ПРОГРАММИРОВАНИЕ, 2012, № 3, с. 45–56

ЯЗЫКИ И СИСТЕМЫ ПРОГРАММИРОВАНИЯ

УДК 681.3.06

ФОНОВАЯ ОПТИМИЗАЦИЯ В СИСТЕМЕ ДИНАМИЧЕСКОЙ ДВОИЧНОЙ ТРАНСЛЯЦИИ

© 2012 г. Р.А. Соколов*, А.В. Ермолович**

**ЗАО МЦСТ, 105066 Москва, ул. Красносельская Нижняя, дом 35, корп. 50*

***ЗАО Интел А/О, 121614 Москва, ул. Крылатская, дом 17*

E-mail: roman.a.sokolov@gmail.com, karbo@pvk13.org

Поступила в редакцию 30.11.2011

Технологии программной двоичной трансляции и динамической оптимизации активно используются для обеспечения совместимости широко распространенных традиционных и вновь разрабатываемых перспективных архитектур микропроцессоров на уровне исполняемых программных кодов. Динамическая оптимизация является одним из основных способов достижения высокой производительности в системах динамической двоичной трансляции, однако может быть и источником значительных накладных расходов, так как оптимизация выполняется непосредственно во время работы транслируемых кодов. Одним из способов снижения издержек, обусловленных динамической оптимизацией, является выполнение оптимизирующей фазы одновременно с исполнением исходного двоичного кода за счет использования не занятых вычислительных ресурсов системы. В данной работе этот принцип применяется для системы динамической двоичной трансляции уровня виртуальной машины. Для этого в системе была реализована возможность многопоточного исполнения, которая позволяет выполнять фазу динамической оптимизации независимо от основного потока исполнения двоично-транслируемых кодов, в отдельном потоке. При этом поток оптимизации может разделять процессорное время с потоком исполнения в однопроцессорной (или одноядерной) системе, либо может быть полностью перенесен на отдельное, не занятое другими задачами микропроцессорное ядро. В первом случае представленная схема позволяет избавиться от латентности, привносимой в систему "тяжелой" фазой динамической оптимизации. Во втором случае перекрывание потоков исполнения и динамической оптимизации также позволяет исключить время, затрачиваемое на получение оптимизированных кодов, из общего времени работы исходных кодов.

1. ВВЕДЕНИЕ

Технологии программной *двоичной трансляции* и *динамической оптимизации* широко применяются в современных программных и аппаратных вычислительных системах [1]. В частности, основанные на этих технологиях *системы динамической двоичной трансляции* (СДДТ) активно используются для обеспечения совместимости распространенных традиционных (*исходных*) и вновь разрабатываемых перспективных (*целевых*) архитектур микропроцессоров на уровне исполняемых программных кодов.

СДДТ выполняют трансляцию кодов исходной архитектуры в коды целевой архитектуры небольшими участками по ходу выполнения задачи, при этом процессорное время попеременно отводится под собственно трансляцию и исполнение полученных на выходе двоично-транслированных кодов. Одно из главных требований к любой СДДТ заключается в том, что производительность исполнения двоичных кодов посредством данной технологии на вычислительных системах целевой архитектуры должна быть сравнима или выше производительности их исполнения на системах исходной архитектуры.

Использование оптимизирующего транслятора является одним из основных способов достижения высокой производительности СДДТ, так как позволяет получать максимально эффективный код, полностью использующий возможности новой архитектуры, а также использовать при проведении оптимизационных преобразований информацию (в частности, профильную) о реальном поведении программы, не доступную статическому компилятору.

Однако так как динамическая оптимизация выполняется непосредственно во время исполнения исходных программных кодов, она может быть и источником значительных накладных расходов. Во-первых, совокупное затраченное на оптимизирующую трансляцию время может составлять заметную долю общего времени работы программы и не обязательно будет компенсировано выигрышем, полученным от применения оптимизационных преобразований.

Во-вторых, работа оптимизирующего транслятора может отрицательно сказываться на *латентности* транслируемого интерактивного приложения или операционной системы в целом. Под латентностью здесь понимается время отклика системы на какие-либо внешние события (в частности, аппаратные прерывания), вызванные подключенными к системе устройствами ввода/вывода. Латентность вычислительной системы как ее способность оперативно реагировать на внешние события является не менее важным свойством, с точки зрения пользователя, подключенного к компьютеру оборудования или другого компьютера, подключенного по сети, чем общая производительность. Этот параметр особенно важен для СДДТ уровня виртуальной машины (ВМ). Такие системы реализуют семантику исходной архитектуры в полном объеме и предназначены для трансляции всей иерархии программного обеспечения вычислительного комплекса – BIOS, операционной системы, приложений, и, как следствие, контролируют всю работу вычислительного комплекса, фактически являясь частью микропроцессора.

Один из способов снижения издержек, связанных с динамической оптимизацией, заключается в том, чтобы выполнять ее одновременно с исполнением исходных кодов, используя для этого не занятые вычислительные ресурсы системы. Он находит применение в системах динамической трансляции и оптимизации различного назначения [2, 8]. В рамках данной работы мы будем называть этот подход *фоновой оптимизацией* (в отличие от последовательной оптимизации, когда исполнение прерывается до конца оптимизации).

