УДК 004.4'422

Д.А. Земляков (АО «МЦСТ», ПАО «ИНЭУМ им. И.С. Брука»)

D. Zemliakov

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ ЦИКЛОВ В ОПТИМИЗИРУЮЩЕМ КОМПИЛЯТОРЕ

SOFTWARE PIPELINING IN OPTIMIZING COMPILER

Программная конвейеризация циклов является важным методом для максимального использования параллелизма программ с целью повышения их производительности. Среди различных методов программной конвейеризации, рассмотренных в статье, можно отметить алгоритм EPS как сопоставимый по эффективности с другими подходами, но лишенный ряда их недостатков. В статье предложена реализация этого алгоритма в оптимизирующем компиляторе, разработанном для семейства вычислительных средств «Эльбрус» применительно к архитектуре SPARC. Приводятся результаты практического использования метода.

Software pipelining is an important way to improve performance by increasing parallelism. There is a quite universal EPS algorithm among the various techniques in this paper, that distinguished them. It is comparable effective but devoid of many disadvantages of other approaches. A realization of the algorithm for optimizing compiler for the SPARC architecture is provided. Practical results of software pipelining are presented.

Ключевые слова: оптимизирующий компилятор, параллелизм на уровне операций, оптимизация, программная конвейеризация циклов.

Key words: optimizing compiler, instruction level parallelism, optimization, software pipelining.

Введение

Методика распараллеливания циклов имеет большое значение для повышения производительности вычислительного процесса во многих приложениях ввиду того, что исполнение циклов занимает значительную часть общего времени исполнения программы. Различными исследователями было предложено и реализовано множество вспомогательных техник на уровне компилятора, которые в случае сильной зависимости по данным внутри цикла оказались малоэффективны. В этом случае особенно актуальна программная конвейеризация циклов, которая позволяет решить проблему [1, 2].

Программная конвейеризация перестраивает циклы таким образом, что операции с различных итераций перемешиваются и исполняются одновременно. В итоге такой подход создает некоторую параллельность, отсутствующую на отдельно взятой итерации. Использование подобного принципа важно для машин, у которых присутствуют несколько функциональных устройств, способных работать одновременно [1, 2], как в разрабатываемых АО «МЦСТ» микропроцессорах с архитектурами SPARC и «Эльбрус» [3].

1. Технология программной конвейеризации циклов

1.1. Специфика процесса

Программная конвейеризация представляет собой программный аналог метода, используемого для планирования аппаратных конвейеров. За счет оптимизации преобразуется тело исходного цикла таким образом, что операции с различных итераций в допустимых случаях выполняются одновременно. В итоге при таком преобразовании планируется цикл целиком, чтобы обеспечить максимальное использование возможностей параллельных вычислений. Последовательность операций для заполнения конвейера называется прологом, тело конвейеризованного цикла представляет собой устойчивое состояние, а последовательность операций, приводящая к опустошению конвейера, называется эпилогом.

Производительность цикла обратна интервалу времени между запусками итераций, поэтому целью программной конвейеризации является планирование цикла с наименьшим

возможным интервалом запуска. При этом учитываются ресурсное ограничение, накладываемое доступными функциональными устройствами, и зависимости по данным между операциями цикла, в частности, различные экземпляры одних и тех же операций с разных итераций.

В конечном счете, от программной конвейеризации требуется максимальное ускорение долго выполняющихся циклов в процедуре, избегая при этом значительного роста кода [1].

1.2. Методы программной конвейеризации

Большинство алгоритмов программной конвейеризации может быть отнесено либо к алгоритмам модульного планирования (modulo scheduling), либо к алгоритмам распознавания устойчивого состояния (kernel recognition).

При модульном планировании операции исходного цикла планируются по очереди с учётом зависимостей. Проблема оптимального планирования с наименьшим возможным интервалом запуска итераций конвейеризованного цикла является NP-сложной задачей. Поэтому на практике сначала определяется некоторое минимальное значение интервала запуска, основанное на анализе доступных ресурсов и зависимостей между операциями исходного цикла. После этого итеративно проводятся попытки спланировать цикл в указанное количество тактов, которое в случае неудачи увеличивают [1, 4].

