На правах рукописи
УДК 004.4'416

Лисицын Сергей Алексеевич

Исследование и оптимизация применения трасс исполнения приложения для статической бинарной трансляции под RISC архитектуры

Специальность: 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ

диссертации на соискание учёной степени кандидата технических наук

Москва — 2022

Работа прошла апробацию в федеральном государственном автономном образовательном учреждении высшего образования «Московский физико-технический институт (национальный исследовательский университет)»

Научный руководитель: **д.т.н., профессор**
Плоткин Арнольд Леонидович

Ведущая организация: **публичное акционерное общество «Институт электронных управляющих машин им. И.С. Брука»**

Защита состоится **30 июня 2022 г. в 11 ч. 00 мин.** на заседании диссертационного совета **ФРКТ.05.13.11.003**, созданного на базе федерального государственного автономного образовательного учреждения высшего образования «Московский физико-технический технический институт (национальный исследовательский университет)» (МФТИ, Физтех)

по адресу: 141701, Московская область, г. Долгопрудный, Институтский переулок, д. 9.

С диссертацией можно ознакомиться в библиотеке МФТИ, Физтех и на сайте организации https://mipt.ru.

Автореферат разослан 29 апреля 2022 года.

Ученый секретарь
диссертационного совета **Сахно Сергей Владимирович**

Общая характеристика работы

Актуальность темы. В настоящее время задача улучшения производительности вычислительных машин приобретает всё большее значение. В 2021 году две трети населения планеты ежедневно использовали мобильные вычислительные устройства, а их количество стало больше 4 миллиардов. Уменьшение времени работы на конкретных задачах или увеличение соотношения производительность/энергопотребление приносит большой выигрыш в масштабе количества устройств.

Повышения производительности приложений можно добиться двумя способами: улучшением аппаратного и программного обеспечения. Под программным обеспечением понимается не только качество написанного кода и выбранных алгоритмов, но и процесс трансляции высокоуровневого языка в бинарное представление вместе с процессом запуска бинарного файла. Для повышения производительности в трансляцию добавляются оптимизирующие стадии, такие как оптимизации компилятора, линкера и бинарные оптимизации. Последние могут быть использованы в случаях, когда в наличии есть только исполняемый бинарный файл приложения, что применимо при отсутствии исходного кода (рисунок 1).

Для повышения качества оптимизаций на вход трансляторам может поступать дополнительная информация: профиль исполнения, вероятные входные данные, дополнительная данные о микроархитектуре целевого процессора и др.

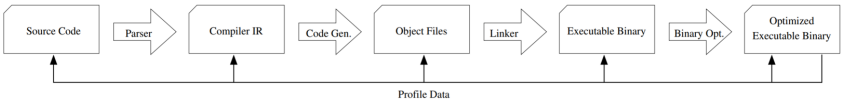


Рис. 1 — Процесс трансляции исходного кода в оптимизированный бинарный файл

Оптимизация приложений является важнейшей частью процесса создания современного программного обеспечения, так как позволяет в разы повысить производительность и энергоэффективность вычислений. Оптимизации на основе профильной информации приложения позволяют получить большой прирост в указанных выше параметрах. Для получения профиля нужно использовать специально собранную версию приложения либо обеспечивать аппаратную поддержку сбора необходимых характеристик.

Для архитектуры x86 существует специальная аппаратная поддержка, позволяющая собирать профильную информацию во время исполнения приложения - LBR (Last Branch Record). Для ARM архитектуры аналог

данной технологии существует, но не поддерживается всеми процессорами, что создаёт проблемы для оптимизаций приложений с учётом профиля исполнения. Произвести статическую инструментацию приложения не всегда является возможным, например, в случае отсутствия исходного кода или информации о символах и релокационных данных в бинарном файле. Поэтому **актуальной** задачей является разработка метода получения профильной информации для ARM архитектуры и дополнительных бинарных оптимизаций.

Целью данной работы является изучение и оптимизация существующих инструментов статической бинарной трансляции под RISC архитектуры. Основной целевой платформой является одна из наиболее распространенных на данный момент RISC архитектур – ARM. Основным классом оптимизаций являются оптимизации, использующие профильную информацию исполнения приложения.

Для достижения поставленной цели необходимо было решить следующие **задачи**:

1. Исследовать существующие статические бинарные трансляторы под RISC архитектуры.
2. Разработать универсальный метод получения профильной информации под RISC архитектуры.
3. Разработать ПО для получения профильной информации приложения по трассе его исполнения.
4. Разработать методы улучшения статического бинарного транслятора.
5. Реализовать наиболее перспективные методы улучшения бинарного оптимизатора.
6. Разработать новые оптимизации на основе внедренных методов и улучшений.
7. Протестировать разработанные методы на реальных приложениях под RISC архитектуру.

Тема и содержание диссертационной работы соответствует паспорту научной специальности 05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, в частности, пунктам:

п. 1 – Модели, методы и алгоритмы проектирования и анализа программ и программных систем, их эквивалентных преобразований, верификации и тестирования.

п. 3 – Модели, методы, алгоритмы, языки и программные инструменты для организации взаимодействия программ и программных систем.

Научная новизна:

1. В диссертационной работе был реализован новый алгоритм получения профильной информации формата BOLT на основе трасс

исполнения приложения. В диссертации показано, что для определенных классов устройств это единственный способ получения полноценного профиля приложения.
2. Была реализована полноценная поддержка бинарного оптимизатора BOLT для ARM архитектуры, необходимая для проведения тестовых замеров на целевых приложениях, в то время как исходная версия BOLT не поддерживает оптимизацию нужных приложений.
3. Был разработан специальный формат, описывающий преобразования исполняемого файла в оптимизированный.
4. Был разработан статический анализатор, верифицирующий оптимизированный исполняемый файл на основе оригинального файла и файла преобразования.
5. Был создан алгоритм мультипрофильного анализа, выделяющий профили приложения на основе трасс исполнения. На его основе реализована оптимизация копирования кода.

