ПРОГРАММИРОВАНИЕ, 2010, № 6, с. 74–80

УДК 681.3.06

РАСПРЕДЕЛЕНИЕ РЕГИСТРОВ ПРИ ПЛАНИРОВАНИИ ИНСТРУКЦИЙ ДЛЯ VLIW-АРХИТЕКТУР

© 2010 г. **Д. С. Иванов**
Московский физико-технический институт
141700 г. Долгопрудный, Институтский пер., 9
E-mail: dimanovs@gmail.com
Поступила в редакцию 11.09.2009 г.

Вопросы взаимодействия фаз распределения регистров и планирования инструкций довольно часто рассматриваются в публикациях, посвященных оптимизациям завершающего этапа компиляции. Как правило, авторы предлагают адаптировать одну из фаз под нужды другой, не объединяя их в единое целое. Тем не менее, их интеграция может существенно уменьшить время работы и улучшить производительность результирующего кода. В данной работе представлена попытка максимально полно объединить эти фазы с учетом особенностей статического планирования для VLIW-архитектур.

1. ВВЕДЕНИЕ

Одной из наиболее важных проблем вычислительной техники на сегодняшний день можно назвать невысокую эффективность взаимодействия памяти и микропроцессора. Начиная с 1980-го года, разрыв между скоростями каналов памяти и скоростями микропроцессоров неуклонно увеличивается [1]. Эффективное распределение регистров в оптимизирующем компиляторе позволяет обойти это узкое место, минимизируя количество создаваемых инструкций обращений в память.

2. ПРЕДЫДУЩИЕ ИССЛЕДОВАНИЯ

Общепринятые схемы, основанные на классическом алгоритме раскраски графа несовместимости [2], не учитывают связь распределения регистров с планированием инструкций.

Наиболее целостной попыткой интегрировать планирование и распределение регистров является разработка унифицированного представления для распределения ресурсов на основе PDG [3] с использованием иерархического распределителя регистров RAP [4]. Данная схема обладает рядом преимуществ, однако она требует построения PDG, имеет ограниченную область видимости (в пределах региона) и использует дополнительный проход для перемещения инструкций обращения в память.

В работе [5] комбинированная схема рассматривается только в пределах линейного участка. В работе [6] предлагается распределять регистры перед планированием с добавлением в граф несовместимости дуг, отображающих возможный параллелизм. Такой подход позволяет в некоторой степени учесть влияние фаз планирования и распределения друг на друга, однако соотношение влияния дуг, отображающих несовместимость переменных, и дуг, отображающих параллелизм, остается неочевидным.

В последних работах, таких, как [7–9], связи распределения регистров с планированием инструкций уделяется мало внимания. В работе [9] предложен алгоритм распределения регистров на спланированном коде по принципу складывания мозаичной головоломки, игнорируя влияние фазы планирования на распределение регистров. В работе [7] представлен итеративный распределитель, выполнение каждой итерации которого позволяет получить более оптимальное распределение регистров с точки зрения размера результирующего кода. Представленный алгоритм также не учитывает связь планирования инст-

рукций и распределения регистров, а результаты производительности, как правило, хуже результатов, полученных на распределителе, основанном на раскраске графа несовместимости. Авторы статьи [8] предлагают технику уменьшения давления на регистры перед планированием.

Пренебрежение связью между планированием и распределением регистров особенно ощутимо в архитектурах с широким командным словом. Распределение регистров перед статическим планированием инструкций накладывает серьезные ограничения на параллельность кода, а схема с планированием до распределения [10] приводит к вставке в код новых широких команд в случае нехватки регистров.

Недостаточно изученной является схема, в которой планирование инструкций и распределение регистров происходят одновременно. Такая схема позволила бы лучше учесть взаимные требования планирования и распределения, однако, несмотря на то, что разработки в этой области ведутся уже довольно долгое время, общепринятых алгоритмов, обладающих приемлемой алгоритмической сложностью, пока еще не появилось.

