

Работа над совместными проектами

Социальная сеть позволяет пригласить любого участника к работе над общим проектом. Рабочие области проектов представляют собой онлайн-офисы с возможностью общения между участниками проекта и обмена файлами внутри одного проекта. Любой участник социальной сети может принять участие в проекте, для чего необходимо пройти предварительную проверку администратором проекта. Таким образом, ученые из разных уголков страны могут работать вместе независимо от географии. Для проектов также могут быть выбраны любые настройки приватности, что позволяет обеспечить защиту конфиденциальной (служебной) информации от третьих лиц.

Все элементы системы доступны и при использовании мобильной версии портала. При посещении с мобильного устройства социальная сеть покажет специальную облегченную версию для небольших экранов и экономии трафика. При этом пользователю будет доступен весь функционал, включая просмотр документов внутри сети без использования внешних программ.

Перспективы развития проекта

Портал социальной сети РАН представляет собой социальную сеть профессиональной тематики и рассматривается прежде всего как полезный инструмент в повседневной жизни российских ученых и специалистов. Главная цель проекта – создание в Интернете единого информационного пространства для всех научных работников России, где люди смогут общаться друг с другом, объединяться в группы и работать над совместными проектами онлайн, находясь в разных частях страны. Все сотрудники региональных отделений и научных центров РАН являются потенциальными пользователями социальной сети, однако наибольшую популярность, как представляется, сеть завоевывает среди ученых молодого и среднего возраста, активно и уверенно использующих возможности Интернета [6].

Использование социальной сети для общения приведет к значительному снижению временных и стоимостных затрат на организацию исследовательских проектов. В результате значительно упростится процедура поиска специалистов в нужной области – достаточно определить параметры поиска с помощью удобного фильтра и написать личное сообщение.

В настоящее время проводится активное тестирование портала и начинается рекламная кампания проекта. В дальнейшем планируется объединить социальную сеть с крупными российскими электронными библиотеками и музеями, создав таким образом единый государственный научно-образовательный информационный портал.

Литература

1. Коноплицкий П. Будущее социальных сетей в России. URL:<http://habrahabr.ru/> (дата обращения: 15.09.2012).
2. Портал Российской академии наук: Об Академии. URL: <http://ras.ru/> (дата обращения: 30.09.2012).
3. Блинков И. Архитектура Facebook. URL: <http://www.insight-it.ru/masshtabiruemost/arkhitektura-facebook/> (дата обращения: 17.09.2012).
4. Model-View-Controller. URL: <http://ru.wikipedia.org/wiki/Model-View-Controller> (дата обращения: 15.09.2012).
5. Социальная сеть Российской академии наук. URL: <http://sn.ras.ru/> (дата обращения: 15.09.2012).
6. Google INC. Доля интернет-пользователей среди населения. URL:<http://www.google.ru/publicdata/> (дата обращения: 25.09.2012).

References

1. Konoplitsky P., *Budushchee sotsialnykh setey v Rossii* [The future of social networks in Russia], Available at: <http://habrahabr.ru/> (accessed 15 September 2012).
2. Rossiyskaya Akademiya Nauk, Available at: <http://ras.ru/> (accessed 30 September 2012).
3. Blinkov I., *Arkhitektura Facebook*, Available at: <http://www.insight-it.ru/masshtabiruemost/arkhitektura-facebook/> (accessed 17 September 2012).
4. *Model-View-Controller*, Available at: <http://ru.wikipedia.org/wiki/Model-View-Controller> (accessed 15 September 2012).
5. *Sotsialnaya set Rossiyskoy akademii nauk*, Available at: <http://sn.ras.ru/> (accessed 15 September 2012).
6. *Google INC. Dolya internet-polzovateley sredi naseleniya*, Available at: <http://www.google.ru/publicdata/> (accessed 25 September 2012).

УДК 004.4

ОПТИМИЗАЦИЯ ПЕРЕХОДОВ В ДВОИЧНОМ ТРАНСЛЯТОРЕ ДЛЯ АРХИТЕКТУРЫ «ЭЛЬБРУС»

A.A. Рыбаков, с.н.с.