Практическая значимость данной работы заключается в использовании разработанных методов и алгоритмов в бинарном оптимизаторе BOLT и его инфраструктуре и успешном **внедрении** компанией ООО «Техкомпания Хуавэй» для оптимизации приложений. Реализованные оптимизации, верификации и методы получения профильной информации позволили получить прирост производительности на целевых приложениях компании.

Результаты данной работы также **внедрены** в кафедральный курс «Современные методы разработки компиляторов» кафедры микропроцессорных технологий в интеллектуальных системах управления МФТИ.

Теоретическая значимость диссертационной работы заключается в разработке новых методов использования трасс исполнения приложения, как для получения профильной информации, так и для использования в анализе и оптимизациях.

Результаты, выносимые на защиту:

1. Метод получения профильной информации для бинарного оптимизатора BOLT из трасс исполнения приложений.
2. Алгоритм оптимизации длинных переходов при работе бинарного оптимизатора BOLT.
3. Алгоритм верификации оптимизированных бинарным оптимизатором BOLT файлов.
4. Алгоритм мультипрофильного анализа трасс исполнения приложений.
5. Алгоритм дублирования кода на основе мультипрофильного анализа трасс исполнения приложений.

Достоверность. Обоснованность и достоверность результатов и выводов диссертации подтверждается подробным описанием проведенных экспериментов и полученных данных.

Апробация работы. Результаты диссертационной работы докладывались на следующих научно-технических конференциях:

1. 63-й всероссийской научной конференции московского физико–технического института (государственного университета, Москва, ноябрь 2020 г.
2. Международном конгрессе «Современные проблемы компьютерных и информационных наук», Москва, ноябрь 2021 г.
3. 64-й всероссийской научной конференции московского физико–технического института (государственного университета, Москва, ноябрь 2021 г.

Личный вклад. Основные результаты диссертационного исследования получены лично автором. Постановка задач и анализ полученных результатов осуществлялся непосредственно автором. В совместных работах вклад автора в результаты исследования являлся определяющим. Автор лично реализовывал и интегрировал разработанные решения в бинарный оптимизатор BOLT.

Публикации. Основные результаты по теме диссертации изложены в 5 печатных изданиях, 2 из которых изданы в журналах, рекомендованных ВАК, 3 — в тезисах докладов.

Содержание работы

Введение содержит обоснование актуальности исследований, проводимых в рамках данной диссертационной работы, обзор научной литературы по изучаемой проблеме, а также формулируется цель, ставятся задачи, излагается научная новизна и практическая значимость представляемой работы.

Первая глава посвящена обзору существующих технологий бинарных трансляторов, используемых для генерации оптимизированных исполняемых файлов приложений. В начале главы приводится принятая классификация современных бинарных трансляторов. Затем рассматриваются общие принципы работы существующих технологий бинарной оптимизации. Далее описывается схема работы бинарных оптимизаторов, использующих профильную информацию исполнения приложения.

Приводится описание работы бинарного оптимизатора BOLT (Binary Optmization and Layout Tool) (рисунки 2 и 4). Обсуждаются вопросы получения профильной информации для его работы на x86 архитектуре с использованием аппаратной поддержки профилирования – LBR (Last Branch Record) (рисунок 3). Также рассматривается альтернативный вариант сбора профильной информации без аппаратной поддержки сбора

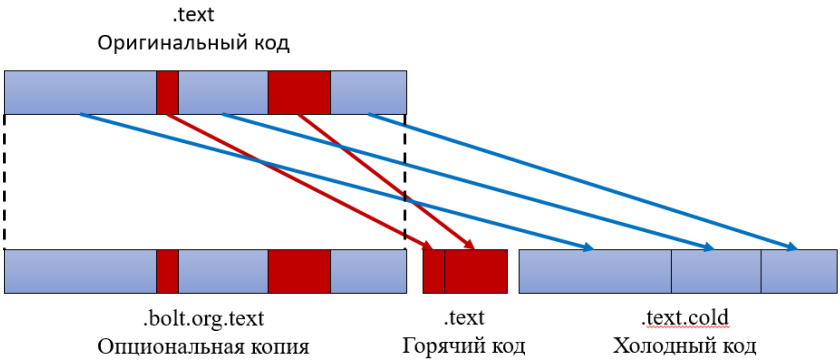


Рис. 2 — Алгоритм работы бинарного оптимизатора BOLT

статистики переходов – режим работы no_LBR_mode. В нём собирается только статистика исполняемых адресов в бинарном файле.

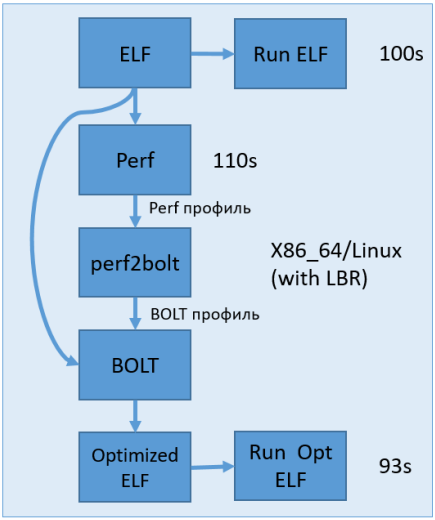


Рис. 3 — Схема оптимизации бинарного файла на x86 архитектуре

Кроме того, подробно рассматриваются вопросы формата профилей с информацией о взятых переходах и без неё. Разбирается пример оптимизации простого приложения с условным переходом в цикле. Приводится его граф потока управления и варианты раскладки его бинарного кода в памяти.