3. ОБЩЕЕ ОПИСАНИЕ АЛГОРИТМА

В дальнейшем под “сетью” будем понимать множество узлов и дуг управляющего графа процедуры [11], при исполнении которых необходимо хранить промежуточное значение переменной. В таком случае говорят, что переменная живет на данном множестве дуг и узлов. Каждой сети соответствует множество инструкций, записывающих или считывающих значение переменной. Первый тип инструкций назовем определениями сети, второй – использованиями сети.

Если переменная живет в пределах одного линейного участка, будем называть соответствующую сеть локальной, в противном случае – глобальной. После распределения регистров каждой переменной должен соответствовать один архитектурный регистр. Поддержка этого свойства для глобальных сетей при планировании линейных участков по отдельности затруднительна, так как может возникнуть ситуация, когда нескольким сетям в одном участке может потребоваться один и тот же архитектурный регистр. По этой причине распределение регистров для глобальных сетей следует выполнять перед планированием.

Переменные глобальных сетей, которым не хватило регистров, откачиваются в память после определений и считываются из памяти перед использованиями соответствующих сетей. Таким образом, каждая глобальная сеть распадается на множество локальных сетей. Локальным сетям регистры назначаются непосредственно при планировании соответствующих им инструкций.

3.1. Распределение регистров для глобальных сетей

При распределении регистров для глобальных сетей (далее “глобальное распределение”), выполняемом перед началом планирования, можно воспользоваться алгоритмом раскраски графа несовместимости Четина-Бриггса [12], обобщенный вариант которого описан в [13]. Отличие обобщенного алгоритма от классического заключается в различном количестве цветов (т.е. регистров), доступных для каждого узла графа (сети). Так как для последующего локального распределения необходимо резервировать некоторое число регистров, вообще говоря, различное для разных узлов управляющего графа, количество доступных регистров для разных глобальных сетей может заметно отличаться.

Одним из недостатков алгоритма с графом несовместимости является его сложность, поэтому было решено разработать более быстрый алгоритм, использующий для представления регистров и сетей битовые векторы. Каждый бит вектора соответствует узлу управляющего графа процедуры. Нулевое значение бита для вектора регистра означает, что на данный регистр в соответствующем биту узле не распределена ни одна сеть. Нулевое значение бита для вектора сети говорит о том, что ее переменная не является живой в соответствующем биту линейном участке.

Пусть N – количество узлов управляющего графа процедуры, а R – количество архитектурных регистров, тогда регистровый файл в нашем алгоритме будет представлять собой битовую матрицу размером $N \times R$.

Сеть заведомо можно распределить на регистр r, если конъюнкция r-го столбца матрицы и битового вектора сети дает нулевой вектор. В противном случае необходимо выполнить дополнительный анализ для каждого узла, соответствующего ненулевому биту результирующего вектора. Распределение нескольких сетей на один регистр в узле возможно, когда каждая из сетей не является живой при выполнении определения любой другой сети [11].

Если дополнительный анализ дает положительный результат для всех ненулевых битов, то сеть-кандидат может быть распределена на данный регистр. При распределении глобальной сети на регистр r единичные биты вектора сети копируются в r-ый столбец регистровой матрицы.

В случае, когда для глобальной сети не удалось найти регистр, она разбивается на несколько локальных сетей посредством создания минимально необходимого [14] количества инструкций откачки/подкачки.

3.2. Распределение регистров для локальных сетей

Распределение регистров по локальным сетям происходит непосредственно при планировании соответствующих им инструкций. В данной работе планирование осуществляется алгоритмом List Scheduling [15], однако ничто не мешает реализовать описываемую схему с использованием любого другого алгоритма планирования.

Локальные сети распределяются на свободные регистры при планировании своих первых определений. При планировании последнего использования регистр освобождается.

