНОП вместо базового режима компиляции. Дальнейший интерес представляют исследование возможности автоматического построения набора линеек с последующим статическим выбором, анализ оптимальности классификации и расширение статистической базы

Авторы выражают благодарность Линчику М.И. за проделанную работу по внедрению решателя в компилятор.

Литература

1. Spyridon Triantafyllis, Manish Vachharajani, Neil Vachharajani, David I. August. Compiler optimization-space exploration. Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization, March 23-26, 2003, San Francisco, California.

2. Prasad A. Kulkarni, David B. Whalley, Gary S. Tyson. Evaluating Heuristic Optimization Phase Order Search Algorithms. Proceedings of the International Symposium on Code Generation and Optimization, p. 157-169, March 11-14, 2007.

3. Suresh Purini, Lakshya Jain. Finding good optimization sequences covering program space. Transactions on Architecture and Code Optimization (TACO), January 2013.

4. M. Haneda, P.M.W. Knijnenburg, H.A.G. Wijshoff. Generating new general compiler optimization settings. Proceedings of the 19th annual international conference on Supercomputing, June 20-22, 2005, Cambridge, Massachusetts.

5. Prasad A. Kulkarni, Michael R. Jantz, David B. Whalley. Improving both the performance benefits and speed of optimization phase sequence searches. LCTES '10 Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems, April 2010.

6. Judea Pearl, Stuart Russell. Bayesian Networks. UCLA Cognitive Systems Laboratory, Technical Report (R-277), November 2000.

7. Jain, Murty and Flynn: Data Clustering: A Review, ACM Comp. Surv., 1999.