Рис. 4 — Последовательность работы бинарного оптимизатора BOLT

В конце первой главы рассматривается вопрос поддержки архитектуры ARM в бинарном оптимизаторе BOLT, варианты получения профильной информации на ней и возможности оптимизаций, приведены примеры профилей в формате BOLT.

Вторая глава посвящена разработанному универсальному методу получения профильной информации приложения из трасс исполнения.

В **разделе 2.1** описывается метод получения трасс исполнения приложений на основе динамической бинарной инструментации (DBI, Dynamic Binary Instrumentation) и предлагается схема оптимизации бинарного файла на ARM архитектуре (рисунок 5).

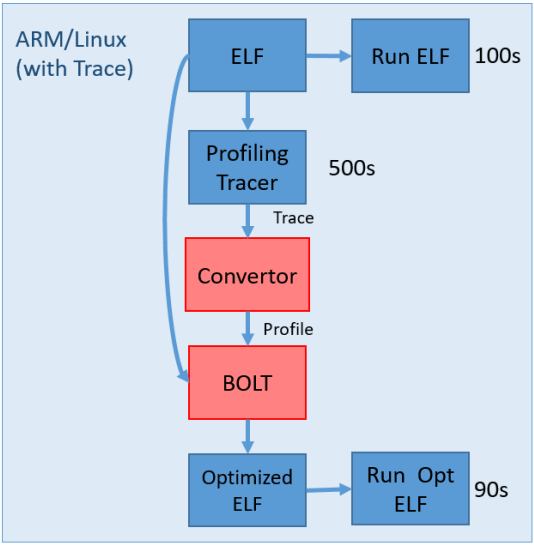


Рис. 5 — Предлагаемая схема оптимизации бинарного файла на ARM архитектуре

С помощью DBI фреймворка производится запись трассы исполнения приложения. Записывается последовательность инструкций, поступающих на процессор. После исполнения последовательность анализируется конвертером для генерации профиля исполнения, который записывается в

формате BOLT. Такое решение приводит к замедлению работы оптимизируемого приложения в несколько раз, но создаёт трассу, которую можно анализировать без привязки к конкретной микроархитектуре.

В **разделе 2.2** изложен алгоритм работы модели предсказателя переходов по трассе исполнения приложения, необходимый для получения информации о промахах предсказателя для профиля (рисунок 6).

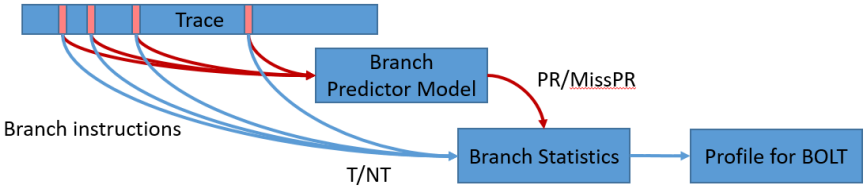


Рис. 6 — Схема генерации профиля из трассы исполнения

Для получения информации о переходах для профиля конвертер проходит по всей трассе и находит инструкции, меняющие поток управления: B, BR, BL, BLR, B.cond, TBZ, TBNZ, CBZ, CBNZ, RET. После этого информация поступает в модель предсказателя переходов (Branch Predictor Model). Данная модель выбирается в зависимости от конкретной микроархитектуры процессора, для которого оптимизируется приложение. В итоге собирается информация для получения количества не предсказанных и количества взятых переходов.

В **разделе 2.3** приводится сравнение предлагаемого решения с уже существующим подходом работы бинарного оптимизатора BOLT. Сопоставляются профили, полученные на ARM и x86 архитектурах. Метод сбора профильной информации с использованием трассы исполнения был проверен на приложениях, собранных под архитектуру ARM. Для аналогичных программ под x86 собранные с помощью LBR профили коррелируют с полученными, что позволяет использовать динамическую бинарную инструментацию с последующим анализом переходов для использования с оптимизатором BOLT.

Помимо этого, сравниваются подходы получения профильной информации с помощью аппаратной поддержки LBR и получение из трасс исполнения приложения. Во-первых, трасса исполнения зависит только от архитектуры, но не микроахитетктуры устройства, на котором происходит динамическая бинарная инструментация, что означает возможность повторной оптимизации приложения под другую микроахитектуру с другой моделью предсказателя переходов. Во-вторых, собранный профиль покрывает всё исполнение оптимизируемого приложения, а не часть исполнения

как в случае с LBR. Таким образом, полученная информация более корректна, так как избавляет профиль от искажений, связанных с ожиданием готовности периферии процессора, например, обращение в память.

В **разделе 2.4** описываются используемые синтетические тесты для проверки корректности работы предлагаемого метода получения профильной информации.

Искусственно были созданы примеры, когда исполняемый код перемешан с не исполняемым холодным кодом, что приводит к неэффективному использованию инструкционного кэша. Для генерации бинарного файла большого размера ($>$ 10 МБ) применялись рекурсивный вызов шаблонных функций в языке программирования C++. С их помощью компилятор создавал множественные копии функций с разными шаблонными константами, тем самым увеличивая размер кода до нужного значения. Исполняемый код равномерно распределяется по секции, чередуясь с холодными функциями.