Непосредственными результатами данной работы являются:

- реализация многопоточной инфраструктуры в системе динамической двоичной трансляции уровня виртуальной машины;
- реализация фоновой оптимизации в однопроцессорной системе посредством разделения процессорного времени между потоками исполнения двоично-транслируемых кодов и оптимизирующей трансляции;
- реализация фоновой оптимизации в двухпроцессорной системе посредством выноса оптимизирующей трансляции на свободное микропроцессорное ядро.

Отметим, что два представленных подхода к реализации фоновой оптимизации преследуют различные цели. Первая схема позволяет исключить задержки, создаваемые работой оптимизирующего транслятора, на фоне ограниченных вычислительных ресурсов.[1] Метод, основанный на использовании для динамической оптимизации дополнительного микропроцессорного ядра, помимо решения вышеупомянутой проблемы позволяет повысить производительность двоичной трансляции в целом. Такой подход особенно актуален на фоне все более широкого распространения многоядерных микропроцессоров и многопроцессорных систем.

Представленные в работе схемы фоновой оптимизации были реализованы в СДДТ уровня виртуальной машины LIntel, обеспечивающей полную двоичную совместимость микропроцессора архитектуры Эльбрус [9, 10] с архитектурой Intel IA-32.

[1] Однако, как будет показано далее, производительность двоичной трансляции при этом снижается.

2. LINTEL

В основе микропроцессорной архитектуры Эльбрус лежит принцип широкого командного слова VLIW (Very Long Instruction Word). Кроме этого, она имеет ряд других особенностей, в частности, аппаратную поддержку для обеспечения полной совместимости с архитектурой IA-32 посредством прозрачной динамической двоичной трансляции.

СДДТ LIntel предназначена для эмуляции поведения машины на базе процессора Intel архитектуры IA-32 полностью совместимым и высокопроизводительным способом за счет трансляции исходных инструкций архитектуры Intel IA-32 в широкие команды архитектуры Эльбрус. При этом архитектура процессора Эльбрус двоично-несовместима с исходной платформой IA-32. LIntel обеспечивает функциональность уровня виртуальной машины, выполняя трансляцию всей иерархии программного обеспечения от BIOS до операционной системы и пользовательских приложений ”прозрачно” для конечного пользователя (рис. 1). Одной из основных особенностей системы LIntel является то, что она использует в своей работе заложенные в микропроцессоре Эльбрус механизмы аппаратной поддержки эффективной двоичной трансляции и оптимизации.

По своей структуре она аналогична другим описанным в литературе системам двоичной трансляции [11, 13] и функционально наиболее близка системе Code Morphing Software компании Transmeta [14, 15]. Так же как и перед другими СДДТ, перед LIntel стоит задача эффективного разделения ресурсов целевой платформы между нуждами собственно СДДТ и исполняемыми двоично-транслированными кодами.

2.1. Адаптивная динамическая двоичная трансляция

LIntel является адаптивной двоично-транслирующей системой (рис. 2), включающей в себя четыре уровня трансляции и оптимизации, различающиеся эффективностью результирующего кода целевой архитектуры и накладными расходами, с которыми связано его получение, а именно: интерпретатор, неоптимизирующий двоичный транслятор трасс и два оптимизирующих транслятора регионов. LIntel выполняет профилирование непосредственно во время исполнения транслируемых кодов для выделения часто исполняющихся участков кода и определения того, какой уровень оптимизации целесообразно применять при трансляции того или иного участка. Кэш трансляций позволяет хранить и переиспользовать единожды полученные двоично-транслированные коды. Система поддержки исполнения управляет всем процессом двоичной трансляции.

На начальном этапе работы программы интерпретатор декодирует и исполняет последовательно каждую x86-инструкцию, точно повторяя поведение аппаратуры процессора исходной платформы. Одновременно происходит сбор информации о частоте исполнения линейных участков x86-кода. По достижении участком некоторого порогового значения запускается его неоптимизирующая трансляция.

Неоптимизирующий транслятор создает *трассу* – семантически точное отображение одного или нескольких последовательных линейных участков исходного кода, объединенных последовательным потоком управления, в коды целевой архитектуры. Трасса формируется из шаблонов – заранее подготовленных последовательностей широких команд архитектуры Эльбрус, реализующих соответствующие инструкции архитектуры IA-32. После генерации кода трассы и подстановки актуальных констант и адресной информации она сохраняется в кэше трансляций. Транслятор трасс создает код целевой архитектуры без применения сложных оптимизационных преобразований и предназначен в первую очередь для быстрой генерации транслированных кодов, а не их оптимизации. Он позволяет существенно улучшить время “холодного” старта по сравнению с интерпретацией. В то же время, неоптимизирующую трансляцию целесообразно применять только для кодов с низким числом повторений.

