

СИСТЕМА ДИНАМИЧЕСКОЙ ДВОИЧНОЙ ТРАНСЛЯЦИИ X86 → «ЭЛЬБРУС»

При появлении новых микропроцессорных архитектур возникает проблема переноса на них ранее созданного программного обеспечения. При этом исходный код того или иного приложения зачастую недоступен, и перекомпилировать его для новой архитектуры не представляется возможным. Технология динамической двоичной трансляции позволяет переносить программное обеспечение путем перевода двоичного кода из набора инструкций исходной архитектуры в набор инструкций целевой архитектуры

Это и по сей день является многообещающей технологией и притягательным для многих инженеров направлением исследований. Поэтому в курсе предмета будет весьма актуально рассмотреть данную тему на конкретном примере, а именно как

СИСТЕМУ ДИНАМИЧЕСКОЙ ДВОИЧНОЙ ТРАНСЛЯЦИИ X86 → «ЭЛЬБРУС»

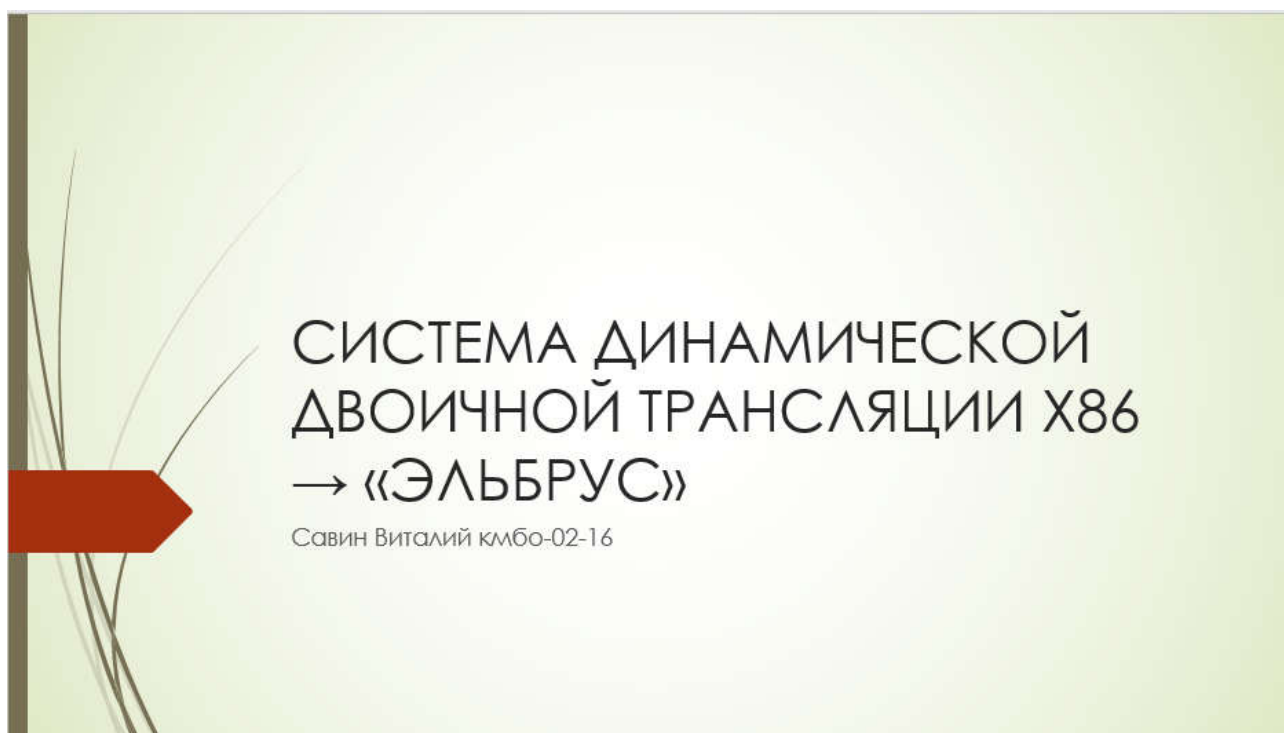


Рисунок 1. Слайд 1 - заголовок

1 Введение

1.1 Слайд 2 – Содержание

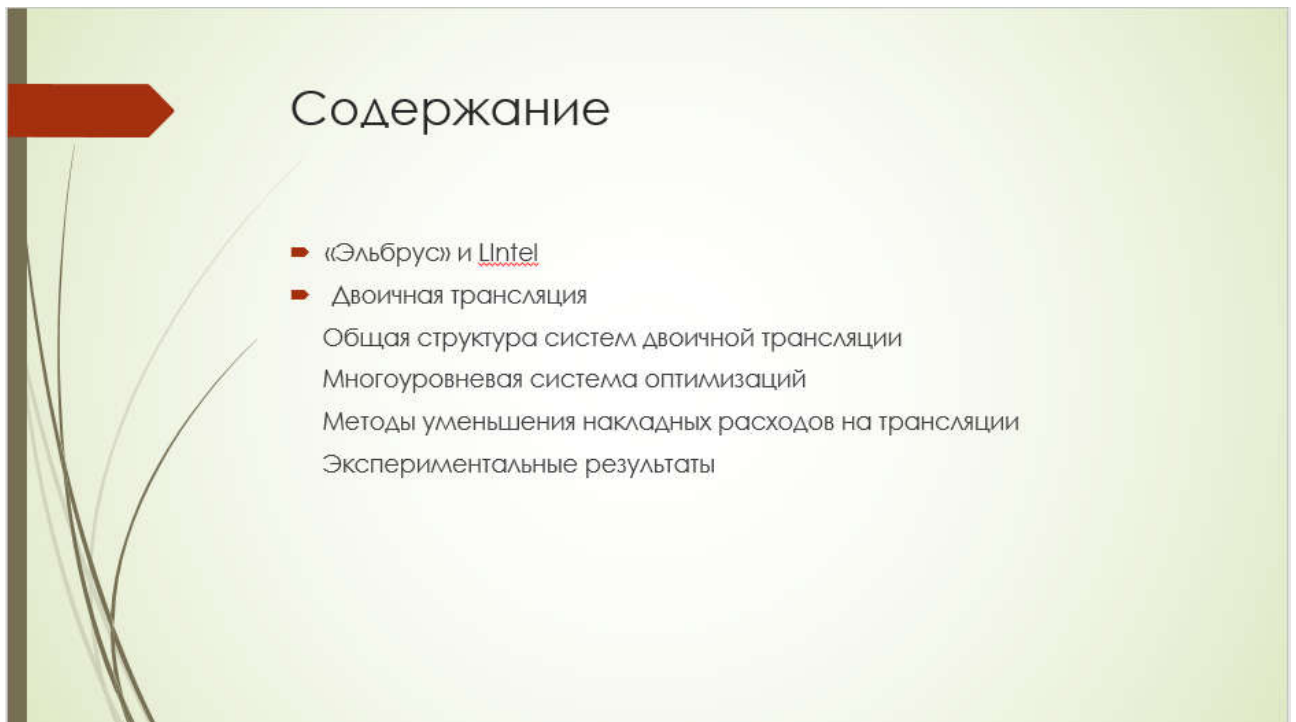


Рисунок 2. Слайд2 - содержание

В процессе моего выступления будет дано описание системы динамической трансляции двоичных кодов архитектуры x86 в архитектуру «Эльбрус». Рассматривается общая схема работы двоичного транслятора, многоуровневая система оптимизаций, технологии сокращения накладных расходов на трансляцию (долговременное хранение кодов и параллельная трансляция).

1.2 Слайд 3 – описание «Эльбрус»



Рисунок 3. Слайд3 – описание «Эльбрус»

В компании ЗАО «МЦСТ» разрабатываются микропроцессоры архитектуры «Эльбрус». Архитектура «Эльбрус» обладает высокой предельной архитектурной скоростью, которая достигается за счет ее многочисленных особенностей. Основной причиной высокой производительности архитектуры «Эльбрус» является параллелизм исполнения кода на уровне отдельных команд. Это обусловлено тем, что «Эльбрус» является VLIW-архитектурой (very large instruction word), или архитектурой с широким командным словом, что позволяет кодировать в одной VLIW команде несколько отдельных операций и выполнять их параллельно.