В **разделе 2.5** приводятся результаты запусков, оптимизированных на основе профильной информации, которая была собрана из трасс исполнения синтетических тестов.

```
erf stat -e L1-icache-load-misses,L1-icache-loads,iTLB-load-misses,iTLB-loads,cycles,instructions
real 4.544466
user 4.532000
sys 0.008000

 Performance counter stats for 'time ./main_ARM64_1k_10_clang.bolt.hack 0 1 1000000':

         38828731      L1-icache-load-misses                                       (16.74%)
      10344492618      L1-icache-loads                                             (16.77%)
              6        iTLB-load-misses          #    0.00% of all iTLB cache hits   (16.74%)
        8292220399      iTLB-loads                                                  (16.69%)
      11658075504      cycles                                                      (16.71%)
      32435371924      instructions              #    2.78  insns per cycle          (16.70%)

      4.558532811 seconds time elapsed

erf stat -e L1-icache-load-misses,L1-icache-loads,iTLB-load-misses,iTLB-loads,cycles,instructions
real 6.375960
user 6.348000
sys 0.008000

 Performance counter stats for 'time ./main_ARM64_1k_10_clang 0 1 1000000':

        567516166      L1-icache-load-misses                                       (16.73%)
      10984131626      L1-icache-loads                                             (16.72%)
        124961644      iTLB-load-misses          #    1.38% of all iTLB cache hits   (16.66%)
       9070438457      iTLB-loads                                                  (16.65%)
      16353327498      cycles                                                      (16.69%)
      32170724391      instructions              #    1.97  insns per cycle          (16.78%)

      6.389770312 seconds time elapsed
```

Рис. 7 — Сравнение результатов запусков оригинального (снизу) и оптимизированного (сверху) бинарных файлов синтетического теста

Полученные статистические данные (рисунок 7) показывают понижение практически до нуля количество промахов по iTLB, увеличение IPC

(instruction per clock) на 41% и уменьшение времени работы на 29%. Для тестирования была выбрана микроархитектура ARM Cortex A76.

Третья глава посвящена описанию улучшений в бинарном оптимизаторе BOLT под ARM архитектуру, их верификации и тестированию.

В **разделе 3.1** рассматривается проблема длинных переходов в бинарной оптимизации. Описывается метод оптимизации трамплинов, позволяющий уменьшить количество добавляемых для трамплина инструкций и не использовать дополнительный регистр.

Перекомпоновка кода проще реализуется на CISC архитектуре за счёт возможности добавления длинных прыжков одной инструкцией. На RISC архитектурах количество битов смещения в инструкциях переходов меньше, что приводит к ограничению диапазона возможного перемещения. Также инструкция загрузки в регистр адреса, зависящего от счетчика инструкций, на ARM архитектуре ограничена 20 битами, выделенными на смещение, и накладывает возможный диапазон перемещения $\pm$ 1 мегабайт. Все инструкции, которые зависят от своего адреса (переходы, загрузка адреса, загрузка по регистру/смещению), после оптимизации необходимо проверить на возможность записи нового смещения в отведенные для этого биты.

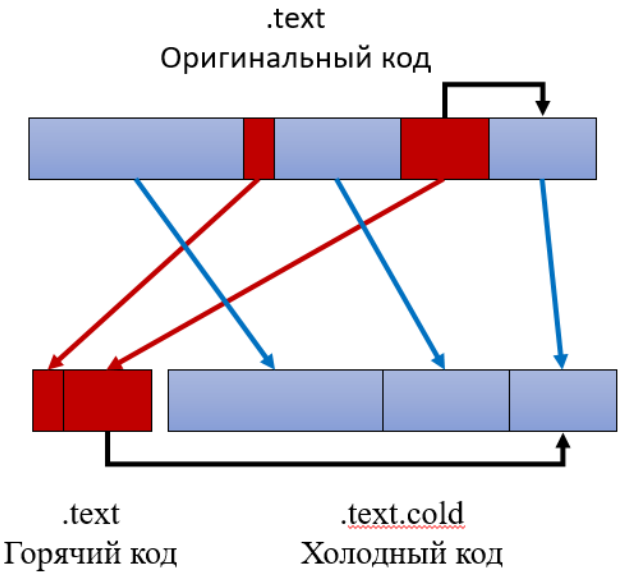


Рис. 8 — Пример перекомпоновки кода

Таким образом, при перемещении горячего кода, ссылающегося на холодный участок (черная стрелка на рисунке 8), происходит переполнение значения смещения, и оптимизация становится некорректной.

Для решения данной проблемы необходимо добавить трамплин – дополнительный код, позволяющий генерировать произвольный адрес и произвести переход либо загрузку по данному значению. Подобное решение будет увеличивать размер кода, что приведёт к понижению производительности, но диапазон перемещаемого кода будет увеличен.

Case	Offset	Add instructions	Add register use
No trampoline	± 1 MB	0	0
Small trampoline	± 128 MB	1 (B)	0
Full trampoline	± 4 GB	3 (ADRP, ADD, BR)	1

Рис. 9 — Сравнение трамплинов

Для условных переходов можно написать маленькие трамплины размером в одну инструкцию с помощью добавления одного безусловного перехода в случае, если смещение занимает больше 19, но меньше 26 бит (рисунок 9).