В случае возникновения дефицита регистров во время планирования создаются инструкции откачки/подкачки, которые планируются на общих основаниях, то есть с соблюдением необходимых задержек. При отсутствии свободных регистров вместо откачки одного из занятых регистров можно также отложить планирование нуждающейся инструкции. Как правило, задержка планирования инструкции имеет смысл, если в результате планирования другой инструкции будет освобожден один или несколько регистров. При этом не должен увеличиваться критический путь [1], иначе величина задержки инструкции должна быть меньше величины задержки инструкции чтения из памяти, восстанавливающей откачанный регистр, в противном случае откачка регистра будет более выгодной.

3.3. Соблюдение регистрового баланса

При раздельном распределении регистров для локальных и для глобальных сетей необходимо как можно точнее определить, сколько регистров выделить для распределения каждому типу в каждом линейном участке программы.

В часто исполняемых (горячих) участках программы откачка локальной сети неизбежно ведет к появлению пары инструкций откачки/подкачки. Глобальная сеть может быть живой на горячем участке, но не иметь в нем ни использований, ни определений. Откачка такой сети в память, то есть преобразование ее в несколько локальных сетей в редко исполняемых (холодных) линейных участках, не приведет к большим потерям производительности. В такой ситуации в горячих участках требуется отдавать предпочтение локальным сетям.

Несколько иначе обстоит дело с глобальными сетями, имеющими использования и определения в рассматриваемом горячем линейном участке. Откачка такой сети в память может привести к появлению инструкций откачки/подкачки в других горячих участках программы, где нет дефицита регистров. В этом случае выделение регистра глобальной сети предпочтительнее, однако в дальнейшем для простоты будем считать, что в горячих участках более приоритетными являются локальные сети.

Откачка локальных сетей в холодных участках не приведет к большим потерям в производительности, однако выделение большого количества регистров для локальных сетей может привести к откачке в память глобальных сетей в горячих участках. Поэтому в холодных участках локальным сетям нужно выделять минимальное количество регистров, необходимое для сходимости алгоритма, то есть равное максимально допустимому количеству регистровых операндов инструкции.

С большой точностью количество регистров, которые необходимо зарезервировать для локальных сетей в горячих участках, можно оценить, посчитав степень параллельности графа

зависимостей [15] для сетей. Степень параллельности, по сути, и есть максимальное количество локальных сетей, живущих одновременно. Для ее оценки можно воспользоваться следующей формулой:

$$LWN = \frac{\sum_i (TU(i) - TD(i))}{H},$$

где LWN – количество одновременно живых локальных сетей, $TU(i)$ – среднее арифметическое времен раннего и позднего запусков последнего использования локальной сети i, $TD(i)$ – среднее арифметическое времен раннего и позднего запусков первого определения локальной сети i, H – высота графа зависимостей линейного участка.

Резервирование конкретных регистров для локального распределения может привести к серьезным потерям в производительности. Например, зарезервировав три регистра для локального распределения, можно легко столкнуться с ситуацией, когда ни один из них не будет использован, при том, что будут созданы инструкции откачки/подкачки, сигнализирующие о нехватке регистров. Это объясняется тем, что локальному распределению хватило регистров, которые в данных узлах не были востребованы глобальными сетями. Соответственно, контроль количества регистров, выделяемых для локального распределения, должен осуществляться динамически при глобальном распределении.

4. ПРЕИМУЩЕСТВА АЛГОРИТМА

4.1. *Учет давления на регистровый файл*

Давлением на регистровый файл называется количество переменных, значение которых хранится на регистрах в данный момент времени. Планирование инструкций без контроля давления на регистры может привести к неоправданному растягиванию времени жизни регистров, что может привести к созданию лишних инструкций откачки/подкачки. Проблему можно решить, добавив в планирование построение сетей и механизм учета давления. Однако в случае наличия промежуточных фаз между планированием и распределением регистров сети придется перестраивать.

При комбинированном планировании и распределении информация о давлении на регистры всегда доступна. Для минимизации количества обращений в память, начиная с некоторого критического давления, можно определять приоритет планирования инструкций, учитывая их вклад в давление на регистровый файл. Таким образом, планирование будет работать в двух режимах: в штатном режиме при отсутствии дефицита регистров и в режиме учета давления, когда регистров будет не хватать.