(Компания «МЦСТ», ул. Нижняя Красносельская, 35, строение 50, Москва, 105066, Россия,
rybakov.aax@gmail.com)

В компании «МЦСТ» разрабатываются микропроцессоры архитектуры «Эльбрус». Для вычислительного комплекса «Эльбрус» создана система двоичной трансляции LIntel, позволяющая исполнять приложения Intel x86 на

микропроцессорах «Эльбрус». Важной составляющей LIntel является многоуровневый оптимизирующий двоичный транслятор. Команды подготовки переходов в архитектуре «Эльбрус» позволяют распараллеливать исполнение программы по разным ветвям и выполнять переходы за один такт без потери тактов. Применение оптимизации переходов в двоичном трансляторе дает возможность переносить команды подготовки переходов между линейными участками программы, что приводит к повышению производительности системы.

Ключевые слова: оптимизирующий компилятор, двоичная трансляция, Intel x86, «Эльбрус», глобальное планирование, промежуточное представление, подготовка перехода.

BRANCHES OPTIMIZATION IN BINARY TRANSLATOR FOR «ELBRUS» ARCHITECTURE

Rybakov A.A., Senior Researcher

(MCST, 35/50, Nizhnyaya Krasnoselskaya St., Moscow, 105066, Russia, rybakov.aax@gmail.com)

Abstract. MCST company develops «Elbrus» architecture microprocessors. For «Elbrus» architecture binary translation system LIntel is developed. Lintel allows execute Intel x86 applications on «Elbrus» microprocessors. Multilevel optimizing binary translator is important component of Lintel. Branch preparing instructions in «Elbrus» architecture allow parallelize program execution and process branches instructions immediately, without loss of processor time. Applying branches optimization in binary translator allows transfer branch preparing instructions between linear sections of program. This optimization leads to system performance improvement.

Keywords: optimizing compiler, binary translation, Intel x86, «Elbrus», global scheduling, intermediate representation, branch preparing.

В течение всего времени развития компьютерной техники проблема скорости выполнения программ не теряет своей актуальности. Для повышения скорости работы приложений создано множество различных микропроцессорных архитектур в совокупности с оптимизирующими компиляторами, позволяющими использовать особенности данных архитектур. При появлении новых микропроцессорных архитектур возникает проблема переноса на них ранее созданного программного обеспечения. При этом исходный код того или иного приложения зачастую недоступен и перекомпилировать его для новой архитектуры не представляется возможным. Технология динамической двоичной трансляции позволяет перенести программное обеспечение путем перевода двоичного кода из набора инструкций исходной архитектуры в набор инструкций целевой архитектуры.

В компании «МЦСТ» разрабатываются микропроцессоры архитектуры «Эльбрус» [1]. Данная архитектура имеет следующие свойства [2]:

- широкое командное слово, позволяющее одновременно исполнять несколько инструкций;
- большой размер регистрового файла;
- наличие спекулятивного (упреждающего) и предикатного (условного) режимов исполнения инструкций;
- специальные операции подготовки переходов для распараллеливания передачи управления по разным веткам;
- аппаратная поддержка конвейеризации циклов;
- механизм асинхронной предварительной подкачки данных.

Технология динамической двоичной трансляции обеспечивает полную совместимость архитектуры «Эльбрус» с архитектурой Intel x86. Для комплекса «Эльбрус» разработана аппаратно поддерживаемая система двоичной трансляции LIntel, которая эмулирует поведение машины x86 путем

декодирования инструкций x86 и перевода их в коды архитектуры «Эльбрус» [3].

Основными составляющими LIntel являются интерпретатор, многоуровневый двоичный транслятор и система поддержки, обеспечивающая функционирование и целостность всей системы.