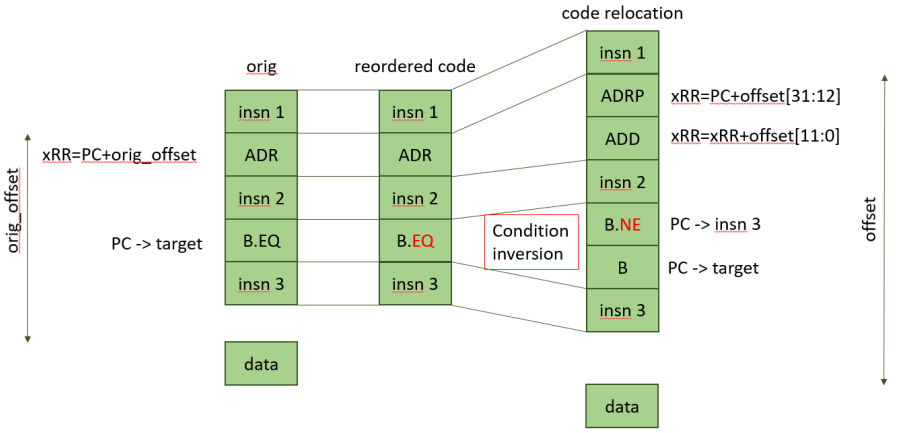


Рис. 10 — Схема добавления трамплинов для ADR (сверху) и Bcc (снизу) инструкций

В данном варианте необходимо инвертировать условие инструкции и метку для прыжка поставить через одну инструкцию прямого перехода, вставленную оптимизатором непосредственно после условного перехода. Получаем увеличение возможного региона перемещения в 128 раз за счёт отличий в кодировках Bcc и B, а увеличение кода произошло всего на 1 инструкцию (рисунок 10).

В **разделе 3.2** описывается проблема бинарной оптимизации исполняемых файлов под ARM архитектуру, в которых используются таблицы переходов.

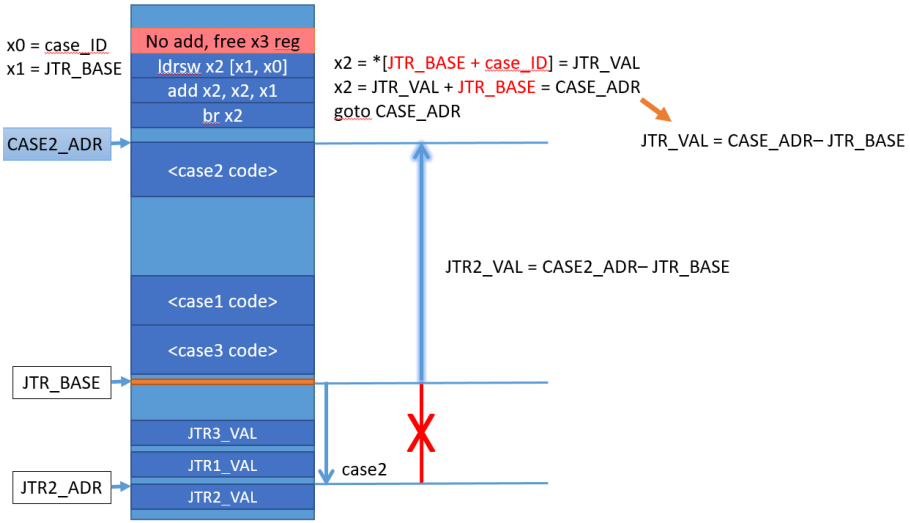


Рис. 11 — Оптимизированный вариант ассемблерного кода для switch-case случая

При перемещении кода конкретного варианта конструкции switch-case, необходимо модифицировать запись в таблице, используемую в косвенном переходе. Оптимизатор узнает адрес записи из релокационной информации, которая для некоторых случаев кода ARM архитектуры не будет корректной (рисунок 11). Это происходит по причине несоответствия адреса, от которого вычисляется целевой адрес для прыжка при исполнении и генерации релокационной информации.

Данная проблема решается переиспользованием оригинального участка кода с данным косвенным переходом. В этом случае необходимо сохранить копию оригинальной секции .bolt.org.text.

В **разделе 3.3** предлагается метод верификации оптимизированного бинарного файла.

Для решения данной задачи был разработан специальный формат, описывающий преобразования исполняемого файла в оптимизированный, так называемый remap-файл. В нём записывается информация о всех совершенных перестановках кода и дополнительная информация, обозначающая инструкции, подвергнутые преобразованиям в процессе перекомпоновки (PC-relative инструкции).

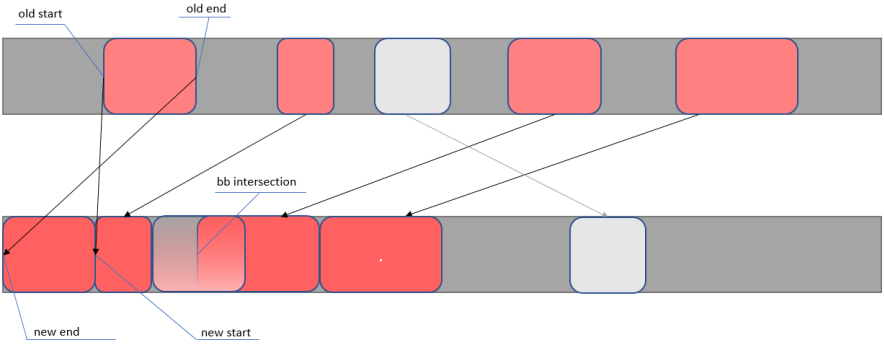


Рис. 12 — Схема верификации бинарного файла

В рамках данной работы в оптимизатор BOLT был добавлен проход, генерирующий remap-файл, а также написан статический анализатор, сравнивающий оригинальный и оптимизированный исполняемые файлы по данному remap-файлу (рисунок 12). Реализованы следующие проверки:

1. Проверка корректности оптимизированных линейных участков кода.
2. Проверка корректности создания теневых точек.
3. Сравнение инструкций оптимизированных и исходный линейных участков.

