УДК 519.685.3

Четверина О.А., Нейман-заде М. И., Степанов П. А.

(ЗАО “МЦСТ”)

Система направленной оптимизации (СНОП).

Разработана система автоматического повышения производительности и ускорения компиляции за счет направленной оптимизации процедур (СНОП). Описаны механизм реализации набора стратегий, функционал качества выбора и задача автоматического выбора стратегии по вычислимым параметрам. Приведены результаты применения СНОП на пакетах задач.

Ключевые слова: СНОП, оптимизирующий компилятор, линейка компиляции, directed optimization system

Введение

Оптимизирующие компиляторы вносят существенный вклад в производительность современных вычислительных систем. Они обеспечивают прирост производительности за счет более удачного планирования кода и повышения эффективности работы с памятью. Особенно важное значение оптимизация кода имеет для VLIW архитектур, поскольку в этом случае производительность обеспечивается за счет одновременного исполнения сразу нескольких команд в одном такте. Это приводит к постоянному усложнению соответствующих компиляторов. Так, компилятор, разрабатываемый для процессора Эльбрус, в процессе базового режима выполняет более 300 этапов оптимизации кода. Это позволяет повысить производительность, но требует значительных временных затрат. Кроме того, остается проблема использования ряда оптимизаций, полезных на редких контекстах Они либо применяются только при возведении дополнительных опций, либо на ряде задач приводят к замедлению работы кода. Нами была поставлена задача создания автоматической системы, позволяющей сократить время компиляции и повысить производительность за счет использования различных *стратегий*, то есть последовательностей преобразований, для компиляции процедур.

К существующим системам автоматической настройки применения оптимизирующих преобразований относятся итеративные системы и системы с многократной компиляцией [3]. Итеративные системы требуют многократного исполнения на представительных данных, что не позволяет использовать их в базовом режиме. Системы с многократной компиляцией используют оценку эффективности полученного кода с помощью предсказания времени работы с учетом счетчиков исполнения, тренировочного запуска и особенностей

архитектуры [1], [5], [7]. Этот механизм частично используется в компиляторе для архитектуры Эльбрус при принятии решения по применению таких значимых преобразований, как unroll и overlap, но применение его в большем объеме повысит уже имеющиеся большие затраты на компиляцию. Проводятся также исследования по предварительному созданию ограниченного набора оптимальных настроек для множества задач [6], хотя на данный момент для них нет реализованной системы автоматического выбора. Дополнительный интерес для нашей работы представляет исследование [2], показывающее теоретический потенциал получения существенного ускорения компиляции практически без деградаций.

В представленном докладе описана разработанная *система направленной оптимизации* (СНОП), позволяющая одновременно получить прирост по производительности и ускорить компиляцию. В разделе 1 затронут вопрос создания стратегий компиляции. Далее, в разделе 2 построен функционал качества выбора стратегии и найден его теоретический минимум на разработанных стратегиях. И, наконец, в разделе 3 будет описана автоматическая система выбора и представлены полученные результаты.

1 Разработка стратегий компиляции.

Для разработки стратегий компиляции была создана система, позволяющая изменять и использовать последовательность и настройки не только оптимизирующих преобразований, но и различных технических и аналитических этапов между ними. В качестве основы этой системы был разработан механизм использования компилятором подаваемых *оптимизационных линеек.* Здесь и далее под оптимизационной линейкой будет подразумеваться последовательность всех производимых в процессе компиляции шагов.

Одна из сложностей при выборе наборов оптимизаций состояла в том, что работа многих из них сильно взаимосвязана. Применение одновременно двух различных преобразований может приводить к возникновению *эффекта интерференции*, который не может быть предсказан по результатам их независимого применения. Пример интерференции можно увидеть на Рис.1. На нем приведены результаты отдельного применения двух опций, и их комбинации (both). Этот эффект затрудняет применение методов внешней настройки таких как Ortogonal Arrays [4]. В то же время он

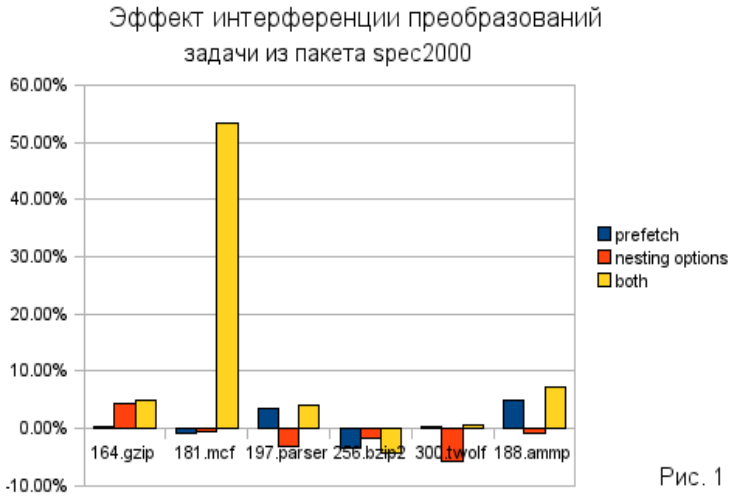

Рис. 1

может быть успешно преодолен при предварительном создании линеек.