Трассы снабжаются счетчиками для про-

Рис. 1. Система динамической двоичной трансляции уровня виртуальной машины LIntel.

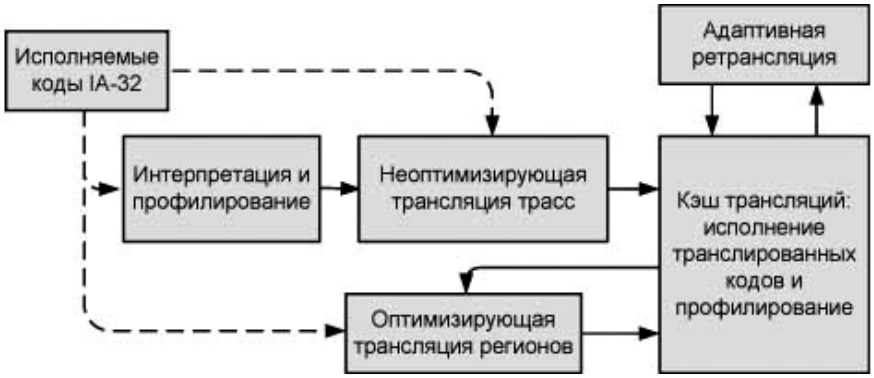

Рис. 2. Адаптивная динамическая двоичная трансляция.

филирования горячих участков кода, которые будут транслированы с уровнем оптимизации O0. Единицей оптимизирующей трансляции является *регион*. В отличие от трасс, регионы могут включать в себя линейные участки, соответствующие разным потокам управления, что открывает более широкие возможности для оптимизации и спекулятивного исполнения (что является важным источником параллелизма на уровне команд для VLIW-архитектур).

Оптимизирующий транслятор уровня O0 применяет базовые оптимизации, не требующие больших вычислительных затрат (peephole, dead-code elimination, constant propagation, code motion, redundant load elimination, superblock if-conversion, scheduling).

Оптимизирующий транслятор уровня O1 является максимальным уровнем оптимизации в LIntel. Его возможности сопоставимы с возможностями оптимизирующих компиляторов языков высокого уровня[2]. Помимо применения сложных оптимизаций (software pipelining, global scheduling, hyperblock if-conversion и пр.) он использует все свойства аппаратуры, введенные для поддержки двоичной оптимизации и исполнения оптимизированных транслированных кодов.

Регионные трансляции также сохраняются в кэше трансляций. Профилирование регионов для транслятора уровня O1 выполняется трансляциями уровня O0 непосредственно во время исполнения.

Оптимизированные трансляции не всегда улучшают производительность исполнения. Применение оптимизаций на основании

[2]Фактически, O0/O1-нотация оптимизирующих трансляторов LIntel соответствует общепринятым уровням оптимизации 02/O3-O4 компиляторов языков высокого уровня.

	Количество тактов на трансляцию одной инструкции	Производительность транслированных кодов
Неоптимизирующая трансляция	1600	0.18
O0-оптимизация	30000	0.58
O1-оптимизация	1000000	1.0

Рис. 3. Сравнение затрат на трансляцию одной инструкции IA-32 и производительности транслированных кодов (по отношению к производительности кодов, транслированных с уровнем оптимизаций O1).

(по отношению к производительности кодов, транслированных с уровнем оптимизаций O1).

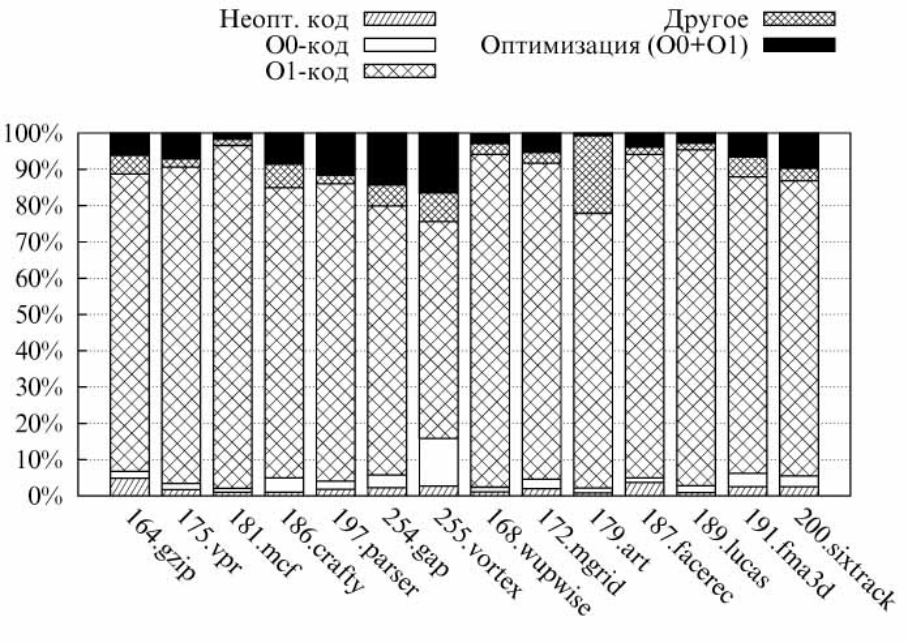

Рис. 4. Профиль двоичной трансляции (для случая последовательной динамической оптимизации).