Для вычислительного комплекса «Эльбрус» разработана система двоичной трансляции LIntel, позволяющая исполнять приложения Intel x86 на микропроцессорах «Эльбрус». Важной составляющей LIntel является многоуровневый оптимизирующий двоичный транслятор.

3.1 Слайд 4 - Общая схема работы



Рисунок 4. Слайд 4 – Общая схема работы

Общая схема работы

*Схема работы системы двоичной трансляции $x86 \rightarrow$ «Эльбрус» изображена на рис. Исполнение $x86$ кода начинается в интерпретаторе, который собирает профильную информацию об исполненных инструкциях и в случае превышения порогового количества исполнений запускает трансляция этого $x86$ -кода в коды «Эльбруса». Транслированный код сохраняется в кэш трансляций. Во время работы оттранслированного кода ведётся статистика исполнения линейных участков $x86$ -кода, на базе которой строится профильный граф. В узле профильного графа хранятся: количество исполнений соответствующего линейного участка, статистика по переходам (счётчики дуг), информация о специфике инструкций, включенных в данный узел (наличие *fp*, *mtx*, *sse* операций и т.д.). При превышении порого-*

вого количества исполнений линейного участка x86-кода, запускается наборщик регионов, который выделяет область горячего кода для трансляции оптимизирующими компиляторами, называемую регионом.

Для осуществления переходов по x86-адресам между трансляциями используется таблица перекодировки адресов, которая хранит соответствие <x86-адрес начала линейного участка → e2k-адрес транслированного кода>. При выходе из трансляции и попытке перейти на определённый x86-адрес проверяется, нет ли в таблице перекодировки адресов соответствующей записи. В случае положительного результата, происходит переход на найденный вход в трансляцию, иначе запускается интерпретатор.

Система поддержки трансляций обеспечивает взаимодействие модуля трансляции с остальной системой: запускает наборщик регионов, выбирает уровень трансляции, размещает полученную трансляцию в кэше трансляций, регистрирует глобальные входы в трансляцию в таблице перекодировки адресов.

Работа системы двоичной трансляции может прерываться приходом прерываний (для транслятора уровня всей системы) или сигналов (для транслятора уровня приложений). Обработка этих событий происходит в *обработчике прерываний/сигналов*.

2.2 Слайд 5 – Трансляторы уровня всей системы и уровня приложений



Рисунок 5. Слайд 5 – Трансляторы уровня всей системы и уровня приложений

Трансляторы уровня всей системы и уровня приложений

Были реализованы два подхода к построению системы двоичной трансляции, различающиеся своей применимостью.

Первый подход – система полной двоичной трансляции. Здесь транслятор работает между микропроцессором и запускаемыми на нём x86-кодами. Транслируются коды BIOS, операционной системы, драйверов и прикладных программ. Вычислительный комплекс на базе микропроцессора «Эльбрус» с системой полной двоичной трансляции для пользователя неотличим от вычислительного комплекса на базе x86-микропроцессоров.

При втором подходе система двоичной трансляции является обычным Linux-приложением и работает под управлением ОС Linux. Она позволяет запускать Linux-приложения для платформы x86, которые могут работать одновременно с приложениями в кодах платформы «Эльбрус».

2.3 Слайд 6 - Многоуровневая система оптимизаций - Мотивация



Рисунок 6. Мотивация

Мотивация

В системе двоичной трансляции $x86 \rightarrow$ «Эльбрус» реализована многоуровневая система оптимизаций. В состав двоичного транслятора включены интерпретатор и три транслятора с различными уровнями оптимизации. Каждый следующий уровень генерирует более эффективный результирующий код. Итоговое количество исполнений участка кода невозможно определить заранее, т.к. это число зависит от логики работы исполняемой программы и исходных данных. Поэтому сначала код транслируется без оптимизаций, а затем, исходя из профильной информации, код перетранслируется более высокими уровнями оптимизаций. В результате использования четырехуровневой схемы достигается заметно более высокая скорость выполнения приложения

2.4 Слайд 7 - Интерпретатор



Рисунок 7. Слайд 7 - Интерпретатор

Интерпретатор

Интерпретатор, по сути, не будучи транслятором, является базовым уровнем многоуровневой системы двоичной трансляции. Он последовательно декодирует инструкции x86 и на основе извлечённой информации вызывает функцию, выполняющую над контекстом (регистры, память и т.д.) такое же преобразование, что и исходная инструкция, затем исполняется следующая инструкция. Состояние контекста перед выполнением каждой x86 инструкции является таким же, как и в исходной архитектуре, причем как внутри функций изменения контекста, так и между исполнением инструкций, интерпретатор проявляет все необходимые исключительные ситуации.

