На правах рукописи

Гимпельсон Вадим Дмитриевич

Сокращение длины критических путей при динамической трансляции двоичных кодов

05.13.11 – математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Автореферат

диссертации на соискание учёной степени
кандидата физико-математических наук

Москва – 2018

Работа выполнена в АО «МЦСТ».

Научный руководитель: **Волконский Владимир Юрьевич,**
кандидат технических наук, старший научный сотрудник, начальник отделения «Системы программирования» публичного акционерного общества «ИНЭУМ им. И.С. Брука»

Официальные оппоненты: **Галатенко Владимир Антонович,**
доктор физико-математических наук, заведующий сектором Федерального государственного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук».

Новиков Сергей Викторович,
кандидат технических наук, принципиальный инженер компании Intel, руководитель направления бинарной трансляции.

Ведущая организация: Акционерное общество «Институт точной механики и вычислительной техники имени С.А. Лебедева Российской академии наук»

Защита состоится «19» апреля 2018 г. в 16 часов на заседании диссертационного совета Д 002.087.01 при Институте системного программирования им. В.П. Иванникова РАН по адресу: 109004, Москва, ул. А. Солженицына, д. 25.
С диссертацией можно ознакомиться в библиотеке и на сайте Федерального государственного бюджетного учреждения науки Институт системного программирования им. В.П. Иванникова Российской академии наук.

Автореферат разослан «____» _____ 2018 г.

Учёный секретарь
диссертационного совета Д 002.087.01,
кандидат физико-математических наук **Зеленов С.В.**

Актуальность

Повышение производительности является одним из главных направлений развития вычислительной техники. Важнейшим способом увеличения производительности микропроцессоров является логическое совершенствование внутренней архитектуры процессорного ядра, позволяющее извлекать параллелизм на уровне отдельных операций как чисто аппаратными, так и аппаратно-программными методами.

В суперскалярных архитектурах анализ зависимостей между отдельными операциями и их распараллеливание выполняются на аппаратном уровне при исполнении программы, что ограничивает параллелизм из-за сложности анализа. Архитектуры с явно выраженной параллельностью на уровне команд (*EPIC)* перекладывают всю работу по распараллеливанию на уровне операций с аппаратуры на компилятор, что упрощает аппаратуру и позволяет увеличить параллелизм по сравнению с суперскалярными архитектурами. Однако для быстрого массового распространения таких архитектур с несовместимой системой команд необходимо решение, обеспечивающее исполнение огромного объема существующего программного обеспечения в двоичных кодах.

Эффективным способом обеспечения двоичной совместимости является *двоичная трансляция*. Суть этого метода заключается в программной перекомпиляции (перетрансляции) двоичных кодов платформы, с которой необходимо обеспечить совместимость, в коды новой платформы.

Из-за особенностей двоичного кода программ двоичный транслятор должен быть динамическим, то есть работать во время исполнения программы. В силу этого на скорость трансляции должны быть наложены очень жёсткие ограничения. С другой стороны, для получения быстрого результирующего кода необходимо проводить различные оптимизации, часто тяжеловесные. Для решения этой проблемы двоичный транслятор имеет несколько уровней трансляции, различающихся между собой качеством и количеством проводимых оптимизаций кода, а также временем работы. Транслятор верхнего уровня

(дающий самый быстрый параллельный код целевой архитектуры), будем далее называть *двоичным оптимизирующим транслятором*.

Эффективным методом, позволяющим распараллелить вычисления в ациклических областях, являются оптимизации, производящие *сокращение длины критического пути* (или просто *сокращение критического пути*) за счет использования возможностей архитектуры с явным параллелизмом операций.

Для уменьшения времени выполнения циклов применим класс оптимизаций, который получил название *"конвейеризация циклов методом наложения итераций"* или, в англоязычной литературе, *"программная конвейеризация циклов"* (software pipelining). Суть этого класса оптимизаций заключается в том, что возможно совместить вычисление нескольких итераций цикла в одной итерации цикла результирующего кода.

Методы сокращения длины критических путей в ациклических и циклических областях широко использовались и используются в языковых компиляторах: IMPACT-I, компиляторы icc и gcc для архитектуры Itanium, языковом компиляторе для архитектуры «Эльбрус». Также ряд алгоритмов был разработан в рамках академических работ. Однако при трансляции двоичных кодов возникает ряд дополнительных ограничений, которые не позволяют использовать разработанные ранее алгоритмы. Во-первых, это дополнительные семантические свойства, накладываемые особой спецификой двоичной трансляции. Пожалуй, основным семантическим ограничением является *задача обеспечения точного состояния регистров и памяти исходной архитектуры при возникновении прерываний и исключений*. Вторым важным требованием, предъявляемым к алгоритмам оптимизаций, является *собственная скорость работы этих алгоритмов*.

Необходимость создания быстрых алгоритмов сокращения критического пути и конвейеризации циклов в двоичном оптимизирующем трансляторе, позволяющих максимально использовать возможности архитектуры с явно выраженной параллельностью на уровне команд, и как следствие, достижение эффективной совместимости, определяет актуальность диссертационной работы.

