ООО "ИНТЕРСОЦИОИНФОРМ"
***** КОМПЬЮТЕРЫ В УЧЕБНОМ ПРОЦЕССЕ *** № 5, май 2005 *****
--------- СТРАННЫЙ СПЕЦИАЛЬНЫЙ ЖУРНАЛ ДЛЯ ПЫТЛИВЫХ УМОВ ЛЮБОГО ВОЗРАСТА---------

ИССЛЕДОВАНИЕ МЕТОДОВ ПРЕОБРАЗОВАНИЯ ПРОГРАММЫ В ПРЕДИКАТНУЮ ФОРМУ ДЛЯ АРХИТЕКТУР С ЯВНО ВЫРАЖЕННОЙ ПАРАЛЛЕЛЬНОСТЬЮ

А.Ю.Дроздов,
С.В.Новиков
Институт микропроцессорных вычислительных систем РАН, Москва
E-mail: sasha@mcst.ru; novikov@mcst.ru

ВВЕДЕНИЕ

В современных микропроцессорах, таких как Эльбрус-3М фирмы ЗАО “МЦСТ” [1] или *Itanium* фирмы *Intel* [2], используется возможность параллельного исполнения операций. Для обозначения такого рода архитектур используется термин – *EPIC (Explicitly Parallel Instruction Computing)* или архитектура с явно выраженной параллельностью на уровне команд. Вычислительные машины такого класса могут одновременно выполнять несколько инструкций. Большую роль в вычислительных системах, использующих параллельность на уровне операций, играют программные средства распараллеливания приложений. Задачу по определению возможностей одновременного исполнения различных ветвей программы и оптимального их планирования и распределения по машинным ресурсам решает оптимизирующий компилятор [3].

Для получения эффективного кода в *EPIC*-архитектурах существует поддержка предикатных вычислений. Предикатными вычислениями называется условное исполнение команд, зависящих от значения условного операнда данной команды, называемого предикатом. Когда значение предиката истина (*TRUE*) команда исполняется нормально; а когда значение предиката ложь (*FALSE*), команда игнорируется. Поддержка предикатных вычислений в *EPIC*-архитектурах позволяет выполнять операции раньше перехода, идущего на ее исходный участок, если выполнение операции осуществлять под условием (предикатом) этого перехода. Перенося операцию выше нескольких переходов, предикат операции будет вычислен на основании условий этих переходов.

Таким образом, будем называть предикатом операции результат условного выражения, постановка под который позволяет переносить операцию выше заданных переходов.

1. ПРЕОБРАЗОВАНИЕ ПРОГРАММЫ В ПРЕДИКАТНУЮ ФОРМУ

Используя поддержку предикатных вычислений можно преобразовать ациклический регион программы, состоящий из нескольких линейных участков, в один линейный участок. При этом действии будут удалены все внутренние переходы ациклического региона. Процесс перевода безусловного кода в предикатный традиционно называется *if-conversion* [5, 6**, 9**].

По сути, *if-conversion* преобразует зависимости по управлению с операциями переходов в зависимости по данным с предикатами. Исходный регион для преобразования *if-conversion* обладает следующими свойствами:

1) регион является ациклическим;

2) имеется единственный вход;

3) управление внутри региона задано только условными предложениями;

4) из условных предложений может быть произвольное количество выходов за пределы региона.

Такой регион принято называть *гиперблоком* [4]. Получающиеся в результате преобразования *if-conversion* участки с предикатным кодом будем называть *расширенными скалярными участками – РСУ (Extended Scalar Block – ESB)*. На рис. 1 приведены примеры гиперблока и расширенного скалярного участка. На рис. 1*а* в гиперблок были включены узлы 1, 2, 3, 5, 6, 7, 8. Преобразование *if-conversion* привело к созданию расширенного скалярного участка 1 (рис. 1*б*).