4.2. *Оптимизация задержек при обращении в память*

Скорость выполнения инструкции чтения из памяти зависит от того, на каком уровне иерархии памяти находятся запрашиваемые данные. При планировании инструкций чтения из памяти обычно делается оптимистичное предположение о нахождении данных в кэш-памяти первого уровня, то есть задержка составляет 1–3 такта [16]. Вставка инструкции чтения из памяти в спланированный код приводит к дополнительным трудностям и потерям производительности. Инструкция чтения непосредственно перед использованием данных приводит к блокировке конвейера до получения результата инструкции. Вставка инструкции чтения выше с учетом задержки требует проведения дополнительного анализа. Например, перенос инструкции подкачки через широкую команду с использованием рассматриваемого регистра невозможен в принципе, так как значение регистра будет перезаписано, поэтому потери в такой ситуации неизбежны.

Однако даже когда зависимости позволяют учесть задержку для инструкции чтения из памяти, в широкой команде может просто не оказаться ресурсов для кодирования этой инструкции. Тогда в код придется вставлять новую широкую команду, в которой будет выполнена всего одна инструкция вместо потенциальных нескольких десятков инструкций. Если при этом задержка не будет выдержена, то количество пустых широких команд будет равно величине задержки, то есть в общем случае потери от такого планирования могут достигнуть

$$IL_{max} = DT_{max} * WIS_{max} - 1$$

инструкций, где IL – количество инструкций, которые могли быть спланированы, DT – вели-

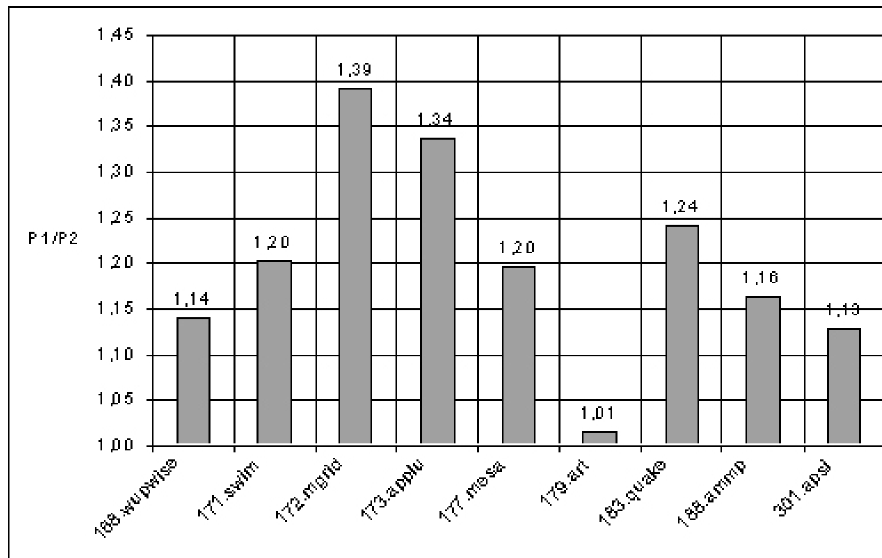

Рис. 1. Относительная производительность старой и новой схемы.

чина задержки, WIS – количество инструкций в широкой команде.

Аналогичные рассуждения можно применить и к планированию инструкций записи в память.

Одновременное планирование инструкций и распределение регистров позволяет выдержать необходимые задержки естественным образом благодаря созданию зависимостей между исходными инструкциями сетей и инструкциями откачки/подкачки. Эти зависимости учитываются алгоритмом планирования, благодаря чему инструкции откачки/подкачки участвуют в планировании на равных правах с остальными инструкциями и не требуют создания новых широких команд.

4.3. *Уменьшение алгоритмической сложности*

Классический алгоритм распределения регистров состоит в построении графа несовместимости и его, вообще говоря, итеративной раскраске R цветами, где R – количество доступных регистров. Распределение регистров путем раскраски графа несовместимости подробно описано в [2, 12]. Сложность алгоритма в варианте Бриггса экспериментально оценивается как $O(N{*}\log(N))$ [11], где N – общее количество сетей в программе.