неверных предположений о поведении кода может привести к накладным расходам на обработку соответствующих исключительных ситуаций. Это могут быть, например, некорректные спекулятивные оптимизации или включение в регион инструкций обращения в память, на самом деле обращающихся к устройствам I/O, отображенным на память (в этом случае корректность доступа к устройству не гарантируется, так как в результате оптимизаций исходные инструкции обращения в память внутри региона могут быть переупорядочены или объединены в одну широкую команду). Корректность оптимизаций контролируется аппаратно в процессе исполнения транслированных кодов. При возникновении исключительной ситуации запускается повторная трансляция региона с применением более консервативных предположений.

На рис. 3 приводится сравнение накладных расходов на трансляцию одной инструкции IA-32 и производительности транслированных кодов для разных уровней трансляции и оптимизации. Адаптивность системы позволяет определять наиболее подходящий уровень для поддержания

оптимального соотношения накладных расходов и производительности в процессе исполнения исходных двоичных кодов.

Диаграмма на рис. 4 на примере задач SPEC2000, запущенных в системе Linux (при этом сама система также исполняется посредством динамической двоичной трансляции) демонстрирует распределение времени работы разных фаз системы LIntel в процессе исполнения двоичных кодов. И хотя большую часть времени исполнения задачи работают транслированные коды, затраты на оптимизирующую трансляцию существенны и в среднем составляют 7%.

2.2. Обработка асинхронных прерываний

Одной из функций системы поддержки двоичной трансляции является обработка асинхронных (внешних) прерываний. Механизм отложенного прерывания позволяет повысить эффективность работы двоично-транслированных кодов и действий системной поддержки по доставке внешних прерываний за счет явного выделения мест приема прерываний. По приходу прерывания, системная поддержка просто запоминает этот факт и возвращает управление исполняемому коду. Во время интерпретации факт наличия отложенного прерывания проверяется перед исполнением очередной команды. В неоптимизированной трассе, из соображений эффективности, проверка и доставка асинхронного прерывания в двоично-транслированный код осуществляется только в начале очередного линейного участка. В коде региона оптимизирующий транслятор расставляет специальные операции проверки прихода асинхронного прерывания, вызывающие прерывание исполнения региона при наличии такового, в тех местах кода, где гарантируется, что контекст исходной архитектуры сведен, то есть соответствует состоянию, когда исполнение всех предыдущих инструкций завершено, а следующих – не начиналось.

Такая расстановка команд проверки наличия прерывания в интерпретаторе и двоично-транслированных кодах, с одной стороны, позволяет не накладывать ограничений на планирование прочих участков кода из предположения о возможном приходе прерывания в неопределенный момент времени и необходимости предоставления системной поддержке достаточной информации для корректного прерывания исполнения текущего кода и перехода к доставке прерывания. При этом система LIntel достаточно оперативно реагирует на внешние события.

Узким местом такого подхода является наличие в системе фазы оптимизирующей трансляции. Если во время ее выполнения в систему приходит внешнее прерывание, оно может быть обработано только по завершении работы транслятора (рис. 5), возможность прерывания которого изначально не была предусмотрена. В силу сложности проводимых оптимизационных преобразований, фаза оптимизирующей трансляции потребляет значительную часть процессорного времени (в зависимости от уровня оптимизации – от нескольких десятков тысяч до нескольких миллионов тактов на одну команду исходной платформы), и, как следствие, задержка реакции системы на внешнее событие может оказаться заметной (оценка времени доставки внешних прерываний обработчику прерываний приведена в разд. 3.2).

3. ФОНОВАЯ ОПТИМИЗАЦИЯ

Для решения проблемы влияния оптимизирующей трансляции на производительность и латентность системы LIntel в ней был реализован принцип фоновой оптимизации.

Принцип фоновой оптимизации заключается в выполнении фазы оптимизирующей трансляции одновременно (или псевдоодновременно) с выполнением основной задачи системы динамической двоичной трансляции – исполнением транслированных кодов. Двоичные трансляторы уровня приложений, как правило, используют для этой цели интерфейс многопоточного исполнения и планирования задач, предоставляемый операционной системой целевой платформы. В двоичных трансляторах уровня виртуальной машины для поддержки фоновой оптимизации необходима внутренняя реализация многопоточности.