Интерпретатор не генерирует двоичной код целевой архитектуры, поэтому, декодирование и исполнение инструкций всегда начнётся заново. Помимо эмуляции x86-инструкций интерпретатор набирает трассу – по-

следовательность исполненных им линейных участков, идущих в порядке возрастания адресов. После того как количество исполнений трассы превышает некоторый порог, запускается процедура трансляции данной трассы не оптимизирующим двоичным транслятором – *шаблонным транслятором*.

Рис иллюстрирует суммарное время, затраченное на исполнение инструкций, выполнившихся меньше заданного количества раз, для приложения Acrobat Reader в двух случаях: исполнение только интерпретатором и исполнение, когда коды сразу транслируются оптимизирующим компилятором. Как видно, накладные расходы на оптимизацию очень велики. Несмотря на то, что оптимизированный код выполняется во много раз быстрее, итоговое время в четыре раза больше времени исполнения в интерпретаторе.

2.5 Слайд 8 - Шаблонный транслятор

Шаблонный транслятор

Двоичный транслятор Intel состоит из трех уровней. Первый уровень представляет шаблонный транслятор. Далее следует быстрый оптимизирующий компилятор (компилятор уровня O0) . И, оптимизирующий компилятор уровня O1

Таблица 1. Характеристики уровней двоичной трансляции.

	Время трансляции	Качество кода
Интерпретатор	100	-
Шаблонный транслятор	1200	1/4
Компилятор O0	15000	2/3
Компилятор O1	500000	1

Сравнительные данные по времени компиляции исходного кода x86 различными уровнями транслятора, а также сравнение результирующего кода с кодом, полученным с помощью компилятора O1

Рисунок 8. Слайд 8 – Шаблонный транслятор

Шаблонный транслятор

Шаблонный транслятор принимает на вход трассу, полученную ин-

терпретатором, и транслирует каждую инструкцию трассы в двоичный код целевой архитектуры. Результирующий код помещается в кэш трансляций

Шаблонный транслятор обрабатывает каждый линейный участок отдельно, несмотря на то, что на вход ему подаётся трасса. Генерация кода участка происходит следующим образом: сначала генерируется пролог, затем отдельно транслируется каждая инструкция x86, и заканчивается код эпилогом.

в шаблонном трансляторе производится подсчёт количества исполнений линейных участков, и в случае превышения им некоторого порога управление передается на *наборщика регионов*. Задачей наборщика регионов является набор региона – подграфа профильного графа, который будет передан для трансляции быстрому региональному компилятору.

В таблице приведены сравнительные данные по времени компиляции исходного кода x86 различными уровнями транслятора, а также сравнение результирующего кода с кодом, полученным с помощью компилятора O1. В колонке «Время трансляции» приведено количество тактов «Эльбрус», затрачиваемое на трансляцию в пересчете на одну исходную инструкцию x86. Для интерпретатора это означает количество тактов, затрачиваемое на эмуляцию одной инструкции. В колонке «Качество кода» приведено отношение средней скорости работы результирующего кода, полученного с помощью определенного уровня транслятора, к средней скорости работы результирующего кода, полученного с помощью компилятора O1. Для интерпретатора эта характеристика лишена смысла, так как для него отсутствует понятие результирующего кода