В **разделе 3.4** описываются результаты тестирования синтетических тестов и общих наборов тестов после оптимизации модифицированным бинарным транслятором BOLT (рисунки 13 и 14).

Для проверки оптимизации на реальном приложении был выбран набор тестов производительности GeekBench. Среди всего набора был выбран тест с наибольшим числом iTLB и L1I промахов – Clang. При компиляции набора тестов были использован флаг «-O2», который соответствует оптимизациям при стандартной сборке приложений.

По результатам тестов был получен прирост в показателях теста на 10%, уменьшение iTLB и L1I промахов на 32% и 38% соответственно, что является показателем увеличения средней температуры кода. При этом остальные тесты из набора GeekBench либо не улучшили производительность, либо ухудшили её.

```
Single-Core
  Clang                          703          5.48 Klines/sec

Benchmark Summary
  Single-Core Score              457
    Integer Score                  703


 Performance counter stats for './gkb5 --workload 208 --single-core --no-upload':

       16975240547      instructions              #    1.32  insns per cycle          (20.59%)
       12847912630      cycles                                                        (20.59%)
          58933576      L1-icache-load-misses                                         (20.74%)
        6167593264      L1-icache-loads                                               (21.05%)
          15616258      iTLB-load-misses          #    0.29% of all iTLB cache hits   (20.46%)
        5398816554      iTLB-loads                                                    (20.14%)

       5.387183853 seconds time elapsed
```

Рис. 13 — Результаты исполнения оригинального Clang теста

```
Single-Core
  Clang                          774          6.03 Klines/sec

Benchmark Summary
  Single-Core Score              503
    Integer Score                  774


 Performance counter stats for './gkb5.bolt --workload 208 --single-core --no-upload':

       17091629207      instructions              #    1.34  insns per cycle          (21.09%)
       12789839024      cycles                                                        (20.91%)
          40128240      L1-icache-load-misses                                         (21.04%)
        6251372347      L1-icache-loads                                               (20.59%)
           9692777      iTLB-load-misses          #    0.19% of all iTLB cache hits   (20.40%)
        5210630729      iTLB-loads                                                    (20.50%)

       5.228721353 seconds time elapsed
```

Рис. 14 — Результаты исполнения оптимизированного Clang теста

В **четвертой главе** приведено описание бинарных оптимизаций на основе трасс исполнения приложения.

В **разделе 4.1** описываются проблемы оптимизаций с профильной информацией.

Для работы оптимизатора BOLT необходимы значения счетчика команд и последние взятые переходы с информацией от предсказателя переходов. Во время сбора характеристик программа исполняется по определенному сценарию: определенные входные данные, параметры окружения, контекст и т.д. Полученная профильная информация будет показывать характеристики этого выбранного сценария.

После оптимизации приложения с данным профилем запуск по данному сценарию будет показывать прирост производительности, но при проверке производительности с другими входными параметрами на данном приложении такого же прироста производительности получить не удастся (вероятно будет происходить регрессия).

Кроме того, приводится пример приложения, при оптимизации которого будет получен либо прирост, либо регрессия производительности в зависимости от выбранного сценария.

В **разделе 4.2** рассматривается дополнительная информация, которую можно извлечь из трассы исполнения и применить для последующей бинарной оптимизации приложения.

Трасса исполнения содержит в себе гораздо больше информации, чем профиль, который в конечном итоге используется для оптимизации приложения. Помимо количества сделанных переходов на каждой инструкции и изменения программного счетчика, есть информация о последовательности сделанных переходов, которую можно учитывать во время оптимизации.

Также для оптимизации можно добавить в трассу информацию про использованные адреса в инструкциях загрузки и выгрузки. Таким образом, можно достать подробную информацию для оптимизации данных в бинарном файле.

В **разделе 4.3** приводится описание мультипрофильного анализа трасс исполнения приложения. Вводятся основные необходимые термины для данного анализа: интервалы инструкций, фазы исполнения и мультипрофиль.

Трасса разбивается на одинаковые интервалы по количеству инструкций. Это необходимо для анализа исполнения в рамках одного отрезка времени – интервала инструкций, так как требуется найти похожие интервалы и объединить их в фазы.

Для большей наглядности граф исполнения отображается в виде тепловой карты (рисунок 15). По оси X – интервалы инструкций согласно времени исполнения, по оси Y – номер линейного участка кода.

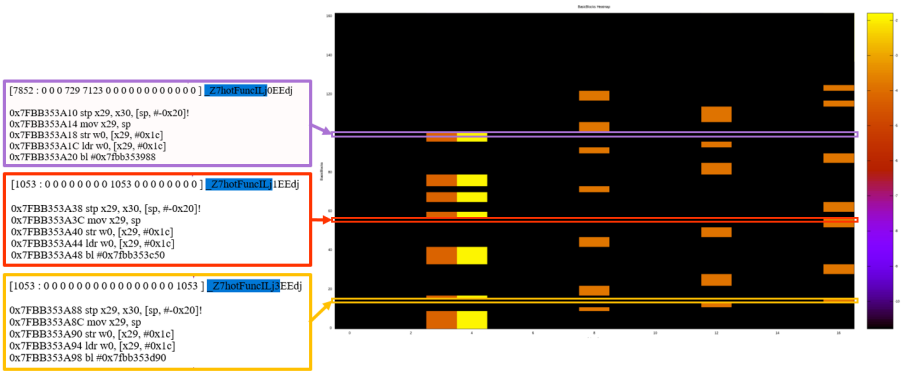
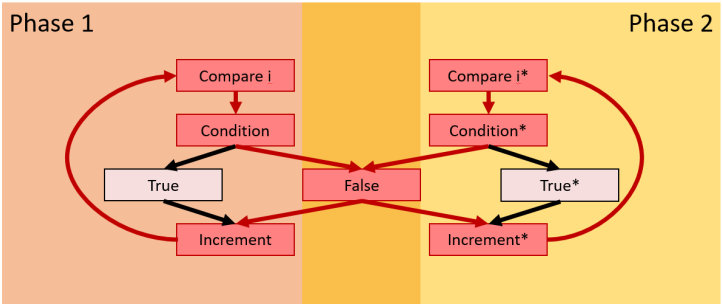