Далее в разделе рассматриваются две реализации принципа фоновой оптимизации и

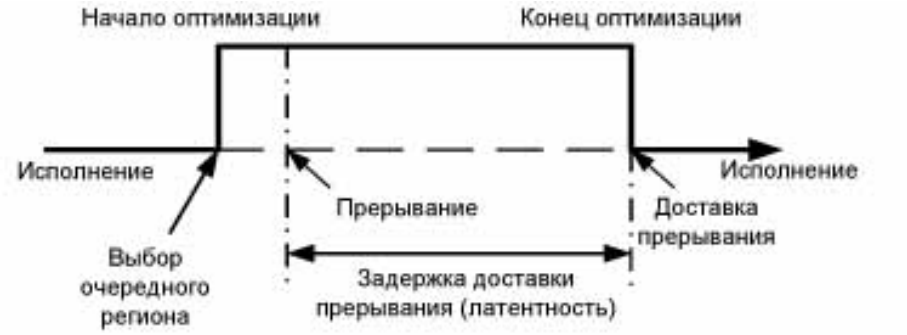

Рис. 5. Задержка реакции системы на внешнее прерывание, обусловленная фазой оптимизирующей трансляции.

разделения процессорного времени между оптимизирующей трансляцией и исполнением: в первом случае система LIntel имеет в своем распоряжении одно микропроцессорное ядро, во втором – два.

Для демонстрации эффекта от применения принципа фоновой оптимизации используются задачи из пакета SPEC2000.

3.1. Потоки исполнения и оптимизации

В системе LIntel была реализована инфраструктура многопоточного исполнения, поддерживающая один поток исполнения и один поток оптимизирующей трансляции. Поток исполнения включает в себя весь процесс двоичной трансляции и исполнения исходных двоичных кодов, за исключением оптимизирующей трансляции (уровней O0 и O1), а именно: интерпретацию, неоптимизирующую трансляцию, системную поддержку и исполнение транслированных кодов. Оптимизирующий транслятор запускается в отдельном потоке – потоке оптимизации – после выделения в процессе исполнения и профилирования очередного региона "горячего" кода. По окончании оптимизирующей трансляции сгенерированный код региона возвращается потоку исполнения, который размещает его в кэше трансляций.

Во время проведения оптимизирующей трансляции региона соответствующий ему исходный код исполняется либо посредством интерпретации, либо посредством полученного ранее транслированного кода более низких уровней оптимизации. Запуск набора новых регионов блокируется до завершения оптимизации текущего.

К концу оптимизации страницы памяти, содержащие исходный код, вошедший в транслированный регион, могут стать невалидными (в результате DMA или самомодификации кодов). Поэтому перед размещением транслированного кода региона в кэше трансляций поток исполнения проверяет целостность соответствующих страниц памяти и отказывается от региона при необходимости. Когерентность исходных и транслированных (в том числе – находящихся в процессе трансляции) кодов контролируется предусмотренным в архитектуре механизмом физической защиты памяти [16].

Разделение потоков исполнения и оптимизации позволяет организовать распределение процессорных ресурсов между ними так же, как это происходит в многозадачных операционных системах. В случае LIntel были реализованы две простые стратегии разделения процессорного времени для одно- и двухпроцессорной систем.

3.2. Фоновая оптимизация в системе с одним ядром

В однопроцессорной системе фоновая оптимизация реализуется посредством квантования потока оптимизации. А именно, во время работы фазы оптимизирующей трансляции региона происходит попеременное переключение процессора между двумя потоками по прерываниям от локального

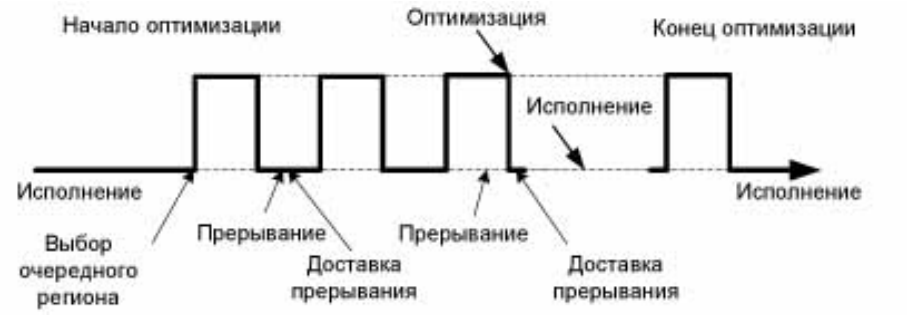

Рис. 6. Доставка асинхронного прерывания в случае квантования оптимизации.

таймера, используемого для нужд СДДТ. Каждому потоку выделяется фиксированный квант времени, и именно в этот квант времени работы потока исполнения пришедшее во время работы фазы оптимизирующей трансляции внешнее прерывание имеет шанс быть обработанным (рис. 6). После окончания трансляции региона поток исполнения снова получает в свое распоряжение все процессорное время.

Для демонстрации была выбрана простая стратегия разделения процессорного времени, при которой оба потока имеют одинаковый приоритет, то есть равные кванты времени. Как видно на рис. 7, время доставки прерывания в такой схеме существенно улучшается.

Однако по сравнению со схемой, когда оптимизация выполняется последовательно, в целом производительность динамической двоичной трансляции при этом снижается (рис. 8).