Цель исследования

Целью диссертационной работы является повышение эффективности двоичного оптимизирующего транслятора за счёт адаптации старых и разработки новых алгоритмов сокращения длины критического пути в циклических и ациклических областях для архитектур с явно выраженной параллельностью на уровне команд. Эти алгоритмы должны учитывать особенности оптимизирующего двоичного транслятора: специфическое окружение и требования по скорости работы. В соответствии с этими целями были определены следующие задачи:

- провести анализ современного уровня развития методов сокращения длины критического пути в циклических и ациклических областях и возможностей их использования в двоичном оптимизирующем трансляторе;
- разработка новых алгоритмов сокращения длины критического пути в ациклических областях;
- разработка новых алгоритмов сокращения длины критического пути в циклических областях, основанных на конвейеризации циклов;
- интеграция методов сокращения длины критического пути в ациклических областях с алгоритмом конвейеризации циклов;
- разработка метода интеграции алгоритма сокращения длины критического пути с другими имеющимися оптимизирующими преобразованиями с целью устранения ситуаций, когда качество некоторой оптимизации зависит от результатов работы алгоритма сокращения длины критического пути, а качество алгоритма сокращения длины критического пути зависит от этой оптимизации;
- обеспечение во всех разрабатываемых алгоритмах семантических требований, накладываемых двоичной трансляцией; обеспечение высокой скорости работы всех разрабатываемых алгоритмов;
- реализация указанных алгоритмов в динамическом двоичном трансляторе.

Научная новизна

Научной новизной обладают следующие результаты работы:

- быстрый алгоритм сокращения длины критического пути в ациклических областях;
- схема взаимодействия алгоритма сокращения длины критического пути в ациклических областях с другими оптимизирующими преобразованиями для преодоления “замкнутого круга”, когда оптимизации являются взаимозависимыми;
- алгоритм разметки времён раннего и позднего планирования на расширенном графе зависимостей;
- алгоритм конвейеризации циклов в динамическом двоичном оптимизирующем трансляторе;
- интеграция алгоритмов сокращения длины критического пути с алгоритмом конвейеризации циклов;
- сформулированы и доказаны теоремы, показывающие корректность и оптимальность предложенных алгоритмов, а также произведена оценка сложности этих алгоритмов.

Результаты работы, выносимые на защиту

В процессе проведения диссертационного исследования были получены следующие результаты, выносимые на защиту:

- разработка и реализация быстрого алгоритма сокращения длины критического пути в ациклических областях; теорема об оптимальности предложенного алгоритма;
- алгоритм разметки времён раннего и позднего планирования на расширенном графе зависимостей; сформулированы и доказаны теоремы о корректности, оптимальности и сложности предложенного алгоритма;
- алгоритм конвейеризации циклов в динамическом двоичном оптимизирующем трансляторе; сформулирована и доказана теорема о сложности предложенного алгоритма; интеграция методов сокращения

длины критического пути в ациклических областях с алгоритмом конвейеризации циклов.

Теоретическая и практическая значимость

Предложен алгоритм сокращения длинны критического пути в ациклических областях. Предложен алгоритм разметки времён на расширенном графе зависимостей и основе него разработан алгоритм конвейеризации циклов.

Практическая ценность диссертационной работы состоит в том, что на основе предложенных алгоритмов удалось значительно повысить эффективность работы динамического двоичного транслятора для архитектуры “Эльбрус”. Было показано, что аппаратная техника вращающихся регистров может эффективно применяться в двоичном трансляторе. Результаты исследований были реализованы в следующих программных и аппаратных системах:

- динамический двоичный оптимизирующий транслятор уровня всей системы с архитектуры Intel x86 на архитектуру “Эльбрус”, разработанный в АО “МЦСТ”;
- динамический двоичный оптимизирующий транслятор уровня приложений ОС Linux с архитектуры Intel x86 на архитектуру “Эльбрус”, разработанный в АО “МЦСТ”;
- статический оптимизирующий транслятор с архитектуры Intel x86 на архитектуру IPF (Itanium), разработанный в АО “МЦСТ” в рамках совместного проекта с Intel Corporation;
- микропроцессор Эльбрус, разработанный в АО “МЦСТ”, обеспечивающий совместимость с архитектурой Intel x86;
- динамический двоичный транслятор уровня приложений ОС Linux, разработанный в ООО “Эльбрус Технологии”.

Предложенные алгоритмы также могут быть использованы в оптимизирующих компиляторах и двоичных трансляторах для различных архитектур.

Апробация.

Основные результаты диссертационной работы докладывались на

следующих научно-технических конференциях и семинарах:

- 5th RISC-V Workshop, Mountain View, CA, November 29-30, 2016.
- Open Conference on Compiler Technologies, Москва, РАН, 2015 г.
- Samsung Compiler Workshop, Москва, Samsung Office, 2014 г.
- На XXXIV Международной молодёжной научной конференции “Гагаринские чтения”, Москва, МАТИ, 2008 г.
- Международная научная конференция, посвящённая 80-летию со дня рождения академика В.А. Мельникова, 2008 г.
- На XXIII научно-технической конференции “Направления развития и применения перспективных вычислительных средств и новый информационных технологий в ВВТ РКО”, Москва, в/ч 03425, 2007 г.
- На XXXIII Международной молодёжной научной конференции “Гагаринские чтения”, Москва, МАТИ, 2007 г.
- На XXI научно-технической конференции войсковой части 03425. Москва, в/ч 03425, 2003 г.
- На семинарах секции программного обеспечения ЗАО “МЦСТ” в 2005-2016 годах.

Публикации.