Интерпретатор предназначен для пошагового исполнения инструкций x86 с точным их моделированием и обработкой возможных прерываний. Интерпретатор используется, когда требуется моделировать поведение процессора при возникновении исключения или при первом исполнении кода. Если код x86 начинает исполняться часто, управление от интерпретатора передается многоуровневому двоичному транслятору. Задачей транслятора является создание кода целевой платформы, который может быть сохранен и впоследствии многократно выполнен.

Двоичный транслятор LIntel состоит из трех уровней. Первый уровень – шаблонный транслятор. Далее следует быстрый оптимизирующий компилятор (компилятор уровня O0), который применяет к созданному коду некоторый ограниченный набор оптимизаций. Качество кода возрастает, возрастает и время компиляции. И, наконец, оптимизирующий компилятор уровня O1, выполняющий полный набор оптимизаций, создает наиболее эффективный код, но еще больше времени затрачивает на трансляцию. Из соображений минимизации времени работы системы в целом нужно придерживаться правила: чем чаще исполняется код, тем более высокого уровня транслятор целесообразно использовать. Во время работы системы переключение между уровнями трансляции происходит динамически.

В таблице приведены сравнительные данные по времени компиляции исходного кода x86 различными уровнями транслятора, а также сравнение результирующего кода с кодом, полученным с помощью компилятора O1. В колонке «Время трансляции» приведено количество тактов ком-

плекса, затрачиваемое на трансляцию, в пересчете на одну исходную инструкцию x86. Для интерпретатора это означает количество тактов, затрачиваемое на эмуляцию одной инструкции. В колонке «Качество кода» приведено отношение средней скорости работы результирующего кода, полученного с помощью определенного уровня транслятора, к средней скорости работы результирующего кода, полученного с помощью компилятора O1. Для интерпретатора эта характеристика лишена смысла, так как для него отсутствует понятие результирующего кода. В данной статье речь идет о быстром оптимизирующем компиляторе [4].

Характеристики уровней двоичной трансляции

| Составляющие транслятора | Время трансляции | Качество кода |
|--------------------------|------------------|---------------|
| Интерпретатор | 100 | - |
| Шаблонный транслятор | 1 200 | 1 / 4 |
| Компилятор O0 | 15 000 | 2 / 3 |
| Компилятор O1 | 500 000 | 1 |

Единицей компиляции быстрого компилятора является регион, который представляет собой объединение некоторого числа линейных участков (последовательностей команд с одной точкой входа), связанных между собой переходами. Если некоторый линейный участок оканчивается не командой перехода, будем говорить, что он оканчивается провалом. В этом случае после исполнения последней команды линейного участка управление перейдет на следующую ячейку памяти. Для своей работы быстрый компилятор использует промежуточное представление, основой которого является граф потока управления. Узлами данного графа являются линейные участки, а ребрами – переходы между ними.

Процесс компиляции региона можно условно разделить на три этапа. На первом этапе для каждого линейного участка осуществляется генерация семантики команд x86. Далее, опираясь на сгенерированное промежуточное представление, последовательно применяется ряд базовых оптимизаций, позволяющих существенно повысить производительность результирующего кода при небольшом увеличении времени компиляции. К наиболее важным оптимизациям относятся слияние суперблоков, перенос операций между узлами, устранение избыточных обращений в память, разрыв зависимостей по доступу в память и другие. После проведения всех оптимизаций для каждого узла промежуточного представления осуществляются распределение регистров, планирование широких команд и генерация кода архитектуры «Эльбрус». В данной статье рассматривается оптимизация переноса подготовок переходов между линейными участками (узлами графа потока управления).

При проведении оптимизаций и планировании кода в рамках узла промежуточного представле-

ния из-за задержек между операциями возникает простояивание вычислительных ресурсов, обусловленное незанятостью исполнительных устройств в некоторых тактах. Оптимизация переноса критических операций между узлами позволяет удалить операции из одного узла и спланировать их в потенциально свободные места в других узлах. По своему назначению данная оптимизация близка к шагу глобального планирования [5]. Идея состоит в переносе критических операций, стоящих в начале узла, вверх по всем входящим дугам в предшественники данного узла. При этом перенесенная операция располагается перед переходом по соответствующей дуге. Цель такого преобразования – попытка загрузить с помощью перемещаемых операций потенциально свободные вычислительные ресурсы.