Исследователями были предложены различные дополнения и модификации алгоритма модульной программной конвейеризации [4]. Например, M. Lam использовала иерархический подход (hierarchical reduction), при котором сильно связанные компоненты графа планируются отдельно. При этом исходная задача делится на более мелкие подзадачи, для каждой из которых проводится конвейеризация. A.M. Zaky была предпринята попытка строго математически описать задачу программной конвейеризации цикла (path algebra) с использованием матрицы, каждое значение в которой показывает минимальное время по зависимостям. Улучшенное модульное планирование (enhanced modulo scheduling) исполь-

зует if-conversion, чтобы устранить все ветвления и представить операции в одном линейном участке с предикатным кодом. Поворотное модульное планирование (Swing Modulo Scheduling) ставит своей целью максимизацию частоты выдачи новых итераций и одновременно минимизацию регистрового давления.

Другой подход состоит в том, чтобы операции с нескольких итераций исходного цикла спланировать вместе и распознать в полученном коде сформировавшееся устойчивое состояние. Основная идея состоит в том, чтобы одновременно просмотреть несколько итераций и попробовать скомбинировать операции, между которыми нет зависимостей по данным.

Здесь также были предложены различные подходы [4]. Метод Perfect Pipelining проводит предварительное планирование, раскрутку и перемешивание итераций цикла, допуская независимый перенос операций после предварительного планирования. Алгоритм поддерживает работу с множественными ветвлениями и концептуально важен тем, что не вынуждает рассматривать все пути в цикле одновременно. У алгоритма на сетях Петри (Petri Net Model) можно отметить строгую математическую ориентацию и возможность адаптации к различным ограничениям, однако имеется ряд сложностей с определением устойчивого состояния. Теоретически интересен алгоритм Vegdahl, в котором предполагается рассмотреть все варианты и выбрать лучший. Так как он имеет экспоненциальную сложность, то не применим на практике для больших задач, однако может оказаться оптимальным для небольших циклов.

1.3. Enhanced pipeline scheduling

Предложенный K. Ebcioglu и T. Nakatani метод улучшенной программной конвейеризации (Enhanced pipeline scheduling, EPS) [4, 5] отличается тем, что, будучи применим к циклам общего вида, всегда поддерживает корректное состояние цикла, не делая попыток перестроить его целиком. Как и в прочих методах, для получения наилучшего результата совмещается трансформация тела цикла с планированием.

Положенный в основу метода алгоритм основан на пошаговом переносе операций из головы и прочих узлов цикла по обратной дуге в его хвост, как это показано на рис. 1, и поэтому поддерживает корректную структуру. При переносе операции происходит её перемещение на итерацию вверх, в результате чего исходная операция одновременно копируется и в пролог цикла, и по обратной дуге в хвост цикла, как операция с последующей итерации. Эффект от подобных перемещений зависит от доступности ресурсов и зависимостей по данным между операциями. Таким образом, для формирования устойчивого состояния используется исходный цикл, поэтому распознавать устойчивое состояние не требуется.

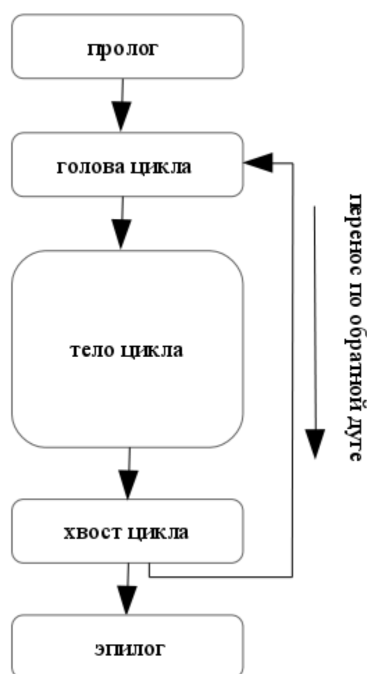

Рисунок 1. Пример структуры простого цикла и направление переноса операций по обратной дуге

Конвейеризация происходит при переносе операции по обратной дуге, когда та копируется в хвост цикла, как операция последующей итерации. Работа продолжается, пока не будут рассмотрены все межитерационные зависимости. Пролог и эпилог строятся автоматически при переносе операций.