По теме диссертации опубликовано 13 печатных работ [1]-[13]. Работы [3], [4], [7], [10], [12] опубликованы в изданиях из перечня ВАК. В работе [12] описаны методы сокращения длины критических путей в ациклических областях. Личный вклад автора заключается в разработке и реализации быстрого алгоритма минимизации высоты графа зависимостей. В работах [4] и [8] представлены методы сокращения длины критических путей в циклах. В работах [1] и [2] личный вклад автора заключается в переносе алгоритмов, изложенных в данной диссертационной работе, в динамические двоичные трансляторы из x86 в ARM и из RISC-V в x86. Работа [3] написана совместно с несколькими авторами. Личный вклад автора в этой работе заключается в описании общей схемы функционирования динамического двоичного транслятора, а также в описании

оптимизирующего двоичного транслятора, из x86 в «Эльбрус». В работе [7] личный вклад автора заключается в постановке задачи и в разработке методов коррекции профильной информации в случае её не консистентности. В работе [10] личный вклад автора заключается в предложенных методах активации оптимизаций в двоичном трансляторе. Совместная работа [13] посвящена методам обеспечения точного состояния контекста в двоичном трансляторе. Личный вклад автора заключается в реализации и поддержке этих методов в алгоритмах сокращения длины критического пути. В ходе выполнения работы было получено свидетельство о государственной регистрации программ для ЭВМ [1, см. стр. 24].

Личный вклад автора.

Все представленные в диссертации результаты получены лично автором.

Структура и объём работы.

Работа состоит из введения, четырёх глав, заключения и двух приложений. Основной текст диссертации (без приложений и списка литературы) занимает 147 страниц, общий объем – 166 страниц. Список литературы содержит 113 наименований.

Краткое содержание работы

Во введении обосновывается актуальность темы работы, сформулированы цели и задачи, описывается практическая значимость результатов.

В **первой главе** производится обзор технологии двоичной трансляции, EPIC архитектур, внутреннего представления в трансляторе и ускорения результирующего кода за счёт сокращения длинны критического пути.

Для новых архитектур двоичная трансляция является эффективным методом обеспечения совместимости со старыми архитектурами. Однако для достижения высокой производительности для EPIC архитектур двоичный транслятор должен иметь развитый оптимизатор. Оптимизации, реализованные в двоичном трансляторе, должны удовлетворять двум важным условиям. Первое –

это высокая скорость работы, так как время трансляции входит во время работы результирующего кода. Второе – это дополнительные ограничения на проводимые преобразования и оптимизации, накладываемые семантикой двоичных кодов.

Говорят, что между операциями есть *зависимость*, если операции должны быть упорядочены по какой-то причине. Например, если первая операция читает некоторый регистр, а вторая пишет в этот регистр, то необходимо сохранить порядок их исполнения и, следовательно, между ними есть зависимость. Каждая зависимость имеет длину, которая равна количеству тактов, на которое должны быть разнесены операции. *Графом зависимостей* называется граф, узлами которого являются операции данного участка кода, а дуги – зависимости между операциями. Самый длинный путь в графе зависимостей называется *критическим путём*. *Временем раннего планирования* операции называется самый длинный путь от начала блока до операции. Если взять время раннего планирования выхода и вычесть из него длину пути от операции до этого выхода и взять минимум этих величин по всем путям и всем выходам, то полученная величина называется *временем позднего планирования* операции.

Одной из важнейших особенностей EPIC архитектур является то, что за одну инструкцию(такт) можно выполнить несколько операций. Вследствие этого оказывается, что время исполнения данного участка кода определяется именно длинной критического пути. То есть получается, что есть свободные места в инструкции, но их нельзя заполнить, так как зависимости мешают этому. Существует ряд подходов, способных сократить длину критических путей. Разделим рассмотрение этих методов на два класса. Первый – методы, применяемые в ациклических областях, второй – методы, применяемые в циклах.

Ациклические области. *Разрывом зависимости* называется преобразование, основной целью которого является устранение зависимости. Методы разрыва зависимостей можно разделить на две группы. Первая – это преобразования без построения новых операций. Такие преобразования практически всегда дают положительный эффект и их можно применять всегда, не опасаясь получить

негативный эффект. Вторая группа – это преобразования, требующие построения новых операций. Такие преобразования, применяемые всегда, когда это возможно, могут оказать отрицательный эффект. Это связно с тем, что количество операций, выполняемых в одной инструкции, ограничено, и построение большого количества новых операций может привести к тому, что потребуется большее количество инструкций, чем требовалось изначально. В связи с этим возникает необходимость разработки алгоритмов, которые бы учитывали негативный эффект от разрыва зависимостей и осуществляли бы преобразования, только тогда, когда это необходимо. Такие алгоритмы будем называть *алгоритмами минимизации высоты графа зависимостей*.

Циклические области. В силу особой регулярной структуры циклов к ним применим класс оптимизаций, получивший название *конвейеризация циклов*. В циклах, как и в ациклических областях, присутствует большое количество зависимостей, которые можно разорвать, поэтому алгоритм конвейеризации должен быть интегрирован с техниками разрыва зависимостей. Основная идея конвейеризации циклов следующая: для увеличения параллельности в вычислениях возможно совместить исполнение нескольких логических итераций цикла в одной физической итерации цикла. То есть операции со следующей итерации могут начать выполняться ещё до того, как предыдущая итерация будет закончена.