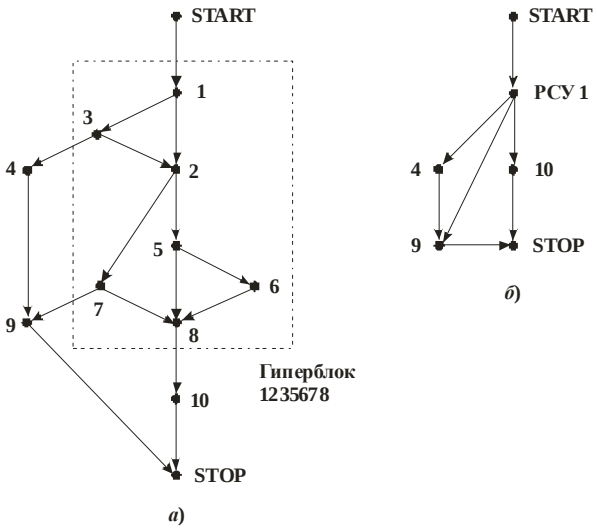

Рис. 1. Пример гиперблока и РСУ (*а* – Исходный управляющий граф;
б – Управляющий граф после *if-conversion*).

Основная трудность применения преобразования *if-conversion* состоит в определении участков, для которых перевод в предикатную форму приводит к ускорению их выполнения. Если гиперблок набран неверно, то после преобразования *if-conversion* время его исполнения может увеличиться. В данной работе предложен метод, позволяющий определять регионы, для которых перевод в предикатную форму эффективен. Главной особенность предлагаемого подхода можно назвать возможность учета эффекта от оптимизирующих преобразований на предикатном коде в момент набора регионов. Основным критерием при оценке эффективности набора регионов является результат планирования операций региона до преобразования и после него. Существенное улучшение качества набора регионов достигается с использованием профильной информации исполнения программы.

2. КРИТЕРИЙ ВКЛЮЧЕНИЯ УЗЛОВ В ГИПЕРУЗЕЛ

Оценить время исполнения расширенного линейного участка программы можно на основании результатов планирования и профильной информации. Планировщик для архитектур с явной параллельностью на уровне команд группирует операции, которые могут исполняться одновременно. При этом в планировании учитываются зависимости ме-

жду операциями и аппаратные ресурсы. Результатом планирования является расположение операций в широких командах или по-другому в группах параллельности. Все операции таких групп могут одновременно запускаться на исполнение. Если все группы перенумеровать, то можно оценить время запуска каждой из них. Таким образом, каждая операция после планирования получает время запуска на исполнение. Для получения оценки исполнения всего расширенного линейного участка нужно знать время планирования каждого перехода этого участка и его вероятность исполнения. Вероятность исполнения операций перехода является результатом профилирования программы. Профильной информацией называется знание о количестве исполнений переходов программы во время ее работы. По результатам профилирования определяется количество исполнений линейных участков, а также вероятности переходов относительно начал линейных участков. Пример спланированного участка с профильной информацией приведен на рис. 2*а*. Формула для определения оценки времени исполнения линейного участка выглядит так:

T(N) = T(br_1)*Prob(br_1) + ... + T(br_k)*Prob(br_k),

где N – расширенный скалярный участок; br_1, ..., br_k – операции перехода расширенного линейного участка; T – оценка времени исполнения; Prob – вероятность исполнения.

На основании тех же данных можно оценить время исполнения любого гиперблока программы. Для этого нужно определить вероятность каждого расширенного участка гиперблока относительно точки входа. Она определяется как отношение счетчиков исполнения участка и точки входа в регион.

Prob(N) = Counter(N)/Counter(Head),

где Counter(N) – счетчик участка; Counter(Head) – счетчик входа (головного участка региона).

На рис. 2*б* приведён пример региона со счетчиками. Формула для оценки времени исполнения ациклического региона программы с одной точкой входа выглядит так:

T(region)=T(N_1)*Prob(N_1)+...+T(N_k)*Prob(N_k).