Алгоритм состоит из двух основных фаз, циклически сменяющих друг друга. В пер-

вой фазе происходит выборка операций для второй. Первые в исполнении операции, не имеющие предшественников по зависимостям в теле цикла, формируют рабочее множество. Все прочие операции зависят от них, поэтому не могут быть перенесены по обратной дуге на итерацию выше. Таким образом, шаг алгоритма ограничивается операциями рабочего множества. Во второй фазе операции рабочего множества переносятся по обратной дуге. Операции, которые выносятся из цикла, формируют пролог. Операции, которые переносятся вниз по обратной дуге, оказываются операциями следующей итерации. Процесс продолжается, пока есть доступные для переноса операции.

Пример работы данного алгоритма представлен на рис. 2. Операции связаны истинными зависимостями, поэтому сначала в рабочее множество попадает только операция А, после чего проводится перенос. Версия операции А с текущей итерации оказывается в прологе, а в теле цикла появляется операция А со следующей итерации. Затем описанные шаги повторяются. В рабочее множество попадает одна операция из текущей итерации и одна или несколько из последующих итераций. В итоге, когда все зависимости рассмотрены и все переносы выполнены, операции с различных итераций оригинального цикла оказываются в одной итерации модифицированного цикла, что и означает его конвейеризацию.

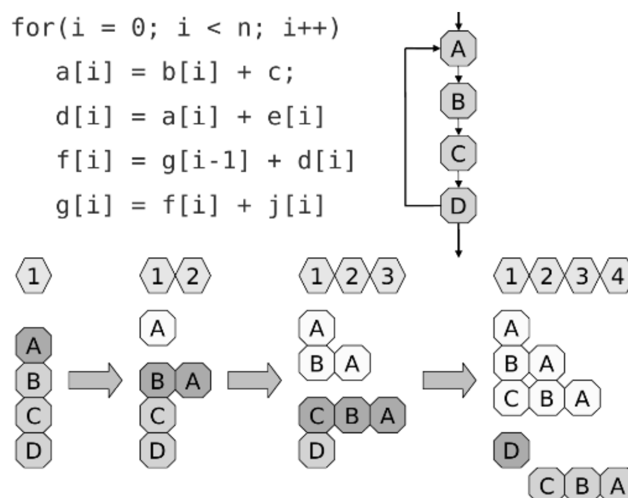

Рисунок. 2. Enhanced pipeline scheduling. Исходный цикл, этапы заполнения рабочего множества алгоритма и переносов операций в пролог и по обратной дуге

Таким образом, при программной конвейеризации цикла с помощью данного метода не требуется ни проводить итеративные попытки планирования с целью уложиться в определенное число тактов, ни раскручивать несколько итераций и пытаться распознать устойчивое состояние. На каждом этапе преобразования сохраняется корректное состояние цикла в отличие от полного перестроения цикла в остальных алгоритмах. Конвейеризация происходит по мере переноса операций по обратной дуге.

2. Реализация алгоритма программной конвейеризации в оптимизирующем компиляторе

Среди рассмотренных выше методов программной конвейеризации, реализуемых в оптимизирующем компиляторе, для архитектуры SPARC в силу своих достоинств был выбран алгоритм Enhanced pipeline scheduling.

2.1. Анализ контекста на возможность применения оптимизации

Оптимизация выполняется для конкретных циклов, поэтому ее можно применять к отдельным процедурам исходного кода. Первым этапом оптимизации традиционно является анализ контекста, который работает с графом потока управления и деревом циклов текущей процедуры. Среди всех циклов процедуры выбираются наиболее вложенные, т.е. не содержащие внутри других циклов. К каждому из них последовательно применяются проверки структуры, операций и количества итераций цикла.

Проверка структуры предполагает выполнение следующих условий:

- цикл должен быть сводимым, т.е. имеющим только две входные CFG-дуги, одна из которых обратная;
- цикл не должен иметь более одной выходной дуги;
- цикл не должен иметь более одной обратной дуги, т.к. при наличии в нем других обратных дуг не гарантируется сохранение корректного состояния цикла.

При ***проверке операций цикла*** необходимо убедиться в том, что:

- их количество превышает некоторое минимальное фиксированное значение и одновременно не превышает некоторое максимальное фиксированное значение;
- в цикле отсутствуют ассемблерные вставки.