При распределении регистров раскраской графа в нем должны быть представлены все сети: как глобальные, так и локальные. Альтернативный алгоритм – заполнение битовой регистровой матрицы – не является итеративным и не требует перестроения каких-либо структур при нехватке регистров. Назначение регистров локальным сетям интегрировано в алгоритм планирования инструкций. Все это позволяет значительно снизить суммарное время работы фаз распределения регистров и планирования инструкций. Сложность данного алгоритма по результатам экспериментальных запусков составляет $O(N)$, где N – общее количество сетей в программе.

5. ЭКСПЕРИМЕНТАЛЬНЫЕ РЕЗУЛЬТАТЫ

Измерения проводились на микропроцессоре “Эльбрус” [17, 18] с архитектурой VLIW. В микропроцессоре доступно 256 регистров, которые могут использоваться как для целочисленных, так и для плавающих вычислений, и 16 слогов для кодирования инструкций, причем в один слог можно закодировать одну или две инструкции в зависимости от их типа.

Основные преимущества алгоритма распределения регистров при планировании инструкций проявляются на задачах, требующих большего количества регистров, чем доступно в архитектуре, так как основной выигрыш в производительности дают уменьшение количества инструкций обращения в память и учет задержек при

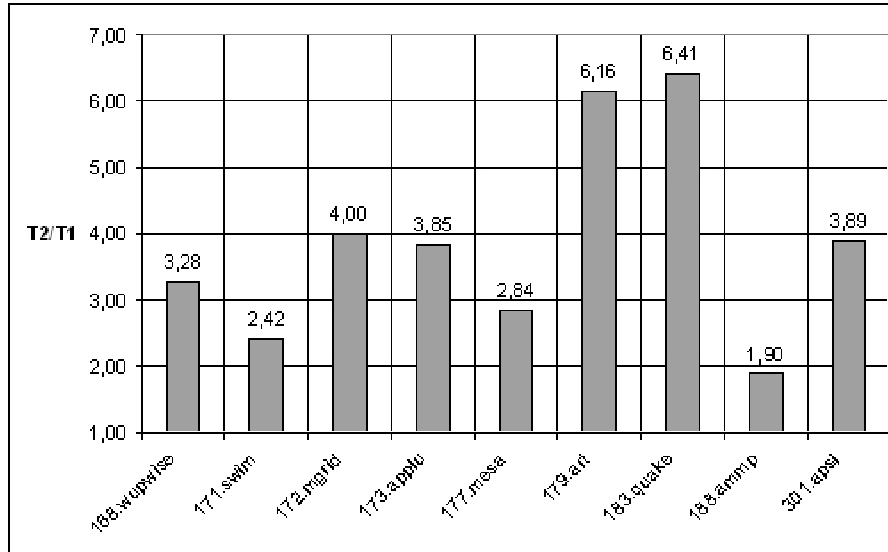

Рис. 2. Отношение суммарного времени работы фаз планирования и распределения регистров.

планировании таких инструкций. Для оценки эффективности алгоритма были выбраны задачи из пакетов SPEC CFP2000. Все результаты получены для регистрового окна процедуры размером 48 регистров.

Обозначим за $P1$ время исполнения задач, скомпилированных с использованием нового алгоритма, а за $P2$ – время исполнения тех же задач с распределением регистров на спланированном коде. Относительная производительность $P2/P1$ на задачах SPEC CFP2000 показана на рис. 1.

Обозначим за $T1$ время работы фазы распределения регистров при планировании инструкций, а за $T2$ – суммарное время работы фаз планирования и распределения. На рис. 2 показано относительное ускорение работы на задачах SPEC CFP2000 комбинированной схемы относительно схемы с распределением на спланированном коде, то есть $T2/T1$.