В значительно степени это объясняется тем, что из-за “растягивания” фазы оптимизации оптимизированные трансляции вводятся в процесс исполнения позже, чем при последовательной оптимизации. В результате этого бóльшую (чем в последовательной схеме) часть исходного приложения исполняется интерпретатором или менее оптимизированными кодами. Кроме того, возникают дополнительные накладные расходы, связанные с переключением контекстов потоков.

В целом, однопроцессорная реализация фоновой оптимизации в LIntel не является приоритетной. Однако мы полагаем, что ее эффективность может быть повышена за счет вариации таких параметров, как соотношения квантов времени потоков исполнения и оптимизации, порогов набора регионов и т.д. – благодаря чему можно было бы добиться более раннего ввода оптимизированных кодов в процесс исполнения, в то же время сохраняя на приемлемом уровне латентность системы. Кроме того, например, инструкция “halt” может использоваться в качестве подсказки для переключения на поток трансляции до истечения кванта времени потока исполнения. Подробное исследование разделения процессорного времени между потоками исполнения и трансляции для рассматриваемого случая приводится в [17].

3.3. Фоновая оптимизация в системе с двумя ядрами

В двухпроцессорной системе для выполнения фоновой оптимизации в LIntel используется отдельное (незанятое в противном случае) микропроцессорное ядро. Поток исполнения, теперь эксклюзивно использующий ресурсы своего ядра, прерывается только на набор очередного региона и размещение в кэше трансляций его оптимизированной версии, полученной от потока оптимизации.

Как видно из рис. 10, в этом случае удается не только исключить высокую латентность системы, но и повысить производительность двоичной трансляции в целом.

Полученный эффект (ускорение на 6% в среднем) хорошо согласуется со средней величиной затрат на оптимизацию, приведенной на диаграмме профиля двоичной трансляции в схеме с последовательной оптимизацией (рис. 4).

	Последовательная оптимизация	Квантование оптимизации
среднее время работы фазы O1	1.54 с	3 с
максимальное время работы фазы O1, T_{01_max}	8.8 с	29.5 с
среднее время доставки прерывания вне работы фазы оптимизации	54 мкс	
максимальное время доставки прерывания (пришедшего во время работы фазы O1)	8.8 с (T_{01_max})	1.7 мс

Рис. 7. Время доставки прерывания (частота процессора = 300 MHz; размер кванта времени потока = 50000 тактов). Для сравнения берется уровень оптимизации O1, так как он потребляет больше тактов на трансляцию одной исходной инструкции, чем уровень O0.

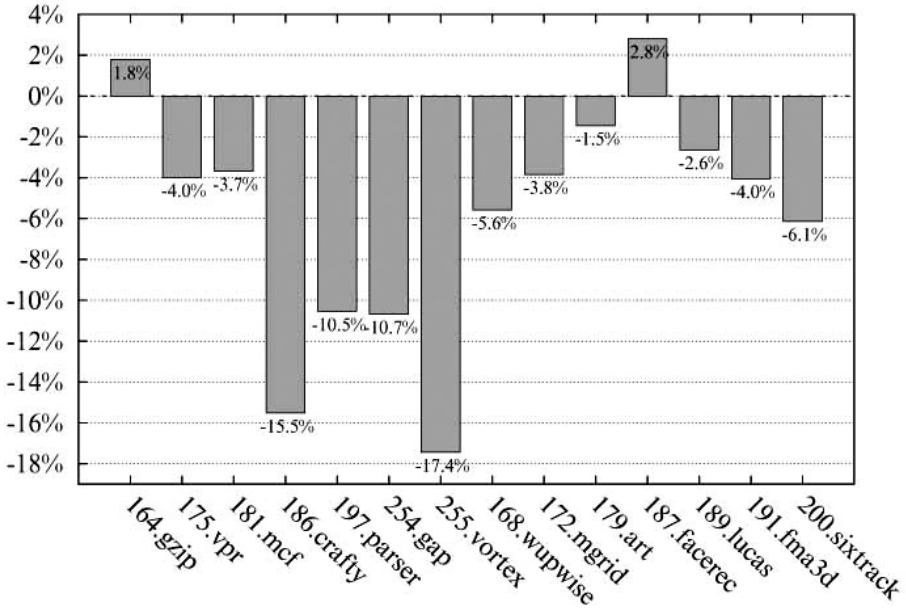

Рис. 8. Снижение производительности двоичной трансляции при квантовании оптимизации (по сравнению с последовательной оптимизацией).

3.4. Обсуждение и развитие

На примере профиля динамической двоичной трансляции тестов SPEC2000 (рис. 4) видно, что при такой доле оптимизирующей трансляции микропроцессорное ядро, закрепленное за потоком оптимизации, бóльшую часть времени исполнения задачи будет простаивать. Для повышения коэффициента его использования можно более часто запускать оптимизирующий транслятор, чего можно достичь посредством динамического понижения порогов набора регионов в зависимости от текущей загрузки оптимизирующего ядра.