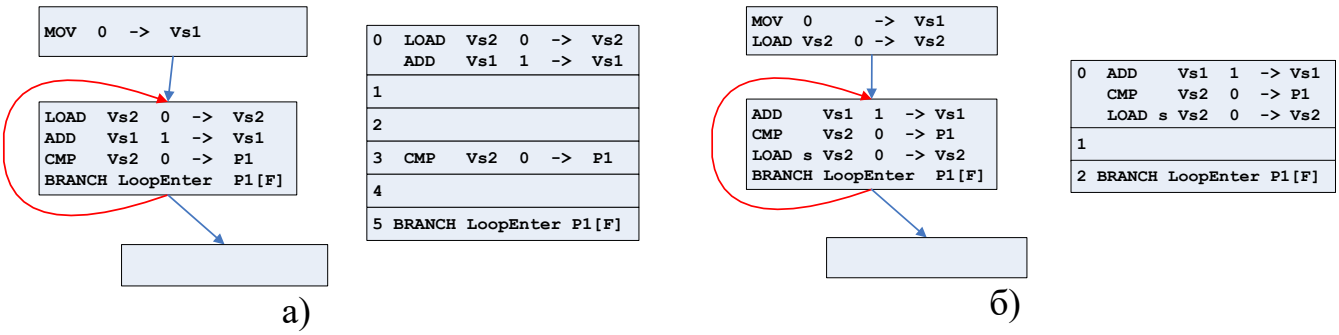

Рисунок 1. Пример конвейеризации цикла.

На рис. 1, а) изображён простой цикл подсчёта количества элементов списка. В данном примере можно перенести операцию `LOAD` вверх с дублированием. Одна копия попадает в верхний узел, вторая проносится по

обратной дуге, переводится в спекулятивный режим (рис. 1, б)). Как видно из рисунков, в результате преобразования размер цикла сократился с шести тактов до трёх.

Алгоритмы сокращения длины критических путей позволяют существенно повысить скорость работы результирующего кода. Однако в случае трансляции двоичных кодов появляется ряд дополнительных ограничений и требований: высокая скорость работы самого алгоритма, специфическая семантика двоичных кодов, большее количество разрываемых зависимостей, чем при языковой трансляции. В связи с этим была поставлена задача разработки: (i) быстрых алгоритмов минимизации высоты графа зависимостей без построения новых операций; (ii) быстрого алгоритма минимизации высоты графа зависимостей с построением новых операций; (iii) разработка алгоритма конвейеризации циклов, интегрированного с методами разрыва зависимостей.

Во **второй главе** рассматриваются три метода разрыва зависимостей в ациклических областях без построения новых операций и соответствующие им алгоритмы сокращения длины критических путей.

Первый метод – переименование регистров. Алгоритм переименования регистров основывается на счётчиках использования регистров и на компонентах потокового графа. *Счётчик использования регистра* содержит информацию о том, сколько раз данный регистр используется в качестве аргумента или результата в участке транслируемого кода. *Графом потока данных* называется граф, узлами которого являются аргументы и результаты операций, а входящие дуги для аргумента отражают информацию о том, какие операции могут выработать этот аргумент. *Компонентой потокового графа* называется любой максимально связанный подграф графа потока данных.

Алгоритм переименования регистров

1. **Цикл** по результатам всех операций
2. Для текущего результата находим всю компоненту потокового графа
3. Если количество использований регистра, соответствующего компоненте, больше количества элементов в компоненте, то всем элементам компоненты назначаем новый уникальный регистр и корректируем счётчики использований регистров.
4. **Конец цикла.**

Предлагаемый алгоритм производит полное переименование регистров, то есть всё, что можно переименовать, переименовывается. Каждая компонента потокового графа получает свой собственный уникальный регистр. Алгоритм является быстрым за счёт использования счётчиков использования регистров.

Второй метод – использование спекулятивности по управлению. Перевод операций в спекулятивный режим производится сразу после построения предикатного кода. Алгоритм применения спекулятивности следующий. Со всех операций, с которых можно снять предикат, он снимается и они переводятся в спекулятивный режим. Если произведено полное переименование регистров, то достаточным условием того, что с операции можно снять предикат, не нарушив семантику программы, является то, что у операции есть только прямые потоковые последователи. Операция b является *прямым потоковым последователем* операции a, если b может использовать только значение, выработанное a.

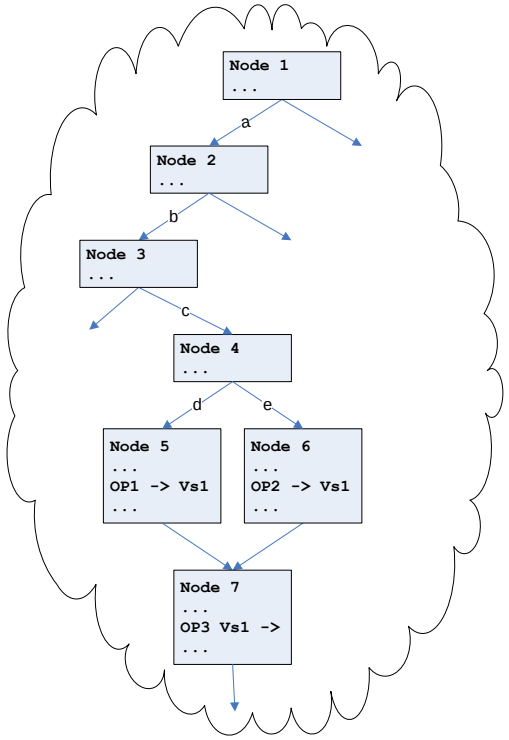

Рисунок 2. Пример графа управления.