Рассмотрим пример перемещения операции между узлами. Допустим, принято решение о переносе операции из узла A в узел B. Существуют три варианта связи между двумя данными узлами.

1. Узел B доминирует над A, узел A постдоминирует над B. Данный случай является самым простым. Операция может быть перенесена из узла A в узел B и помещена перед переходом на узел A.

2. Узел B не доминирует над A. Это значит, что существует путь в узел A в обход узла B (пусть такой путь проходит через узел C). В данном случае операция должна быть перенесена не только в узел B – в узле C должна быть расположена ее копия.

3. Узел A не постдоминирует над B. Это значит, что существует путь из узла B, не проходящий через A (пусть такой путь проходит через узел D). В данном случае необходимо контролировать, чтобы при потоке исполнения по пути *Node B → Node D* выполнение перенесенной операции не оказывало влияния на выполнение в узле D и далее.

Чтобы точно определить, является ли операция критической, то есть задерживает планирование идущих за ней операций, нужно провести предварительное планирование, что значительно увеличит время компиляции региона. Поэтому для принятия решения применяются приближенные эвристические оценки. Нужно учитывать и возможный негативный эффект от излишне агрессивного применения оптимизации. Так, если операция должна быть перенесена хотя бы по одной дуге в узел, количество исполнений которого существенно больше количества исполнений исходного узла, то от применения оптимизации к данной операции следует отказаться, так как это приведет к перемешиванию часто исполняемого кода с кодом, исполняемым гораздо реже.

В архитектуре «Эльбрус» для переходов (BRANCH) используются специальные регистры, которые содержат адрес перехода и некоторую

дополнительную информацию (CTPR). Эти регистры, которых всего три, также называют станками переходов. Для инициализации станков используются специальные команды подготовки переходов (CTP), позволяющие распараллелить передачу управления по разным веткам. При использовании операций подготовки переходов возможно выполнение предварительной подкачки кода, что приводит к уменьшению задержек подкачки кода, связанных с промахами в кэш инструкций. При планировании команд между переходом и его подготовкой должна быть выдержанна определенная задержка. Если эта задержка выдержана, переход выполняется за один такт без потери тактов. Однако если подготовка перехода стоит близко к переходу, это может привести к потере тактов. Поэтому операции подготовки переходов, использующиеся в архитектуре «Эльбрус», являются подходящими кандидатами для применения оптимизации переноса операций между узлами.

Рассмотрим алгоритм, позволяющий перемещать подготовки переходов. Во время проведения оптимизации доступен профиль выполнения оптимизируемого региона, в частности, для каждого узла известно количество его исполнений (счетчик узла).

Из каждого узла будем пытаться вынести подготовку только для одного первого перехода. Из-за небольшого количества станков выносить другие переходы (кроме первого) нецелесообразно. Узлы промежуточного представления будем обрабатывать последовательно в порядке убывания счетчика, начиная с узла с максимальным счетчиком. Это поможет уменьшить размер кода прежде всего для самых частых узлов. Последовательность действий при обработке каждого узла следующая.