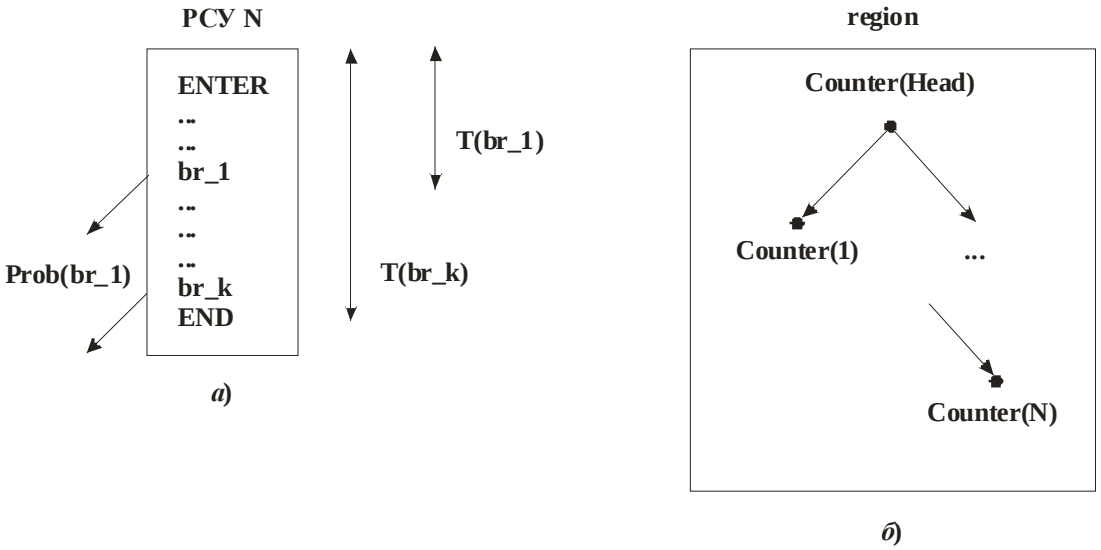

Рис. 2. Результаты профилирования и планирования
(*а* – Спланированный РСУ с профильной информацией;
б – Регион с профильной информацией).

Введя такую систему оценки исполнения регионов программы можно сформулировать критерий включения узлов в гиперузел. Если оценка исходного региона будет больше, чем оценка результирующего расширенного линейного участка, то данный регион может являться гиперблоком. В соответствии с рис. 1 критерий выглядит так:

T(гиперблок 1235678)>T(РСУ 1).

3. ГРАФ ЗАВИСИМОСТЕЙ

Планирование расширенного скалярного участка выполняется с учетом зависимостей между операциями. Зависимость задает порядок исполнения для двух операций. Преемник по зависимости не может быть выполнен раньше, чем предшественник. Отображение всех зависимостей в данной работе было реализовано в виде графа. Граф зависимостей представляет собой ориентированный связный граф без циклов (рис. 3). Узлами этого графа являются операции. Дугами графа являются зависимости. В предлагаемом подходе граф зависимостей должен быть корректным на каждом шаге алгоритма, так как является основой для работы алгоритма планирования, который в свою очередь позволяет оценивать эффективность набора регионов.

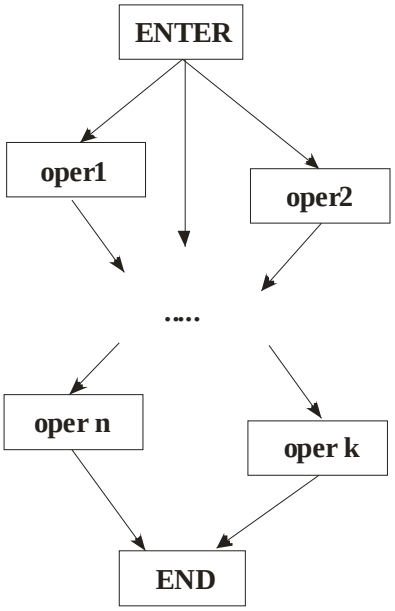

Рис. 3. Граф зависимостей.