Третий метод – использование частичных предикатов. Эффективной оптимизацией для EPIC архитектур является техника if-conversion. Она позволяет объединить несколько узлов управляющего графа в один, используя предикатное исполнение. Пусть имеется область графа управления, изображённая на рисунке 2, которую необходимо объединить в один узел. Предикатом дуги a – $P(a)$ – будем называть предикат условного перехода, соответствующий этой дуге. Полным предикатом стартого узла является тождественно истинный предикат. *Полным предикатом дуги* является «и» полного предиката узла и предиката дуги. *Полным*

предикатом узла является «или» полных предикатов всех дуг, входящих в узел. По умолчанию при слиянии узлов, каждая операция должна быть поставлена под полный предикат своего исходного узла. Например, операция `OP1` должна встать под предикат $P(a)\&P(b)\&P(c)\&P(d)$. В ряде случаев возможно поставить операцию не под полный предикат, а под так называемый частичный предикат, который шире чем полный, и, следовательно, вычисляется за меньшее количество операций. Например, на рисунке 2, операцию `OP1` достаточно поставить под $P(d)$, а операцию `OP2` под $P(e)$.

Чтобы найти частичный предикат, производится мысленный перенос операции в ближайший доминатор. Перенос возможен, если он не нарушает семантику программы, а именно такой перенос не перезапишет регистр – результат операции. То есть нигде ниже доминатора не должно быть использования этого регистра. Если перенос не возможен, то операцию необходимо поставить под «или» предикатов дуг входящих в исходный узел. Далее мысленный перенос в доминатор продолжается до тех пор, пока не достигнем стартового узла.

Описанный алгоритм построения частичных предикатов был реализован в двоичном трансляторе для микропроцессора «Эльбрус». Применение алгоритма даёт **1-2%** прироста производительности результирующего кода на широком классе задач. При этом замедление времени работы транслятора составляет 0,4%.

В **третьей главе** рассматриваются методы разрыва зависимостей в ациклических областях с построением новых операций и соответствующий им алгоритм минимизации высоты графа зависимостей.

В начале главы описываются классы разрываемых зависимостей и элементарные преобразования, разрывающие эти зависимости с построением новых операций: антизависимости, зависимости по результату, предикатные зависимости, зависимости между обращениями в память. Далее проводится обзор известных алгоритмов минимизации высоты графа зависимостей (АМВГЗ). АМВГЗ в суперблоках и АМВГЗ в процессе работы планировщика, реализованные в компиляторе IMPACT-I. АМВГЗ в гиперблоках на основе

времени раннего планирования. Недостатком этого алгоритма является то, что он не ограничивает количество созданных новых операций, то есть фактически предполагает, что в одной инструкции может исполняться бесконечно большое количество операций. На некоторых классах графов управления это может даже привести к экспоненциальному росту количества операций. Ещё один рассмотренный алгоритм – АМВГЗ с помощью решения задачи целочисленного линейного программирования. Его идея заключается в том, что задача переформулируется в терминах комбинаторной оптимизационной проблемы. Затем применяется целочисленное линейное программирование для нахождения оптимального решения. Его достоинством является то, что он позволяет найти оптимальный с точки зрения производительности набор зависимостей, которые надо разорвать. Однако у этого метода есть существенный недостаток: в нём используется NP-сложный алгоритм и, соответственно, он не может быть применён в двоичном трансляторе в силу жёстких требований к скорости работы. Таким образом, ни один из ранее разработанных алгоритмов не удовлетворяет требованиям двоичного транслятора.

Предлагаемый алгоритм минимизации высоты графа зависимостей использует все техники разрыва зависимостей, упомянутые в предыдущем абзаце. Основной аналитической структурой данных для алгоритма являются времена раннего и позднего планирования операций. Суть алгоритма состоит в том, что сначала разрываются все зависимости, которые можно разорвать, а далее восстанавливаются неэффективные разрывы, то есть те, которые заведомо не уменьшают критический путь. **Алгоритм** состоит из 4-х шагов. На первом шаге производится разрыв всех зависимостей, которые можно разорвать. На втором шаге производится разметка времён раннего планирования. Одновременно с этим производится откат неэффективно разорванных зависимостей. Время раннего планирования операции равно $\max_{v}(early(pred(v)) + delay(v))$, где v пробегает по всем входящим зависимостям, $early(pred(v))$ – время раннего планирования предшественника зависимости, $delay(v)$ – длина зависимости. Зависимость

является *определяющей*, если описанный выше max реализуется на этой зависимости. Откатываются все разрывы зависимостей, которые не являлись определяющими. На третьем шаге производится разметка времён позднего планирования. На четвёртом шаге производится откат ещё ряда зависимостей для случая, когда время позднего планирования исходной операции больше или равно времени раннего планирования новой операции, построенной для разрыва. Это условие гарантирует, что при откате зависимости длина критического пути не увеличится. После каждого отката зависимости производится корректировка времён раннего и позднего планирования, чтобы принятие решения о следующих откатах принималось на основе корректных времён. В работе доказывается корректность алгоритма коррекции времён.

Теорема. Приведённый алгоритм минимизации высоты графа зависимостей преобразует произвольный граф зависимостей в граф зависимостей с минимально возможной высотой для этих преобразований.

Доказательство этой теоремы основано на том, что после разрыва всех зависимостей высота графа зависимостей минимальна. Далее доказывается, что при откатах на втором и четвёртом шагах, высота графа зависимостей не может увеличиться.