Достижение максимальной производительности, кроме настройки работы оптимизации, обычно обеспечивается за счет использования расширенных режимов. К ним относится компиляция в режиме межмодульной оптимизации, при которой компилятору становится доступна информация одновременно обо всех модулях, и режим использования тренировочного профиля исполнения, при котором производится предварительных запуск на представительных тренировочных данных. Использование расширенных режимов ограничено. Компиляция в режиме «вся программа» неприменима в случаях создания библиотек или других подключаемых модулей, а создание полноценных тренировочных данных становится отдельной серьезной задачей, поскольку их несоответствие дальнейшему использованию программы может привести к ошибочным решениям при оптимизации. Поэтому исследования по созданию СНОП велись в двух направлениях: использование пиковых оптимизаций при работе в расширенном режиме и ускорение компиляции и повышение производительности в базовом режиме при использовании статического профиля. Было выявлено, что оптимальные наборы опций для статического и реального профиля довольно существенно отличаются. Основная причина заключается в невозможности в статическом случае определить неисполняемые участки программы, в результате чего повышение спекулятивности чтений или других операций нередко приводит к отрицательному эффекту. В данном докладе основное внимание будет уделено по-модульному режиму компиляции со статическим профилем как наиболее востребованному большинством пользователей.

При создании набора оптимизационных линеек для работы со статическим профилем были созданы две направленные линейки: fpline — для задач с многоитерационными циклами и работой с памятью и intline — для задач с малоитерационными циклами и сложноразветвленным управлением. Кроме того, были использованы по-процедурные линейки, соответствующие компиляции в режимах -O2 и -O3. Основное отличие между последними — отсутствие в O2 большинства увеличивающих размер кода оптимизаций и меньшая спекулятивность создаваемого кода.

2 Функционал качества выбора стратегии.

Вместо обычной задачи выбора наиболее эффективной стратегии компиляции с точки зрения исполнения, рассматривалась задача одновременного улучшения производительности и сокращения времени компиляции задач. В этом случае оптимизационную линейку для компиляции нужно выбирать не по одному параметру, а по двум одновременно, то есть необходимо построить функционал, минимизация которого

будет соответствовать наилучшему выбору.

На рис. 2 построен график с использованием значений времен компиляции и спланированного по тренировочному профилю времени исполнения на процедурах spec2000 н 5-ти линейках. Начальная точка $\min$ - выбор наиболее быстрой по компиляции линейки. Далее увеличивается параметр t и для каждой процедуры выбирается линейка i с минимальным временем исполнения среди тех, которые удовлетворяют условию $\frac{comp[i]-comp[\min]}{exe[\min]-exe[i]}<t$. Как видно из графика, большая часть прироста производительности была достигнута в точке, которая соответствует примерно половине всего времени компиляции.

Рис.2

Был построен функционал качества F, минимизация которого соответствует оптимальной точке. Пусть есть вектор $(l_{i1},l_{i2}\dots l_{in})$, где n — число процедур, l_{ik} — номер стратегии для k-ой процедуры p_k. И пусть есть таблицы $comp[p_k,i]$, $exe[p_k,i]$ размерами $K\times I$, где K - число процедур, I - число стратегий. Тогда функционал качества:

$$F(l_{i1},l_{i2},\dots l_{in})=(\sum_k exe[p_k,l_{ik}])^3*(\sum_k comp[p_k,l_{ik}])$$

В случае, если требуется уменьшать только время исполнения, а время компиляции не представляет интереса, вместо построенного функционала можно использовать сумму времени исполнения:

$$F(l_{i1},l_{i2},\dots l_{in})=\sum_k exe[p_k,l_{ik}]$$

3 Создание системы автоматического выбора стратегии.

После разработки стратегий и выбора функционала, позволяющего оценить качество выбора стратегий на множестве процедур, главной задачей становится создание автоматической системы выбора. Как упоминалось выше, для выбора оптимальных набора опций в ряде случаев используется оценка времени исполнения по планированию операций и оценочным счетчикам их исполнения. У этой оценки есть ряд недостатков:

- Ее практически невозможно использовать для оценки возможных блокировок по памяти. Оптимизации cache-opt, list-prefetch всегда признаются невыгодными при такой оценке. Серьезность этого недостатка можно оценить по результатам замеров, показывающих, что

блокировки по памяти на задачах spec2000 достигают 67% времени исполнения.