6. ЗАКЛЮЧЕНИЕ И ДАЛЬНЕЙШАЯ РАБОТА

Учет взаимодействия планирования инструкций и распределения регистров в среднем дает значительный прирост производительности, особенно для архитектур с широким командным словом. Однако на планирование и распределение сильное влияние оказывают и другие фазы, например, раскрутка циклов [15]. Определение таких фаз и разработка для них эффективных эвристик, учитывающих их взаимодействие с финальными фазами, является важной задачей для дальнейших исследований.

При отказе от графа несовместимости комбинированный алгоритм справляется со своей задачей в несколько раз быстрее. Однако практическая реализация глобального распределения регистров с помощью битовой матрицы требует дальнейшего, более глубокого исследования.

СПИСОК ЛИТЕРАТУРЫ

1. *Hennessy J.L., Patterson D.A.* Computer Architecture: A Quantitative Approach. San Francisco: Morgan Kauffman, 2003.
2. *Chaitin G.J.* Register Allocation and Spilling via Graph Coloring. Proceedings of the 1982 SIGPLAN Symposium on Compiler construction. June, 1982. P. 98–105.
3. *Berson D.A., Gupta R., Soffa M.L.* GURRR: A Global Unified Resource Requirements Representation // ACM SIGPLAN Notices. March, 1995. V. 30. Issue 3. P. 23–34.
4. *Norris C., Pollock L.L.* Register Allocation over the Program Dependence Graph. Proceedings of the 2004 ACM SIGPLAN Conference on Programming Language Design and Implementation. June, 2004. P. 266–277.
5. *Motwani R. et al.* Combining Register Allocation and Instruction Scheduling (CRISP). Technical Report TR 698. Courant Institute. July 1995.

6. *Pinter S.S.* Register Allocation with Instruction Scheduling: a New Approach // ACM SIGPLAN Notices. June, 1993. V. 28. Issue 6. P. 248–257.
7. *Koes D.R., Goldstein S.C.* A Global Progressive Register Allocator. Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation. June, 2006. P. 204–215.
8. *Xu W., Tessier R.* Tetris: a New Register Pressure Control Technique for VLIW Processors. Proceedings of the 2007 ACM SIGPLAN-SIGBED Conference. July, 2007. P. 113–122.
9. *Pereira F.M.Q., Palsberg J.* Register Allocation by Puzzle Solving. Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation. June, 2008. P. 216–226.
10. *Боханко А.С.* Распределение регистров методом раскраски графа несовместимости для современных вычислительных систем. Дис. канд. тех. наук: 05.13.11. М.: ЗАО "МЦСТ", 2005.
11. *Muchnick S.S.* Advanced Compiler Design And Implementation. San Francisco: Morgan Kaufman, 1997.
12. *Briggs P. et al.* Coloring Heuristics for Register Allocation. Proceedings of the ACM SIGPLAN 1989 Conference on Programming Language Design and Implementation. July, 1989. P. 275–284.
13. *Bernstein D. et al.* Spill Code Minimization Techniques for Optimizing Compilers. // ACM SIGPLAN Notices. July, 1989. V. 24. Issue 7. P. 258–263.
14. *Allen R., Kennedy K.*. Optimizing Compilers for Modern Architectures. San Francisco: Morgan Kauffman, 2002.
15. *Корнеев В., Киселев А.* Современные микропроцессоры. Санкт-Петербург: БХВ-Петербург, 2003.
16. *Babayan B.* E2K Technology and Implementation. Proceedings of the Euro-Par 2000 – Parallel Processing: the 6th International Conference. V. 1900/2000. January, 2000. P. 18–21.
17. *Галазин А.Б., Ступаченко Е.В., Шлыков С.Л.* Программный метод предварительной подкачки кода в архитектурах со статическим планированием // Программирование. 2008. № 1. С. 67–74.
18. *Fisher J.A., O'Donnel J.J.* VLIW Machines: Multiprocessors We Can Actually Program. CompCon'84 Proceedings, IEEE. February, 1984. P. 299–305.