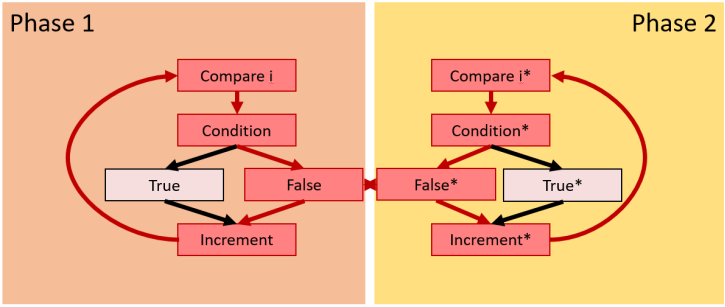
Рис. 15 — Построение тепловой карты линейных участков

На каждом интервале исполнялся определенный набор линейных участков кода, который будет характеризовать интервал инструкций. Если составить n-мерное пространство, где n – количество линейных участков кода, а координаты – количество исполненных соответствующих линейных участков на выбранном интервале, то каждому интервалу в данном пространстве будет соответствовать точка.

Также в разделе приводится алгоритм кластеризации интервалов. Каждый полученный кластер интервалов инструкций означает определенную фазу. Кластеризация проводится с помощью итеративного алгоритма объединения фаз. По переключениям между фазами строится граф исполнения. Он является аналогом графа потока управления, но на более высоком уровне, показывая переключения не между линейными участками, а между макросостояниями исполнения программы.



а)до оптимизации



б)после оптимизации

Рис. 16 — Дублирование кода на основе мультипрофиля

Для завершения анализа приложения необходимо построить соответствие между фазами и линейными участками приложения. Для этого каждый линейный участок анализируется отдельно, выбирая интервал с

его наибольшим количеством исполнений. Линейному участку кода будет соответствовать фаза, которая включает в себя этот интервал.

В **разделе 4.4** описывается бинарная оптимизация - дублирование кода на основе мультипрофиля.

Выстраивая соответствие между линейными участками кода и фазами, выбиралась фаза, к которой принадлежит интервал инструкций с наибольшим количеством исполнений данного линейного участка. Однако, может произойти ситуация, когда линейный участок исполнялся на интервалах инструкций, относящихся к разным фазам. В этом случае необходимо запоминать такие линейные участки для последующего анализа необходимости дублирования данного кода.

Используя эвристически подобранные соотношения, выбираются те линейные участки, которые относятся одновременно к нескольким фазам. После этого данные участки помечаются для дублирования бинарным оптимизатором BOLT (рисунок 16).

В **разделе 4.5** приводятся результаты запусков тестов, оптимизированных с помощью дублирования кода на основе мультипрофиля.

Для проверки анализа был использован набор тестов GeekBench. Были записаны трассы его исполнения на различных задачах. На тепловой карте (рисунок 17) выделены запуски различных задач, представляющие в данном случае сценарии запуска приложения. По верхней строке тепловой карты видно корректное выделение фаз на каждую отдельный тест из набора.

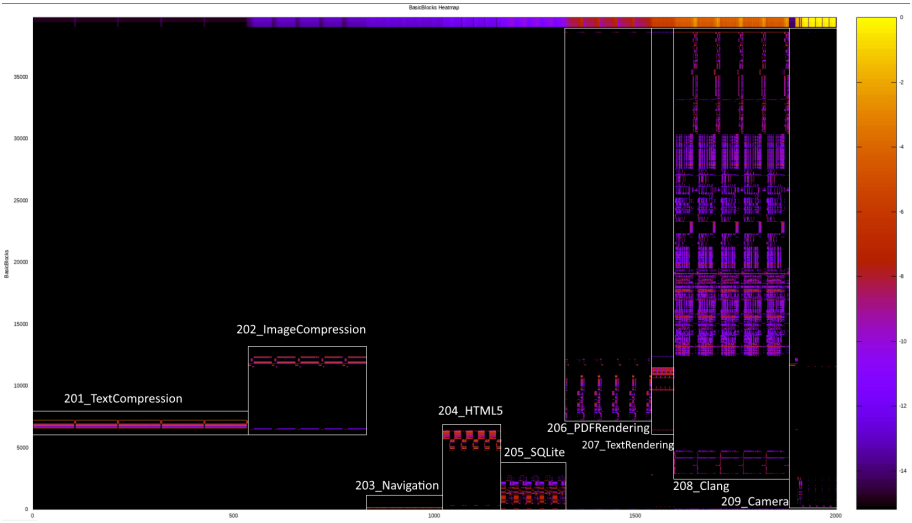


Рис. 17 — Тепловая карта набора тестов Geekbench с выделенными тестами

При использовании оптимизации дублирования кода на основе мультипрофильной информации удалось получить прирост производительности до 4% на отдельных тестах из набора, но средний прирост на всех тестах не превышает 1% (рисунок 18). Это связано с отсутствием больших пересечений в коде набора тестов. Максимальный прирост был получен на мультипрофиле с 12 фазами, что коррелирует с порядком количества тестов в наборе.