4. ВОЗМОЖНОСТЬ ОТКАТА ВО ВРЕМЯ НАБОРА ГИПЕРБЛОКА

Необходимым условием в реализации предлагаемого метода набора гиперблоков является возможность восстановления исходного представления после применения преобразования *if-conversion* к ациклическому региону. Используя механизм оценки времени исполнения произвольных ациклических регионов, **описанный в гл. 2, можно** принимать решение об эффективности данного действия. Возможность отката позволит не делать неэффективных шагов. Это является основной идеей описываемого метода. Каким бы не был алгоритм набора гиперблоков, используя схему с откатами всегда можно избежать возможной деградации при применении преобразования *if-conversion*. Реализуется данный подход путем создания копии промежуточного представления со всеми необходимыми структурами данных и поддержание её в согласованном состоянии с исход-

ным представлением на каждом шаге алгоритма. При принятии положительного решения на некотором этапе копия должна быть согласована с оригиналом. При принятии отрицательного решения оригинал восстанавливается по копии. Основная задача в реализации механизма сохранения-восстановления промежуточного представления состоит в минимизации накладных расходов на это действие. Для этой цели в данной работе было обеспечено свойство локальности коррекций в момент принятия решений. Сохранение или восстановление всегда применяется только к тем участкам, которые были подвержены изменениям. Например, если набор узлов 1, 2, 3 в гиперблок 123 оказался неэффективен, то узел 123 удаляется, и восстанавливаются узлы 1, 2, 3. При этом никакой работы с узлами 5, 6, 7 не проводится.

5. АЛГОРИТМ НАБОРА ГИПЕРБЛОКОВ

Алгоритм набора гиперблоков состоит из последовательности проверок эффективности применения преобразования *if-conversion* к различным вариантам гиперблоков. Изначально вся программа разбивается на гиперблоки максимально возможных размеров. Границы выделяемого региона определяются – 1) границами циклов; 2) границами *switch*-конструкций; 3) границами других максимальных гиперблоков, которые получаются алгоритмом набора максимальных гиперблоков с использованием профильной информации и/или эвристик; 4) количественными характеристиками, задающими максимальное количество базовых участков и максимальное количество операций. В рамках максимальных гиперблоков возможно применение различных стратегий набора. В данной работе проведен сравнительный анализ трех стратегий и определен наиболее оптимальный подход к набору гиперблоков с точки зрения качества получаемых гиперблоков и времени работы алгоритма.

В каждой из трех стратегий формируются промежуточные варианты гиперблоков, которые общей техникой проверяются на качество набора. Алгоритм оценки качества набора гиперблока включает в себя применение преобразования *if-conversion* к исходному гиперузлу, коррекцию структур данных, используемых алгоритмом, применение локальных оптимизаций к полученному расширенному скалярному участку и оценку времени его исполнения. К структурам данных, требующих коррекции при работе алгоритма, относятся операции промежуточного представления, узлы и дуги управляющего графа, граф зависимостей. Особенностью предлагаемого подхода является возможность интеграции целого спектра локальных оптимизирующих преобразований в алгоритм оценки эффективности набора. В алгоритме были подключены локальные удаления избыточных вычислений, локальный сбор общих подвыражений, в том числе контекстных операций записи и чтения, набор оптимизаций критических путей [7, 8], оптимизации по упрощению предикатных выражений. При этом в алгоритме есть возможность наращивать оптимизирующий блок, подключая новые преобразования, чей эффект было бы полезно учесть при наборе гиперблоков.

Основным свойством оценки качества набора гиперблоков является то, что ни один шаг алгоритмов, основанных на этой оценке, не ухудшает временных характеристик получаемых участков.

Алгоритм 1. *Оценка качества набора гиперблока:*

На входе подается гиперузел.

1. Оценка времени исполнения исходного гиперузла.
2. Преобразование *if-conversion* для гиперузла.
3. Достраивание графа зависимостей на РСУ.

4. Применение локальных оптимизаций на РСУ.
5. Коррекция графа зависимостей на РСУ, который мог стать некорректным в результате применения локальных оптимизаций.
6. Оценка времени исполнения РСУ.
7. В случае ухудшения времени исполнения в результате применения преобразования *if-conversion* откат преобразования.
8. В случае улучшения времени исполнения осуществляется сохранение нового состояния в копии.