Сложность приведённого алгоритма равна $O(m) + O(e + m) + O(e) + O(em)$, где e – количество дуг в графе зависимостей, m – количество зависимостей, которые можно разорвать. Разметка времён планирования имеет сложность $O(e)$, поэтому быстрее, чем за $O(e)$, алгоритм не может работать. Рассмотрим более детально член $O(em)$. Он описывает 4-й шаг алгоритма, а именно для каждого отката потенциально может потребоваться обойти весь граф зависимостей для коррекции. Однако в большинстве случаев коррекция быстро затухает. В работе проведено дополнительное исследование на представительной выборке задач с целью установить, какой процент дуг надо обойти на этом шаге. Результаты показали, что в среднем надо обойти всего 2,8% дуг. Это показывает, что предлагаемый алгоритм действительно обладает хорошими скоростными характеристиками в среднем.

Описанный алгоритм минимизации высоты графа зависимостей был реализован в двоичном трансляторе для микропроцессора «Эльбрус». На целочисленных и вещественных задачах из пакета SPEC CPU2000 применение алгоритма даёт **12%** и **22%** увеличения скорости работы результирующего кода соответственно. На горячих участках из Window 2000 и пользовательских приложений увеличение скорости работы результирующего кода составляет **23%**. Время работы алгоритма составляет 3.8% от общего времени трансляции.

В **четвертой главе** рассматриваются методы сокращения критических путей в циклах, получившие название *конвейеризация циклов*. В начале главы проводится обзор известных алгоритмов конвейеризации, их преимущества и недостатки. Алгоритм модульного планирования первым делом производит оценку минимального размера цикла, обозначим её x. Затем производится планирование цикла, но при этом все инструкции, равные по модулю x, отождествляются, то есть если операция спланировалась в первой инструкции, то она занимает соответствующие ресурсы в инструкция $x + 1$, $2x + 1$ и так далее. Если планирование завершилось неуспешно, то x увеличивается на 1 и повторяется планирование и так до тех пор, пока не удастся спланировать все операции цикла. Основным недостатком данного алгоритма для применения в двоичном трансляторе является то, что нельзя строить новые операции во время работы алгоритма и соответственно разрывать зависимости.

Следующим описанным ранее известным методом является метод URCR и его развития. Цикл раскручивается на два, затем операции объединяются вместе и перемешиваются, а затем ищется самый короткий интервал, который содержит все операции цикла. Алгоритм является быстрым, однако, качество результирующего кода не высокое.

Последним описанным ранее известным алгоритмом является EPS. Для конвейеризации в нём операции переносят вверх через голову цикла: одна копия операции переносится в предцикл, вторая несётся по обратной дуге и ставится перед переходом по обратной дуге. Решение о переносе принимается на основе

планирования. На каждом шаге алгоритма переносятся все операции из первой инструкции. Затем перенесённые операции перепланируются как можно выше. Преимуществом алгоритма является независимость каждого шага алгоритма, что даёт возможность проводить оптимизирующие преобразования между шагами, в частности, разрывать зависимости. Главным недостатком этого алгоритма является не оптимальность алгоритма принятия решения о том, какие операции переносить: могут переноситься как лишние операции, так и не перенестись операции, которые надо было переносить.

Предложенный в работе алгоритм конвейеризации циклов основан на идее на каждом шаге переносить через начало цикла несколько операций (как EPS). Однако принятие решения о том, какие операции переносить, основывается на совсем других принципах (а именно на разметке времён раннего и позднего планирования на расширенном графе зависимостей) и переносит только те операции, которые нужно переносить, что соответственно повышает качество результирующего кода. *Межитерационная зависимость* строится между операциями A и B, если операция B на текущей итерации зависит от операции A на какой-либо из предыдущих итераций. *Расширенным графом зависимостей* называется граф зависимостей, дополненный межитерационными зависимостями. Для описания алгоритма конвейеризации циклов потребуется ввести понятия времени раннего и позднего планирования на расширенном графе зависимостей, аналогичные временам раннего и позднего планирования на обычном графе зависимостей. Будем говорить, что *не межитерационная зависимость выдержана по времени раннего планирования*, если разница между временем раннего планирования последователя и предшественника зависимости не меньше длины зависимости. Для межитерационной зависимости считается сумма «время перехода по обратной дуге минус время раннего планирования предшественника зависимости» плюс «время раннего планирования последователя зависимости» плюс «один». Если эта сумма не меньше длины зависимости, то будем говорить, что *межитерационная зависимость выдержана по времени раннего планирования*. Аналогичные условия должны выполняться для того, чтобы

зависимость была выдержана по времени позднего планирования. *Зависимость выдержана*, если она выдержана по времени раннего планирования и выдержана по времени позднего планирования одновременно. Временами *раннего и позднего планирования на расширенном графе зависимостей* будем называть такие пары целых неотрицательных чисел для каждой операции, что все зависимости выдержаны. Дополнительным условием является то, что время раннего планирования должно быть минимально возможным, а время позднего планирования максимально возможным.

Предложенный алгоритм конвейеризации на каждом шаге берёт a – минимальное время позднего планирования среди всех операций, и переносит все операции с временем позднего планирования равным a. Перенос продолжается до тех пор, пока не достигнута оценка минимального размера цикла по ресурсам или по самой длинной рекуррентности. Каждый шаг алгоритма независим, что даёт возможность проводить дополнительные оптимизирующие преобразования между отдельными шагами. Это позволяет интегрировать в конвейеризацию техники разрыва зависимостей следующим образом. Все разрываемые зависимости помечаются, и разметка времён делается без учёта этих зависимостей. Если на очередном шаге необходимо перенести операцию, являющуюся последователем разрываемой зависимости, и, предшественник этой зависимости не попал в переносимые операции, то производится разрыв этой зависимости, и после этого перенос операции. Такая интеграция позволяет существенно повысить эффективность результирующего кода.