- Она плохо помогает при работе со статическим профилем.
- Даже в случае наличия тренировочного профиля возникают проблемы с тем, что он недостаточно соответствует боевому исполнению, и с тем, что после части преобразований без дополнительных знаний о счетчиках всех путей невозможно точно скорректировать профильную информацию.

По указанным причинам была поставлена задача классификации процедур и выбора оптимальной линейки на начальном этапе компиляции по доступным параметрам самой процедуры. Задачу такого выбора можно разделить на два этапа: выбор набора параметров и создание системы выбора.

В качестве параметров были выбраны: число операции в процедуре, среднее оценочное число исполнения операций в процедуре, средний по числу операций размер узла, количество операций вызова, максимальная глубина вложенности циклов, количество операций с плавающей точкой, количество операций чтения поля. Рассматривались и другие параметры, но они оказались менее значимыми с точки зрения классификации процедур.

При наличии параметров задачу построения функции выбора оптимальной линейки можно поставить следующим образом:

Пусть есть таблицы размерами $K \times I$, где K - число процедур, I - число стратегий в множестве стратегий L, в которых для каждой процедуры p_k из множества P и для всех рассматриваемых оптимизационных линеек $\{l_1, l_2 \ldots l_I\}$ есть время компиляции и замеренное время исполнения $comp[p_k, i]$, $exe[p_k, i]$. Кроме того, каждой процедуре p_k соответствует точка в пространстве параметров R^J. И определен функционал $F(l_{i1}, l_{i2} \ldots l_{in})$ на множестве L^K. Нужно построить отображение из множества параметров в множество линеек $G: R^J \to L$ таким образом, чтобы на нем функционал $F(G(p_1), \ldots G(p_K))$ достигал минимума при условии, что для любой процедуры p_k в пространстве параметров есть некоторая окрестность, содержащая S точек p_s из P, для которой применение $G(p_K)$ не увеличивает среднее значение функционала по сравнению с базовым.

Поскольку критерием качества выбора является не вероятность выбора оптимальной линейки для отдельной процедуры, а значение общего функционала, то для решения задачи плохо подходят вероятностные модели, такие как Байесовские сети, или в явном нейронные сети. Кластеризация по метрике тоже невозможна, поскольку метрика в данном случае определена только при фиксации общей для процедур линейки. Вместо этого для выбора

линеек был разработан *алгоритм кластерной минимизации ошибки*.

В результате применения алгоритма на разработанных линейках на основе процедур пакета spec2000 были выделены 5 дополнительных кластеров. Применение соответствующих кластерам линеек дает 17% ускорения компиляции и 8,5% ускорения производительности в среднем на задачах spec2000 (см Рис. 3).