Выбор стратегии набора гиперблоков определяется двумя показателями: качеством набора и затраченным на это временем. В данной работе были опробованы три стратегии, к которым относится парная стратегия, стратегия набора по доминированию и смешанная стратегия. Из анализа была исключена стратегия полного перебора всех возможных наборов гиперблоков в силу своей большой временной сложности. Время работы алгоритма является существенным ограничением на возможность включения его в состав оптимизирующего компилятора.

Парная стратегия проверяет на каждом шаге эффективность гиперблоков, состоящих из двух расширенных скалярных участков. Процесс набора продолжается пока в исходном гиперузле не останется парных гиперблоков с положительной оценкой применения преобразования *if-conversion*. Предварительно пары узлов сортируются по частоте исполнения для того, что бы сначала набирать гиперблоки с наибольшим эффектом от преобразования. Стратегия набора гиперузлов по доминированию набирает регионы по дереву доминаторов [3]. Первым гиперузлом для проверки является исходный максимальный гиперузел. Последующие гиперузлы получаются в результате рекурсивного обхода вниз дерева доминаторов от головы исходного региона. Они представляют собой поддеревья дерева доминаторов с вершиной в текущем узле обхода, состоящие из узлов исходного максимального гиперузла. Стратегия по доминированию более быстрая, чем парная стратегия, так как количество шагов алгоритма меньше. Смешанная стратегия представляет собой комбинацию стратегии по доминированию и парной стратегии. Сначала к региону применяется стратегия по доминированию. Затем на результатах применения стратегии по доминированию запускается парная стратегия. Применение стратегии по доминированию обусловлено желанием ускорить работу алгоритма. Для получения более эффективного набора дополнительно применяется парная стратегия.

Алгоритм 2. *Парная стратегия набора гиперузлов.*

На вход подается максимальный гиперузел.

1. Сортировка пар узлов по частоте исполнения.
2. **Цикл** по парам узлов пока происходят положительные оценки преобразования if-conversion.
3. Оценка качества парного гиперблока алгоритмом 1.
4. **Конец цикла.**

Алгоритм 3. *Стратегия набора гиперузлов по доминированию.*

На вход подается максимальный гиперузел.

1. **Цикл** по узлам исходного гиперузла в порядке, определяемом деревом доминаторов.
2. Оценка качества алгоритмом 1 гиперузла, состоящего из поддерева дерева доминаторов, корневой вершиной которого является текущий узел обхода и все узлы поддерева принадлежат исходному гиперузлу.
3. **Конец цикла.**

Алгоритм 4. *Смешанная стратегия набора гиперузлов.*

На вход подается максимальный гиперузел.

1. Применение на исходном гиперузле стратегии набора гиперузлов по доминированию.
2. Применение на текущем гиперузле парной стратегии набора.

6. ЭКСПЕРИМЕНТАЛЬНЫЕ РЕЗУЛЬТАТЫ

Целью эксперимента был сравнительный анализ трех стратегий набора гиперблоков. Сравнение проводилось по качеству набора и по времени работы алгоритмов. Сравнение проводилось в контексте работы оптимизирующего компилятора проекта Эльбрус-**3М [1]** на реальных задачах пакета *Spec*95. Для оценки качества производились запуски оптимизирующего компилятора с тремя стратегиями набора гиперблоков, а также без применения преобразования *if-conversion*. Время исполнения полученных задач, измеренное в тактах архитектуры Эльбрус-3М, отображено в табл. 1. Для сравнения временных характеристик трех стратегий набора была собрана статистика по количеству шагов в алгоритмах. Число шагов было усреднено на одну процедуру задач. Результаты экспериментов по сравнительному анализу времени приведены в табл. 2.

Табл. 1.

Сводная таблица результатов исследований

Тест	Применение смешанной стратегии	Применение парной стратегии	Применение стратегии по доминированию	Исходное время работы тестов	Ускорение по времени работы (отношение %)
129.compress	18709082	18709074	19230792	23472179	25/25/22
099.go	1104560449	1095859257	1232696639	1642344260	48/50/33
124.m88ksim	248821928	249750384	266416000	452169977	81/81/70
134.perl	7724736	7768614	8006504	10775394	39/38/34
132.ijpeg	197878993	198523075	199775616	254531249	28/28/27
147.vortex	219155702	221784930	224753935	533074354	143/140/137
130.li	226893037	228965256	247870299	367636284	62/60/48

Табл. 2.