Закономерным является вопрос о том, почему не использовать незанятое микропроцессорное ядро для исполнения исходных кодов. Другими

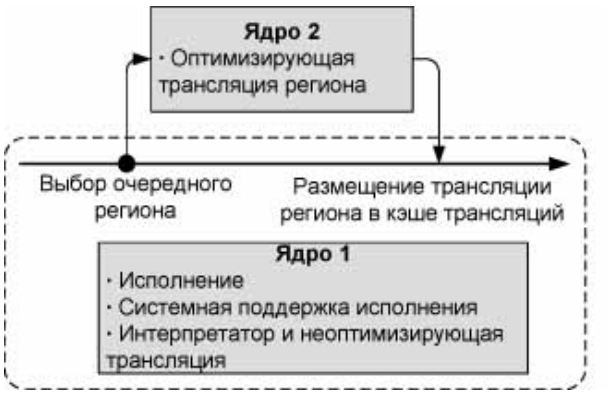

Рис. 9. Использование отдельного ядра для выполнения оптимизирующей трансляции.

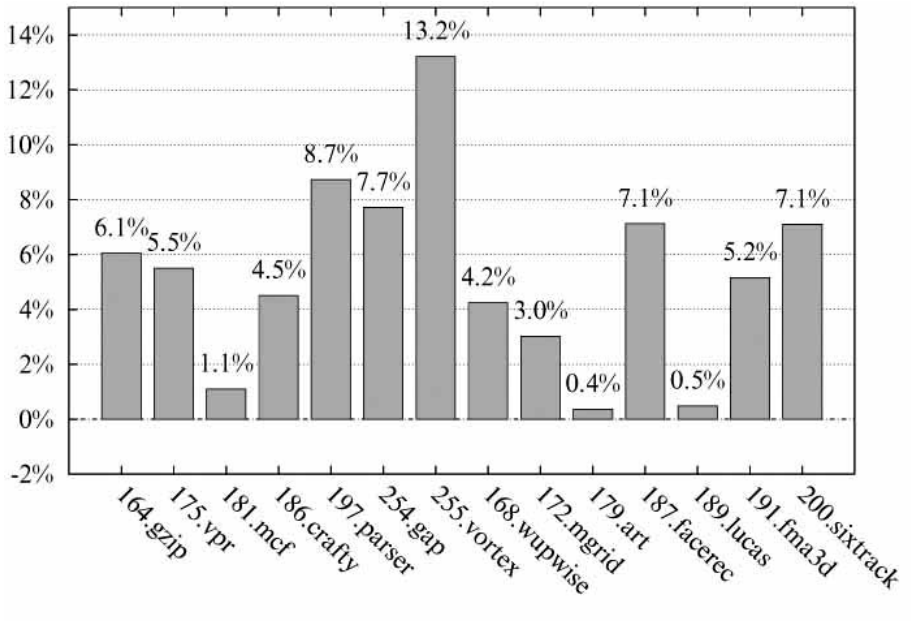

Рис. 10. Повышение прозводительности двоичной трансляции при выносе оптимизации на отдельное ядро (по сравнению с последовательной оптимизацией).

словами, если в системе имеется несколько микропроцессорных ядер целевой платформы, системное программное обеспечение исходной платформы должно “видеть” такое же количество микропроцессорных ядер исходной платформы. В реализации архитектуры Эльбрус, доступной на момент подготовки работы, существуют архитектурные ограничения, не позволяющие этого сделать (ввиду несоответствия требованиям системы команды IA-32 в части организации многопроцессорных систем). Однако мы считаем, что вынос динамической оптимизации на отдельные микропроцессорные ядра имеет право на существование, по крайней мере, по следующим причинам:

- различные классы программного обеспечения, не всегда поддерживающие многозадачность или многопоточность (напр., программное обеспечение для встраиваемых систем), все еще могут использовать преимущества многоядерных микропроцессоров и систем за счет выноса динамической оптимизации на незанятые вычислительные ресурсы;

- на фоне все более широкого распространения многоядерных микропроцессоров и многопроцессорных систем видится оправданным использование части вычислительных ресурсов для эффективной реализации технологии динамической двоичной трансляции; причем потребителем этих ресурсов может быть не только оптимизирующий транслятор; в общем случае, в параллель с основным потоком исполнения могут выполняться такие задачи, как выбор кода для оптимизации [18], программная предподкачка данных [19], сохранение оптимизированных кодов в базе кодов [20] на внешнем носителе[3] и т.д.

В целом, перспективным видится построение такой системы динамической двоичной трансляции, которая могла бы поддерживать существование произвольного количества потоков исполнения (в терминах виртуальной машины исходной архитектуры – в результате чего, например, операционная система исходной платформы могла бы "видеть" более одного микропроцессорного ядра), оптимизации и т.д., и эффективно распределять эти потоки на имеющиеся вычислительные ресурсы в зависимости от их количества, а также характера и поведения задач исходной платформы.