В работе разработан **алгоритм** разметки времён раннего и позднего планирования на расширенном графе зависимостей. На первом шаге производится разметка времён раннего планирования без учёта межитерационных зависимостей. На втором шаге подсчитывается максимальная величина *избыточной задержки*, то есть насколько длина зависимости превышает расстояние между операциями по разметке времён. Если величина избыточной задержки больше нуля, то переход по обратной дуге отодвигается на одну инструкцию (к времени раннего и позднего планирования прибавляется единица),

производится коррекция времён позднего планирования и опять вычисляется избыточная задержка. Это повторяется до тех пор, пока избыточная задержка положительна.

Корректность разработанного алгоритма разметки времён планирования доказывает следующая теорема.

Теорема. Приведённый алгоритм разметки времён планирования на расширенном графе зависимостей производит корректную разметку времён планирования.

Под оптимальностью алгоритма разметки времён раннего и позднего планирования на расширенном графе зависимостей понимается то, что время перехода по обратной дуге (полученное в результате работы алгоритма) является минимально возможным для того, чтобы существовала корректная разметка времён раннего и позднего планирования на расширенном графе зависимостей. Оптимальность разработанного алгоритма доказывает следующая теорема.

Теорема. Приведённый алгоритм разметки времён раннего и позднего планирования на расширенном графе зависимостей производит оптимальную разметку времён.

Доказательство этой теоремы построено следующим образом. Показано, что если в результате работы алгоритма не удалось построить корректную разметку для заданного значения времени перехода по обратной дуге, то такой разметки не существует. Поскольку в алгоритме время перехода на каждом шаге увеличивается на единицу, то алгоритм построит корректную разметку при минимальном значении времени перехода.

Теорема. Сложность приведённого алгоритма разметки времён раннего и позднего планирования на расширенном графе зависимостей составляет $O(e)$, где e количество зависимостей.

Доказательство основано на оценке каждого шага алгоритма, приведённой в утверждениях 1-3. Сложность каждого шага не превышает $O(e)$, а следовательно и сложность всего алгоритма не превышает $O(e)$.

Теорема. Сложность приведённого алгоритма конвейеризации циклов

составляет $O(e)$, где e количество зависимостей.

Идея доказательства в следующем. Алгоритм конвейеризации производит M шагов (переносов операций). Сложность каждого шага определяется сложностью разметки времён раннего и позднего планирования, которая составляет $O(e)$. Доказывается, что M ограничена константой, не зависящей ни от количества операций, ни от количества зависимостей.

Также в четвёртой главе описывается, как можно произвести оценку минимального размера цикла, который требуется для остановки алгоритма конвейеризации. Первая оценка – это ограничение по ресурсам. В каждой инструкции может быть исполнено только ограниченное количество операций. Например, если в одной инструкции может быть исполнено не более шести операций, то цикл из 14 операций не может быть исполнен быстрее, чем за три инструкции, так как за две инструкции можно исполнить только 12 операций. Вторая оценка – это ограничение по длине рекуррентности. В расширенном графе зависимостей возможны циклы, называемые рекуррентностями. Цикл не может быть исполнен быстрее, чем самая длинная рекуррентность. В работе приводится приближённый **алгоритм** поиска длины максимальной рекуррентности и его сравнение с точным алгоритмом. Применение точного алгоритма позволяет повысить качество результирующего кода в среднем на 0,1-0,2%, но при этом приближенный алгоритм в 100 раз быстрее точного алгоритма.

Также в этой главе описана аппаратно-программная схема реализации поддержки точного восстановления контекста в двоичном трансляторе для архитектуры «Эльбрус». И как эта схема была адаптирована для взаимодействия с аппаратным механизмом вращающихся регистров.

Описанный алгоритм конвейеризации циклов был реализован в двоичном трансляторе для микропроцессора «Эльбрус». На целочисленных и вещественных задачах из пакета SPEC CPU2000 применение алгоритма даёт **5%** и **28%** увеличения скорости работы результирующего кода соответственно. На горячих участках из Window 2000 и из пользовательских приложениях увеличение

скорости работы результирующего кода составляет **17%**. Время работы алгоритма составило 3.5% и 10% от общего времени трансляции на целочисленных и вещественных задачах соответственно.

В **заключении** приводятся основные результаты диссертационной работы и приводятся направления дальнейших исследований.

Основные результаты диссертационной работы:

1. Разработан и реализован алгоритм построения частичных предикатов. Использование данного алгоритма позволило повысить качество результирующего кода на **1-2%**, при этом замедление времени трансляции составляет **0,4%**.
2. Разработаны и реализованы алгоритмы переименования регистров и применения спекулятивности по управлению.
3. Разработан и реализован алгоритм минимизации высоты графа зависимостей в ациклических областях с целью сокращения длины критических путей. Данный алгоритм обладает высокой скоростью работы, что позволяет использовать его в динамических оптимизирующих системах. Использование данного алгоритма позволило повысить качество результирующего кода на целочисленных задачах пакета SPEC CPU2000 на **12%**, а на вещественных задачах пакета SPEC CPU2000 на **22%**. Время работы алгоритма составило 3.8% от общего времени трансляции. Дано строгое описание это алгоритма, доказана его корректность и оценена сложность.
4. Разработан и реализован алгоритм разметки времён раннего и позднего планирования на расширенном графе зависимостей. Доказана его корректность и оптимальность. Произведена оценка сложности алгоритма как с теоретической, так и с практической точек зрения. Проведённое исследование показало, что предложенный алгоритм обладает высокой скоростью работы и может быть использован в динамических оптимизирующих системах.
5. На базе алгоритма разметки времён раннего и позднего планирования на расширенном графе зависимостей был разработан и реализован алгоритм конвейеризации циклов с целью сокращения длины критических путей в

циклах. Использование данного алгоритма позволило повысить качество результирующего кода на целочисленных задачах пакета SPEC CPU2000 на **5%**, а на вещественных задачах пакета SPEC CPU2000 на **28%**. Время, затрачиваемое на работу алгоритма конвейеризации циклов, составило 3.5% и 10% от общего времени трансляции на целочисленных и вещественных задачах соответственно.