Таблица временных характеристик стратегий набора

Тест	Применение смешанной стратегии	Применение парной стратегии	Применение стратегии по доминированию
129.compress	11	94	10
099.go	50	210	21
124.m88ksim	10	68	6
134.perl	21	107	10
132.ijpeg	7	42	5
147.vortex	15	154	10
130.li	8	53	6

Экспериментальные данные по качеству набора гиперблоков показывают, что все три стратегии существенно ускоряют исполнение реальных задач (от 22% до 143%).

Тем не менее, парная стратегия и смешанная стратегия всегда дают более высокое ускорение, чем стратегия по доминированию. Если сравнивать временные затраты трех стратегий, то самой затратной является парная стратегия. При ее применении требуется значительно больше шагов алгоритма, чем при применении смешенной стратегии и стратегии по доминированию. Таким образом, на основании экспериментальных данных можно сделать вывод, что при относительно небольших временных затратах наилучшее качество набора осуществляется при смешанной стратегии.

ЗАКЛЮЧЕНИЕ

В данной работе предложен новый подход к решению задачи набора гиперблоков.

В рамках этого подхода проведен сравнительный анализ трех стратегий набора. **При анализе были рассмотрены временные и качественные характеристики стратегий.** По результатам экспериментов определена наиболее оптимальная стратегия. Предложенный подход позволил преодолеть основные трудности задачи набора гиперблоков. Во-первых, алгоритм набора всегда улучшает время исполнения приложений. При данном подходе исключено принятие решений, ведущих к деградации. Во-вторых, в рамках рассматриваемого алгоритма реализована возможность учета локальных оптимизирующих преобразований на каждом шаге набора. Интеграция оптимизаций в алгоритм набора гиперблоков способствует получению более качественного результата. В настоящий момент схема с откатами, а именно смешанная стратегия набора гиперблоков, успешно применяется в оптимизирующем компиляторе архитектур Эльбрус-3М и Itanium.

ЛИТЕРАТУРА

1. **Dieffendorf K.** The Russians Are Coming. Supercomputer Maker Elbrus Seeks to Join x86/IA-64 Melee // Microprocessor Report, V. 13, № 2. February 15, 1999. P. 1-7.

2. **Intel Corporation.** Intel® Itanium® 2 Processor Reference Manual for Software Development and Optimization. Apr. 2003.

3. **Muchnick, Steven S.** Advanced Compiler Design and Implementation. San Francisco: Morgan Kauffman, 1997. Chapter 7.2.

4. **Mahlke S.A., Lin D.C., Chen W.Y., Hank R.E., Bringmann R.A.** Effective Compiler Support for Predicated Execution Using the Hyperblock / Proceedings of the 25th International Symposium on Microarchitecture, December 1992. P. 45-54.

5. **August D.I., Wen-mei W. Hwu,** and **Mahlke S.A.** A Framework for Balancing Control Flow and Predication // Proceedings of the 30th annual IEEE/ACM International Symposium on Microarchitecture. December, 1997. P. 92-103.

6. **Park J.C.H.; Schlansker M**. On Predicated Execution. Software and System Laboratory HPL-91-58, May, 1991.

7. **Волконский В.Ю., Окунев С.К.** Предикатное представление как основа оптимизации программ для архитектур с явно выраженной параллельностью // Информационные технологии, № 4, 2003. С. 36-44.

8. **Волконский В.Ю., Окунев С.К.** Оптимизация критического пути на предикатном представлении программы // Информационные технологии, № 9, 2003. С. 2-13.

9. **Дроздов А. Ю., Новиков С. В., Шилов В. В. Эффективный алгоритм преобразования потока управления в поток данных. // Приложение к журналу "Информационные технологии", № 2, 2005. С. 24-31.**