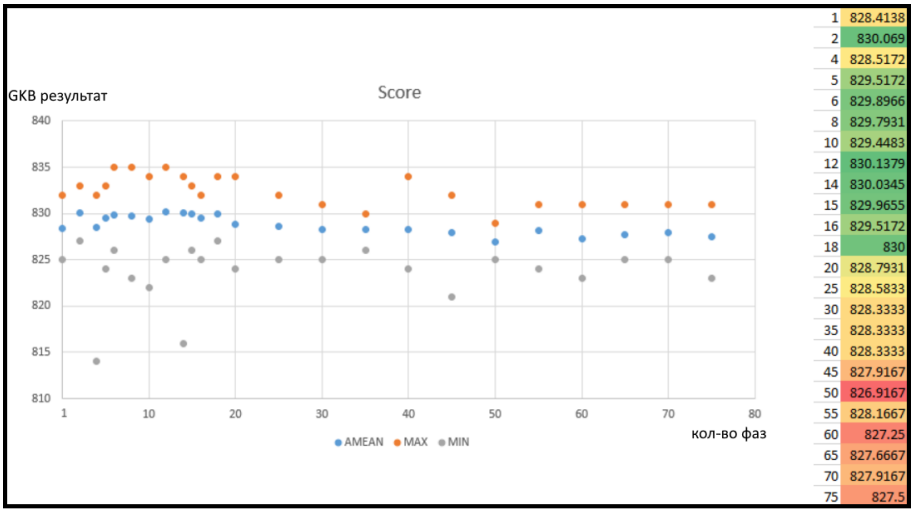


Рис. 18 — Результаты тестирования мультипрофильного дублирования кода на наборе тестов GeekBench

В **заключении** приведены основные результаты работы:

1. В диссертации был реализован новый алгоритм получения профильной информации формата BOLT на основе трасс исполнения приложения. В работе показано, что для определенных классов устройств это единственный способ получения полноценного профиля приложения.
2. Было разработано ПО, транслирующее трассу исполнения приложения для архитектуры ARM в профильную информацию.
3. Была реализована полноценная поддержка бинарного оптимизатора BOLT для ARM архитектуры, необходимая для проведения тестовых замеров на целевых приложениях. Исходная версия BOLT изначально не поддерживала оптимизацию нужных приложений.

4. Был получен прирост производительности на 10% при использовании бинарного оптимизатора BOLT для целевых приложений на ARM архитектуре.
5. Был разработан специальный универсальный формат, описывающий преобразования исполняемого файла в оптимизированный.
6. Был разработан статический анализатор, верифицирующий оптимизированный исполняемый файл на основе оригинального файла и файла преобразования.
7. Был создан алгоритм мультипрофильного анализа, выделяющий профили приложения на основе трасс исполнения. На его основе реализована оптимизация копирования кода.

Разработанные в рамках диссертационного исследования алгоритмы и методы внедрены и используются:

1. В бинарном оптимизаторе BOLT и его инфраструктуре, применяемых в ООО «Техкомпания Хуавэй» для оптимизации приложений.
2. При чтении кафедрального курса «Современные методы разработки компиляторов» кафедры микропроцессорных технологий в интеллектуальных системах управления МФТИ.

Публикации автора по теме диссертации

1. *Лисицын, С. А.* Сбор профильной информации с помощью трасс исполнения приложения для статической оптимизирующей бинарной трансляции [Текст] / С. А. Лисицын // Международный научный журнал «Современные информационные технологии и ИТ-образование». — 2021. — Т. 17, № 2. — С. 369—378. — (перечень ВАК).
2. *Лисицын, С. А.* Мультипрофильная статическая бинарная оптимизация приложения на основе трасс исполнения [Текст] / С. А. Лисицын // Международный научный журнал «Современные информационные технологии и ИТ-образование». — 2021. — Т. 17, № 3. — С. 560—579. — (перечень ВАК).
3. *Лисицын, С. А.* Изучение проблем статической двоичной трансляции под RISC архитектуры [Текст] / С. А. Лисицын // ТРУДЫ 63-й Всероссийской научной конференции МФТИ. Радиотехника и компьютерные технологии. — 2020.
4. *Лисицын, С. А.* Сбор профильной информации с помощью трасс исполнения приложения для статической оптимизирующей бинарной трансляции [Текст] / С. А. Лисицын // Программа международного конгресса «Современные проблемы компьютерных и информационных наук». — 2021.
5. *Лисицын, С. А.* Верификация статической двоичной оптимизирующей трансляции под RISC архитектуры [Текст] / С. А. Лисицын, А. А. Шурыгин // ТРУДЫ 64-й Всероссийской научной конференции МФТИ. Радиотехника и компьютерные технологии. — 2021.

Лисицын Сергей Алексеевич

Исследование и оптимизация применения трасс исполнения приложения для статической бинарной трансляции под RISC архитектуры

Автореф. дис. на соискание ученой степени канд. тех. наук

Подписано в печать 27.04.2022. Заказ №
Формат 60×90/16. Усл. печ. л. 1. Тираж 20 экз.
Типография «11-й ФОРМАТ» г. Москва, Варшавское ш., 36