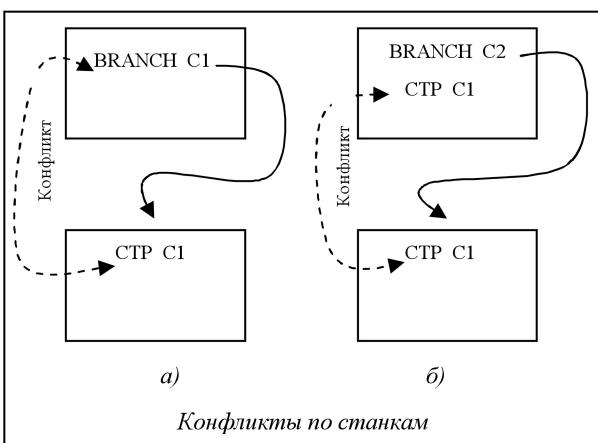
- Находим первую подготовку перехода в узле, если таковая есть.
- При переносе операции подготовки перехода вверх по входным дугам в узлы-предшественники возможно возникновение конфликтов по станку. При возникновении конфликта станок переносимой операции должен быть изменен. Для определения доступных номеров станков вычисляется соответственно маска допустимых для использования в подготовке станков. Если эта маска нулевая, значит, перенос выполнить не удастся и нужно переходить к обработке следующего узла.
- Далее следует проверить, нужно ли проводить переименование станка в операции подготовки перехода (и в соответствующем переходе). Если в маске допустимых станков присутствует номер станка подготовки, переименование производить не нужно. В противном случае следует выбрать любой номер станка из маски допустимых номеров и выполнить переименование станка в подготовке и соответствующем ей переходе.

- Перемещаем копию операции подготовки перехода вверх по всем входным дугам. Сама операция подготовки перехода удаляется из узла.

Рассмотрим некоторые особенности описанного алгоритма. Так как при переносе подготовки перехода возможно возникновение конфликтов по станку, необходимо обнаружить и разрешить данные конфликты. Конфликты по станку бывают двух типов.

Пусть переносимая операция подготовки перехода использует некоторый станок C1. Если данная подготовка должна переноситься по дуге, переход по которой также выполняется по станку C1, имеем конфликт первого типа (на рисунке (а)).

Конфликт второго типа связан с тем, что в одном такте могут быть спланированы одна операция перехода и одна операция подготовки перехода. Пусть переносимая операция подготовки перехода использует станок C1. Пусть она переносится из узла Node A в узел Node B по дуге, переход по которой осуществляется по другому станку – C2. Если в узле Node B ниже перехода по дуге Node B → Node A присутствует другая подготовка, использующая станок C1, то она может быть спланирована в одном такте с данным переходом. В этом случае имеем конфликт второго типа (на рисунке (б)).



Все возможные конфликты по станкам по всем входящим в узел дугам должны быть учтены при переносе подготовки перехода в маске доступных станков, иначе может возникнуть ошибка исполнения.

Заметим также, что оптимизация использует некоторые эвристики, которые позволяют повысить производительность результирующего кода. Например, перенос подготовок для маловероятных переходов зачастую оказывается бесполезным, так что в некоторых случаях от него можно отказаться. С другой стороны, переименование станков может привести к возникновению двух подряд идущих переходов по одному и тому же станку. Это гарантированно ухудшает результирующий код, так что переименования станков в этом случае желательно избегать.

Для оценки сложности алгоритма обозначим граф потока управления (control flow graph, CFG) через G . Пусть $v = v_G$ – количество его узлов, $\varepsilon = \varepsilon_G$ – количество ребер. Так как оптимизация применяется к узлам в порядке уменьшения их счетчиков, перед ее применением необходимо произвести сортировку узлов графа. Сложность данного действия равна $O(v \log v)$. Для нахождения первой подготовки в каждом узле CFG необходимо обойти операции каждого узла, начиная с первой операции и заканчивая первой подготовкой. Количество действий, необходимое для этого, пропорционально общему количеству операций в графе, или $O(lv)$, где l – среднее число операций в узле. Для каждой подготовки для каждой входной дуги необходимо выполнить анализ конфликтов по станкам.

Так как всего подготовок, которые могут быть перенесены, $O(v)$, а среднее количество входных дуг в узел равно $O\left(\frac{\varepsilon}{v}\right)$, нужно проверить на конфликты $O(\varepsilon)$ входных дуг. Для проверки на конфликт по станкам необходимо обойти все операции от соответствующего перехода и до конца узла (количество действий пропорционально l). С учетом этого общее количество действий, необходимое для проверки конфликтов, равно $O(l\varepsilon)$. Количество действий, требуемое для самого переноса операций подготовок, равно $O(\varepsilon)$. Суммируя все необходимые для проведения оптимизации действия, получим итоговую оценку сложности алгоритма: $T(v, \varepsilon, l) = O(v \log v + l(v + \varepsilon))$.