4. ЗАКЛЮЧЕНИЕ

В статье рассмотрено применение принципа фоновой оптимизации для снижения накладных расходов в системе динамической двоичной трансляции, обусловленных наличием фазы оптимизирующей трансляции. Представлена реализация этого принципа для систем с одним и двумя микропроцессорными ядрами. В обоих случаях выполнение динамической оптимизации в фоне позволяет решить проблему высокой латентности системы, которая возникает при последовательной динамической оптимизации, что особенно важно для СДДТ уровня виртуальной машины. В двухпроцессорной системе вынос динамической оптимизации на незанятое ядро позволяет полностью исключить связанные с ней накладные расходы из времени исполнения исходных кодов и тем самым повысить производительность двоичной трансляции в целом.

[3] Асинхронная работа с базой кодов в системе LIntel реализована уже сейчас, однако в настоящей статье мы рассматриваем только эффект от реализации принципа фоновой оптимизации.

СПИСОК ЛИТЕРАТУРЫ

1. *Smith J., Nair R.* Virtual Machines: Versatile Platforms for Systems and Processes (Morgan Kaufmann, 2005).
2. *Campanoni S., Agosta G., Reghizzi S.C.* ILDJIT: a parallel dynamic compiler, in VLSISoC' 08: Proceedings of the 16th IFIP/IEEE International Conference on Very Large Scale Integration (2008). P. 13–15.
3. *Krintz C.J. , Grove D., Sarkar V., Calder B.* Reducing the overhead of dynamic compilation, Software: Practice and Experience. V. 31 I. 8. 717 (2001).
4. *Mars J.* Satellite optimization: The offloading of software dynamic optimization on multicore systems (poster), in PLDI '07: 2007 ACM SIGPLAN conference on Programming language design and implementation (2007).
5. *Unnikrishnan P., Kandemir M., Li F.* Reducing dynamic compilation overhead by overlapping compilation and execution, in Proceedings of the 11th South Pacific Design Automation Conference (ASP-DAC '06) (IEEE Press, Piscataway, NJ, USA, 2006). P. 929–934.
6. *Voss M.J., Eigenmann R.* A framework for remote dynamic program optimization, in Proceedings of the ACM SIGPLAN workshop on Dynamic and adaptive compilation and optimization (2000). P. 32–40.
7. *Zhang W., Calder B., Tullsen D.M.* An event-driven multithreaded dynamic optimization framework, in Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT '05) (IEEE Computer Society, Washington, DC, USA, 2005). P. 87–98.
8. *Guan H., Liu B., Li T., Liang A.* Multithreaded optimizing technique for dynamic binary translator CrossBit, Computer Science and Software Engineering, International Conference on 5, 945 (2008).
9. *Babayan B.* E2k technology and implementation, in Euro-Par '00: Proceedings from the 6th International Euro-Par Conference on Parallel Processing (Springer-Verlag, London, UK, 2000). P. 18–21.

10. *Volkonskiy V.* Optimizing compilers for Elbrus-2000 (E2k) architecture, in 4th Workshop on EPIC Architectures and Compiler Technology (2005).

11. *Baraz L., Devor T., Etzion O., Goldenberg S., Skaletsky A., Wang Y., Zemach Y.* IA-32 Execution Layer: a two-phase dynamic translator designed to support IA-32 applications on Itanium-based systems, in MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture (IEEE Computer Society, Washington, DC, USA, 2003). P. 191.

12. *Chernoff A., Herdeg M., Hookway R., Reeve C., Rubin N., Tye T., Yadavalli S.B., Yates J.* FX!32: A profle-directed binary translator, IEEE Micro 18, 56 (1998).

13. *Gschwind M., Altman E.R., Sathaye S., Ledak P., Appenzeller D.* Dynamic and transparent binary translation, Computer 33, 54 (2000), ISSN 0018-9162.

14. *Dehnert J.C., Grant B.K., Banning J.P., Johnson R., Kistler T., Klaiber A., Mattson J.* The Transmeta Code Morphing Software: Using speculation, recovery, and adaptive retranslation to address real-life challenges, in Proceedings of the First Annual IEEE/ACM International Symposium on Code Generation and Optimization (2003).

15. *Klaiber A.* The technology behind Crusoe processors, Tech. Rep., Transmeta Corporation (2000).

16. *Ермолович А.* Методы повышения производительности двоично-транслирующих систем с аппаратной поддержкой, диссертация на соискание степени кандидата технических наук. Институт микропроцессорных вычислительных систем (2003).

17. *Kulkarni P., Arnold M., Hind M.* Dynamic compilation: the benefits of early investing, in VEE '07: Proceedings of the 3rd international conference on Virtual execution environments (ACM, New York, NY, USA, 2007). P. 94–104.

18. *Mars J., Soffa M.L.* MATS: Multicore adaptive trace selection, in Proceedings of the 3rd Workshop on Software Tools for MultiCore Systems (STMCS 2008) (2008).

19. *Mars J., Williams D., Upton D., Ghosh S., Hazelwood K.* A reactive unobtrusive prefetcher for multicore and manycore architectures, in Proceedings of the Workshop on Software and Hardware Challenges of Manycore Platforms (SHCMP) (2008).

20. *Ермолович А.* База кодов как средство повышения эффективности динамический двоично-транслирующей системы, Информационные технологии 9, 14 (2003).