Статическая ***проверка итераций*** позволяет проконтролировать, что известное из анализа цикла количество итераций гарантированно превосходит некоторую пороговую величину, так как:

- при выполнении оптимизации происходит перемешивание операций с различных итераций, и в случае, когда число итераций цикла окажется недостаточным, может произойти перемешивание операций с отсутствующей итерации, вследствие чего корректное состояние цикла после оптимизации гарантировать нельзя;
- при создании пролога и эпилога цикла происходит дублирование кода, и при малом числе итераций негативный эффект от роста кода может превысить пользу от программной конвейеризации.

2.2. Версионирование цикла

В случае, когда количество итераций статически неизвестно, при выполнении ряда условий можно воспользоваться динамической проверкой. При этом перед циклом достраиваются операции, позволяющие вычислить число итераций цикла, а тело исходного цикла дублируется. Одна версия оптимизируется, другая – нет. Версионирование выполняется по базовой индуктивности цикла, которая определяет количество итераций цикла и должна иметь вид *induct_var = oper(induct_var, const)*, где *induct_var* – базовая индуктивность, *const* – константа, *oper* – операция сложения или вычитания.

Версионирование проводится в том случае, если:

- рассматривается исчислимый цикл, т.е. с помощью построения дополнительных операций удастся определить число его итераций;
- одна из версий окажется конвейеризована. Так как по завершении анализа прини-

мается решение о копировании цикла и построении ряда дополнительных операций, что увеличит исходный размер программы, то надо убедиться, что данный цикл удастся конвейеризовать.

После проведения анализа формируется вторая версия цикла. Для этого находится или создаётся пролог исходного цикла, в котором строятся операции, вычисляющие количество итераций цикла. Кроме того, в прологе строится операция сравнения вычисленного и минимально допустимого количества итераций. На основе этих вычислений определяется, с какой версией цикла продолжится работа в программе при исполнении. Затем выполняется копирование цикла с корректировкой всех вспомогательных структур. Создается ветвление управления на основе операции сравнения.

2.3. Конвейеризация

После того как были выполнены все проверки и, возможно, осуществлено версионирование, к успешно прошедшим анализ циклам применяются следующие действия:

- определяется критический путь цикла, для всех операций которого время раннего планирования совпадает со временем позднего планирования. С данными операциями продолжается дальнейшая работа;
- определяется длина ресурсного планирования цикла. На основании правил совместимости операций в команде для архитектуры SPARC, основанных на доступных функциональных устройствах, вычисляется минимальное время исполнения цикла;
- формируется список операций рабочего множества из операций критического пути цикла согласно алгоритму, представленному на рис. 2;
- проверяется эффективность текущего шага алгоритма;
- операции рабочего множества переносятся на итерацию вверх в пролог и в хвост цикла по обратной дуге с коррекцией всех необходимых аналитических структур;
- в случае необходимости создаётся эпилог цикла, операции для которого формировались в течение всей оптимизации.

Первые два этапа выполняются перед основным циклом оптимизации. Если результат ресурсного планирования превосходит длину вычисленного критического пути цикла, то в цикле уже присутствует достаточная параллельность. Поэтому выполнять дальнейшее преобразование не имеет смысла.

Отметим также, что перед тем как перейти к основному циклу оптимизации, в случае дальнейшего построения эпилога, следует определить операции, которые в дальнейшем необходимо будет скопировать в эпилог. В их число включаются все операции, записывающие или устанавливающие какое-либо значение или признак, а также операции, имеющие потокового преемника за пределами цикла. В дальнейшем, в случае участия этих операций в преобразовании, они могут исключаться из данного множества, а само множество – пополняться вновь созданными операциями.

Все шаги, кроме ресурсного планирования и создания эпилога, выполняются циклично до тех пор, пока алгоритму удается сформировать очередное непустое рабочее множество операций. После того как все возможности по переносу операций оказываются исчерпаны, формируется эпилог.

3.4. Анализ эффективности применения оптимизации

Перенос операций в некоторых случаях может не привести к улучшению планирования цикла. Чтобы избежать бесполезных действий, используется анализ эффективности шага алгоритма, выполняемый после формирования рабочего множества, но перед перестроением цикла. В ходе анализа временно создается копия изменяемой структуры оригинального цикла, где выполняется упрощенный перенос операций. По результатам последующего пересчета нового критического пути цикла либо выносится решение о выполнении данного шага, либо проводится моделирование последующих шагов алгоритма в копии с целью поиска возможного положительного эффекта.