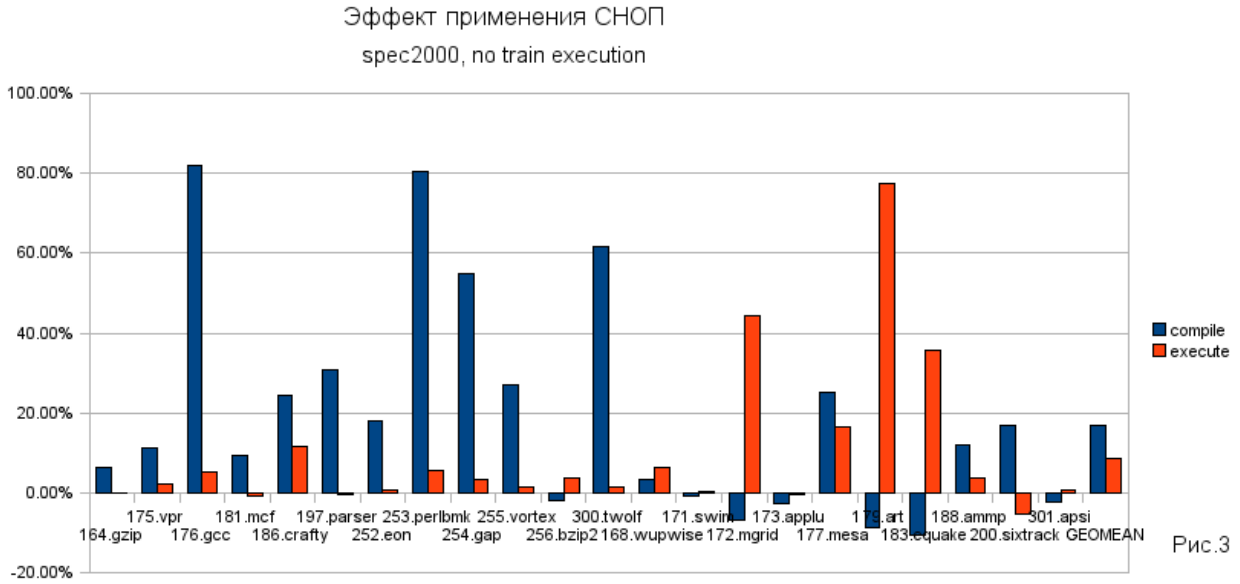

Рис.3

В качестве тестовой базы разработанной системы использовались задачи пакета spec95, не использовавшиеся при создании кластеров. В результате применения СНОП на spec95 в среднем на 16% ускорилась компиляция и на 3% уменьшилось время исполнения (см Рис. 4).

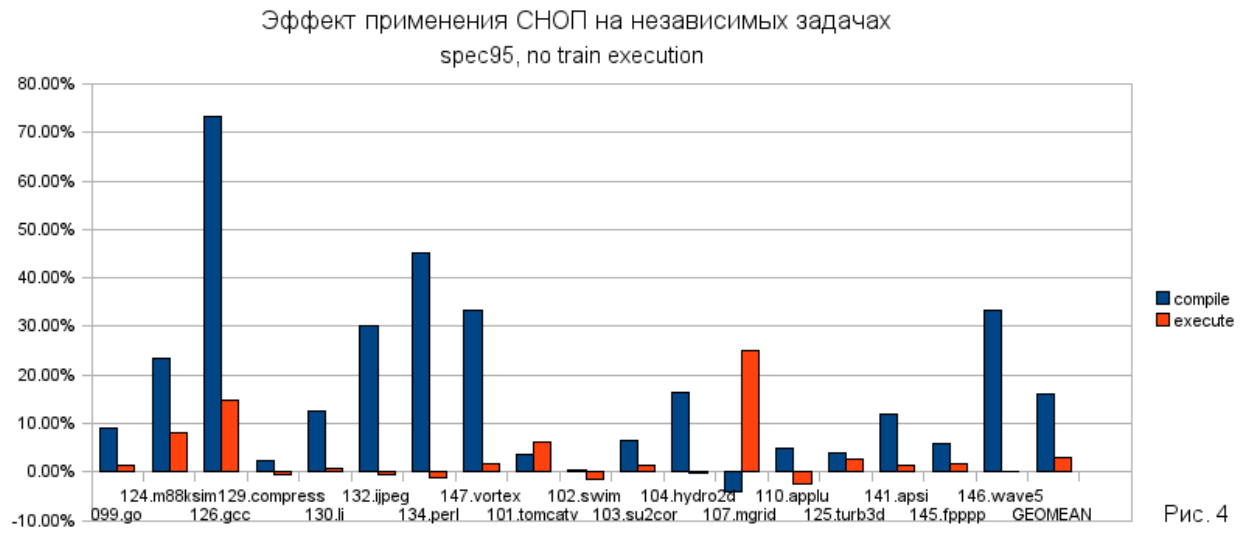

Рис. 4

Выводы.

В докладе описывается разработка последовательности направленных оптимизаций и создание для них системы автоматического выбора СНОП. Предложенный механизм

позволяет добиться прироста производительности с одновременным сокращением времени компиляции, что показано на пакетах задач spec95 и spec2000. Система направленной оптимизации реализована в составе оптимизирующего компилятора для микропроцессоров Эльбрус-3М, Эльбрус-3S, Эльбрус-2S. Работы над эффективностью системы продолжаются и направлены в сторону увеличения обучающей базы, выявления дополнительных параметров классификации, разработки новых оптимизационных линеек и настройки механизма выделения кластеров.

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. 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
3. Zhelong Pan , Rudolf Eigenmann, Rating Compiler Optimizations for Automatic Performance Tuning, Proceedings of the 2004 ACM/IEEE conference on Supercomputing, p.14, November 06-12, 2004
4. R. P. J. Pinkers , P. M. W. Knijnenburg , M. Haneda , H. A. G. Wijshoff, Statistical Selection of Compiler Options, Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04), p.494-501, October 04-08, 2004
5. 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
6. Keith D. Cooper , Alexander Grosul , Timothy J. Harvey , Steven Reeves , Devika Subramanian , Linda Torczon , Todd Waterman, ACME: adaptive compilation made efficient, ACM SIGPLAN Notices, v.40 n.7, July 2005
7. Suresh Purini, Lakshya Jain, Finding good optimization sequences covering program space, Transactions on Architecture and Code Optimization (TACO), January 2013
8. 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