Оценка эффективности описанного алгоритма производилась путем запуска исполняемых файлов x86 задач из пакета SPEC CINT2000, транслированных с помощью быстрого компилятора, на симуляторе микропроцессора «Эльбрус». Для оценки общего эффекта от использования переноса подготовок между узлами были произведены запуски задач, откомпилированных базовым компилятором в следующих режимах:

- без использования переноса подготовок между узлами (результаты запусков в данном режиме приняты за этalon);
- с использованием переноса подготовок между узлами.

Прирост производительности на разных задачах составил от 0,2 % до 2,8 % в зависимости от задачи. Особенно заметный прирост производительности наблюдается на задаче 176.gcc, так как эта задача характеризуется сильно разветвленным управлением и небольшой длиной линейных участков.

Оптимизация переноса подготовок переходов между узлами промежуточного представления применяется в рамках общей оптимизации переноса операций. Кроме подготовок переходов, могут быть перенесены также операции чтения из

памяти. Поэтому подсчитать точное время, затрачиваемое компилятором на перенос именно подготовок переходов, затруднительно. Но примерно это время можно оценить в 0,5 % от времени работы быстрого компилятора.

Технология двоичной трансляции в современном мире является востребованной, так как позволяет выполнять двоичный код одной архитектуры на процессорах других архитектур. При этом при создании кода целевой платформы использование оптимизирующей компиляции способно существенно увеличить работу приложения. Важным моментом является создание адаптивных многоуровневых оптимизирующих двоичных трансляторов, позволяющих регулировать линейку и агрессивность применяемых оптимизаций в зависимости от частоты исполнения оптимизируемого кода. Типичный контекст для работы быстрого компилятора системы двоичной трансляции для архитектуры «Эльбрус» – код с не слишком большой частотой исполнения и с очень разветвленным управлением.

Таким образом, важной частью быстрого компилятора является оптимизация переходов. Алгоритм переноса подготовок переходов между узлами промежуточного представления направлен на оптимизацию переходов и позволяет добиться заметного улучшения производительности результирующего кода.

Литература

1. Babayan B. Main principles of E2K architecture. Free Software Magazine. Vol. 1, no. 2, February 2002.
2. Волконский В.Ю. Оптимизирующие компиляторы для архитектуры с явным параллелизмом команд и аппаратной поддержкой двоичной совместимости // Информационные технологии и вычислительные системы. 2004. № 3.
3. Воронов Н.В., Гимпельсон В.Д., Маслов М.В., Рыбаков А.А., Сюсюкалов Н.С. Система динамической двоичной трансляции x86 «Эльбрус» // Вопросы радиоэлектроники: сер. ЭВТ. 2012. Вып. 3.
4. Рыбаков А.А., Маслов М.В. Быстрый региональный компилятор системы двоичной трансляции для архитектуры «Эльбрус» // Современные информ. технологии и ИТ-образование: сб. избран. тр. V Междунар. науч.-практич. конф. М.: МГУ, 2010. С. 436–443.
5. Muchnick S. Advanced Compiler Design and Implementation. Morgan Kaufmann Publishers, 1997.

References

1. Babayan B., Free Software Magazine, Vol. 1, no. 2, February 2002.
2. Volkonsky V.Yu., *Informatsionnye tekhnologii i vychislitelnye sistemy*, 2004, № 3.
3. Voronov N.V., Gimpelson V.D., Maslov M.V., Rybakov A.A., Syusyukalov N.S., *Voprosy radioelektroniki*, 2012, Vol. 3.
4. Rybakov A.A., Maslov M.V., Sbornik trudov V Mezhdunar. Konf. «Sovremennye inform. tekhnologii i IT-obrazovanie» [Proc. V Intern. Conf. «Modern Information Technologies and IT-education»], Moscow, MSU, 2010, pp. 436–443.
5. Muchnick S., *Advanced Compiler Design and Implementation*, Morgan Kaufmann Publ., 1997.