5. Экспериментальные результаты

Оценка эффективности оптимизации была проведена на пакетах тестов производительности SPEC95, SPEC2000 и SPEC2006 [3]. Замеры времени исполнения задач без программной конвейеризации и с ее использованием производились на машинах с микропроцессорами МЦСТ R1000, совместимыми с архитектурой SPARC V9. Согласно данным, приведенным на рис. 3-5, эти эксперименты показали прирост производительности 1-3% с максимумом в 11%.

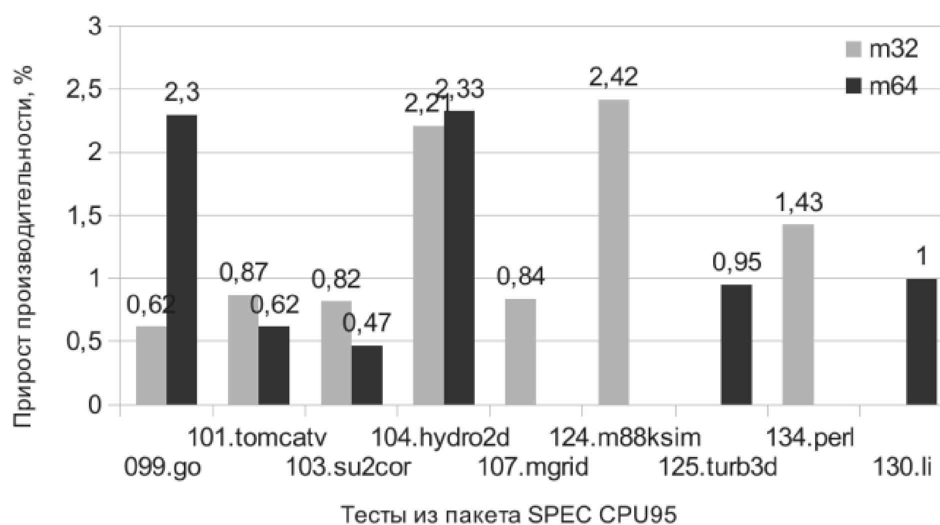

Рисунок 3. Ускорение времени исполнения задач из тестового пакета SPEC CPU95

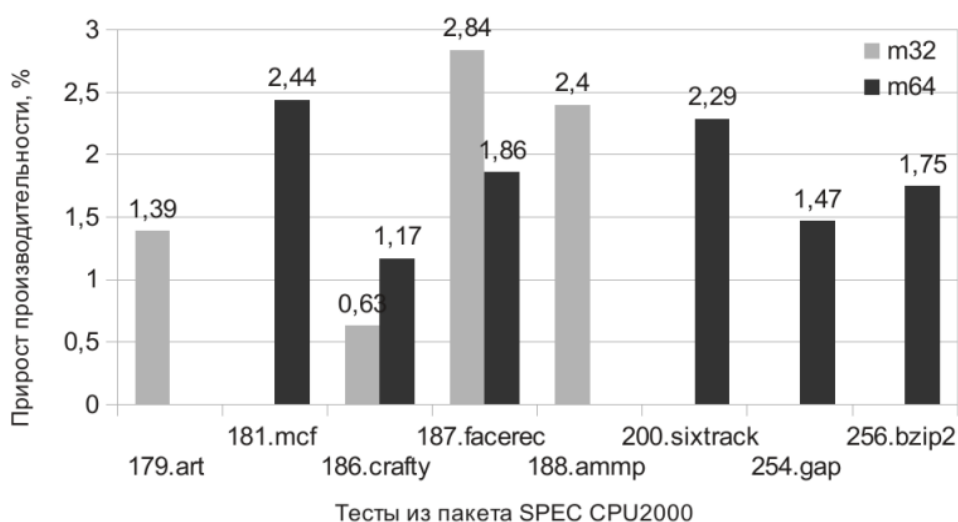

Рисунок 4. Ускорение времени исполнения задач из тестового пакета SPEC CPU2000

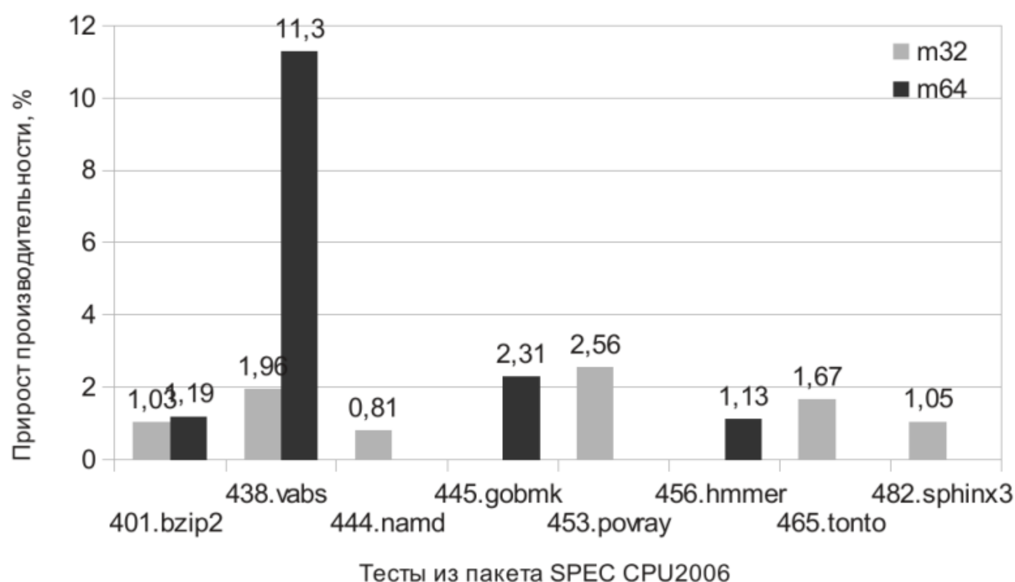

Рисунок 5. Ускорение времени исполнения задач из тестового пакета SPEC CPU2006

Заключение

В статье рассмотрена программная конвейеризация циклов для максимального использования параллелизма на уровне инструкций, кратко изложены различные подходы и методы. Среди возможных техник был отмечен алгоритм Enhanced pipeline scheduling как не уступающий в эффективности, но лишенный ряда недостатков прочих методов. Предложена реализация данного алгоритма для архитектуры SPARC. Оптимизация была внедрена в оптимизирующий компилятор, разрабатываемый АО «МЦСТ». Экспериментальные результаты показали ее эффективность.