6. Разработан и реализован метод интеграции различных техник разрыва зависимостей в предложенный алгоритм конвейеризации циклов. Разрыв зависимостей во время работы алгоритма конвейеризации цикла позволяет существенно повысить скорость работы результирующего кода.
7. Разработаны и реализованы эффективные (по соотношению качества работы к скорости работы) алгоритмы учёта ограничений на минимальный размер цикла: учёт ресурсов и вычисление максимальной длины рекуррентности.
8. Суммарное повышение скорости работы результирующего кода от всех методов, предложенных в работе, составило **19%** и **58%** на целочисленных и вещественных задачах из пакета SPEC CPU2000 соответственно.

В **Приложении А** описан ранее известный алгоритм конвейеризации циклов Perfect Pipelining. В **Приложении Б** описан ранее известный алгоритм конвейеризации циклов с использованием сетей Петри.

Список работ автора, опубликованных по теме диссертации

[1] Vadim Gimpelson, Anatoly Konuhov, “Running Intel on ARM Servers: Tips and Tricks of Optimizations”, ARMTechCon 2013, Santa Clara CA, 29 Oct – 31 Oct 2013. 9 pp.

[2] Vadim Gimpelson, Anatoly Konuhov, “x86 to ARM Binary Translator”. ARMTechCon 2012, Santa Clara CA, 30 Oct – 01 Nov 2012. 11 pp.

[3] Воронов Н.В., Гимпельсон В.Д., Маслов М.В., Рыбаков А.А., Сюсюкалов Н.С. ”Система динамической двоичной трансляции x86 → «Эльбрус»”. Вопросы радиоэлектроники, серия ЭВТ, выпуск 3, 2012. С. 89-108.

[4] Гимпельсон В.Д. “Конвейеризация циклов в двоичном динамическом трансляторе”. Вопросы радиоэлектроники, выпуск 3, 2009 г. С. 67-78.

[5] Гимпельсон В.Д. “Статистический метод определения времени начала оптимизаций в динамического оптимизирующем трансляторе”.

Международная научная конференция, посвящённая 80-летию со дня рождения академика В.А. Мельникова. Сборник докладов, 2009 г. С. 135-137.
[6] Гимпельсон В.Д. “Сокращение длины критического пути циклических и ациклических участков в динамическом двоичном оптимизирующем трансляторе для архитектуры “Эльбрус”. Научные труды XXXIV Международной молодёжной научной конференции “Гагаринские чтения”, Москва, МАТИ, 2008 г. 1 сс.
[7] Загребин А.А., Гимпельсон В.Д. “Проблема восстановления профильной информации по неполным исходным данным в динамическом двоичном компиляторе”. Информационные технологии, Приложение, №11, 2008. С. 21-26.
[8] Гимпельсон В.Д. “Оптимизация циклов методом наложения итераций в динамическом трансляторе для архитектуры “Эльбрус”. Научные труды XXXIII Международной молодёжной научной конференции “Гагаринские чтения”, Москва, МАТИ, 2007 г. 1 сс.
[9] Гимпельсон В.Д. “Сокращение длины критического пути в циклах в оптимизирующем динамическом двоичном трансляторе”. Сборник тезисов XXIII научно-технической конференции войсковой части 03425. Москва, в/ч 03425, 2007 г. 2 сс.
[10] Волконский В.Ю., Гимпельсон В.Д. “Методы определения порогов активизации динамического оптимизирующего транслятора”. Информационные технологии, № 4, 2007 г. С. 32-41.
[11] Гимпельсон В.Д. “Статистически оптимальное время начала оптимизаций в динамическом двоично-оптимизирующем комплексе”. Высокопроизводительные вычислительные системы и микропроцессоры: сборник трудов ИМВС РАН, Выпуск № 9, 2006 г. С. 38-48.
[12] Волконский В.Ю., Гимпельсон В.Д., Масленников Д.М. “Быстрый алгоритм минимизации высоты графа зависимостей”, Информационные технологии и вычислительные системы, №3, 2004 г. С. 102-116.
[13] Масленников Д.М., Василец П.С., Гимпельсон В.Д., Матвеев П.Г., Муслинов Р.Г. “Программно-аппаратный метод обеспечения точного состояния контекста при прерываниях в двоично-оптимизированном коде”. Сборник тезисов XXI научно-технической конференции войсковой части 03425. Москва, в/ч 03425, 2003 г. 1 сс.

Свидетельства о государственной регистрации программы для ЭВМ

[1] Гимпельсон В.Д., Маслов М.В., Рыбаков А.А., Воронов Н.В., Садовников О.А., Айрапетян Р.Б., Савченко Р.А., Крылов С.М., Анисимов А.Б., Фомин А.А. «Eltechs ExaGear». Свидетельство о государственной регистрации программы для ЭВМ №2014611961 от 14.02.2014.