В силу достаточной универсальности алгоритма после небольшой доработки он может быть использован и для других архитектур. В частности, реализованная оптимизация была адаптирована для использования в оптимизирующем компиляторе для архитектуры «Эльбрус». Для этого был проведен ряд доработок анализа циклов и переноса операций между итерациями, самыми значимыми из которых стали изменения правил для ресурсного планирования и расчёта длины критического пути цикла.

Литература

1. Альфред Ахо, Моника Лам, Рави Сети, Джеффри Ульман. Компиляторы: принци-

пы, технологии и инструментарий. Изд. 2-е. – М.: ИД «Вильямс», 2008. 1184 с.

2. Steven S. Muchnick. Advanced Compiler Design Implementation. Изд. 5-е. – San Francisco: Morgan Kaufmann Publishers, 1997. 856 с.

3. Ким А.К., Перекатов В.И., Ермаков С.Г. Микропроцессоры и вычислительные комплексы семейства Эльбрус // СПб.: Питер, 2013. 272 с.

4. V. H. Allan, R. B. Jones, R. M. Lee, and S. J. Allan. Software pipelining. ACM Comput. Surv., 27(3), 1995.

5. K. Ebcioglu, T. Nakatani. A New Compilation Technique for Parallelizing Loops with Unpredictable Branches on a VLIW Architecture. In D. Gelernter, editor, Languages and Compilers for Parallel Computing, p. 213-229. – MIT Press, Cambridge, MA, 1990.

6. Standart Perfomance Evaluation Corporation. The Spec Benchmark Suites. CPU-intensive benchmark suite. [Electronic resource]. – 1995-2015. http://www.spec.org/cpu.