3.6 Слайд 9 - Быстрый региональный компилятор

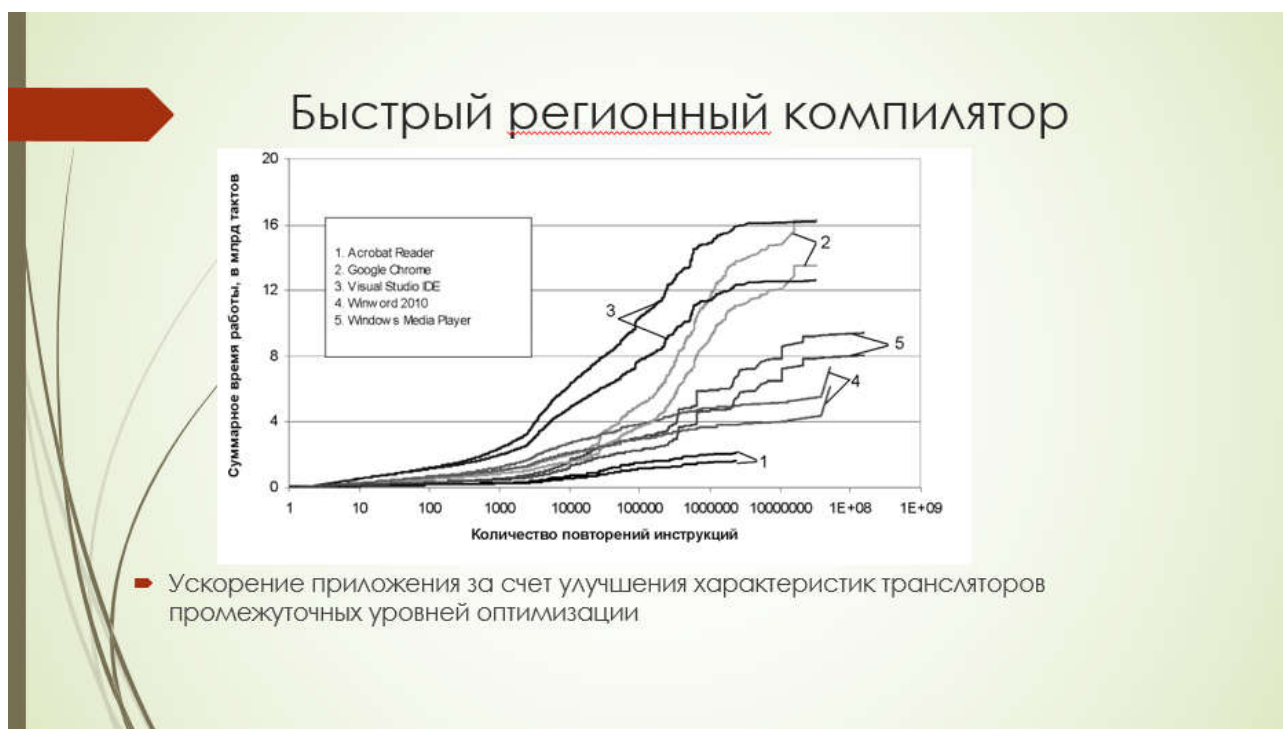


Рисунок 9. Слайд 9 – быстрый региональный компилятор

Быстрый региональный компилятор

Функциональность

Быстрый компилятор предназначен для оптимизации недостаточно горячих регионов. Он включает в себя урезанный набор базовых оптимизаций, которые позволяют достичь существенного прироста производительности результирующего кода при наименьших затратах по времени компиляции.

Процесс компиляции региона быстрым компилятором условно можно разделить на три этапа. На первом из них, генерации семантики, осуществляется перевод кода x86 в промежуточное представление компилятора. Вторым этапом является последовательное применение к сгенерированному промежуточному представлению основных оптимизаций. На третьем этапе, планировании кода, осуществляется распределение регистров, планирование широких команд и генерация результирующего кода.

Оптимизации, обеспечивающие основной прирост производительности

сти

Оптимизации описаны в той последовательности, в которой они применяются.

удаления избыточных вычислений (peerhole), Слияние узлов (if-conversion) является одним из важнейших преобразований, при котором могут объединяться различные линейные участки.

Еще одним преобразованием, выходящим за рамки одного линейного участка, является перенос операций, стоящих на критическом пути, между узлами (code motion).

Одной из важных особенностей архитектуры «Эльбрус» является явный параллелизм на уровне команд. Широкая команда «Эльбрус» позволяет исполнять до шести арифметических операций за один такт

На этапе планирования осуществляется и ряд других оптимизаций. Удаление избыточных операций чтения (redundant loads elimination) Удаление мертвого кода (dead code elimination). Динамический разрыв зависимостей по доступу в память (memory access disambiguation)

Стоит отметить, что, как следует из рис. 7, увеличение скорости работы кодов промежуточных уровней оптимизации заметно влияет на общее время работы приложения.

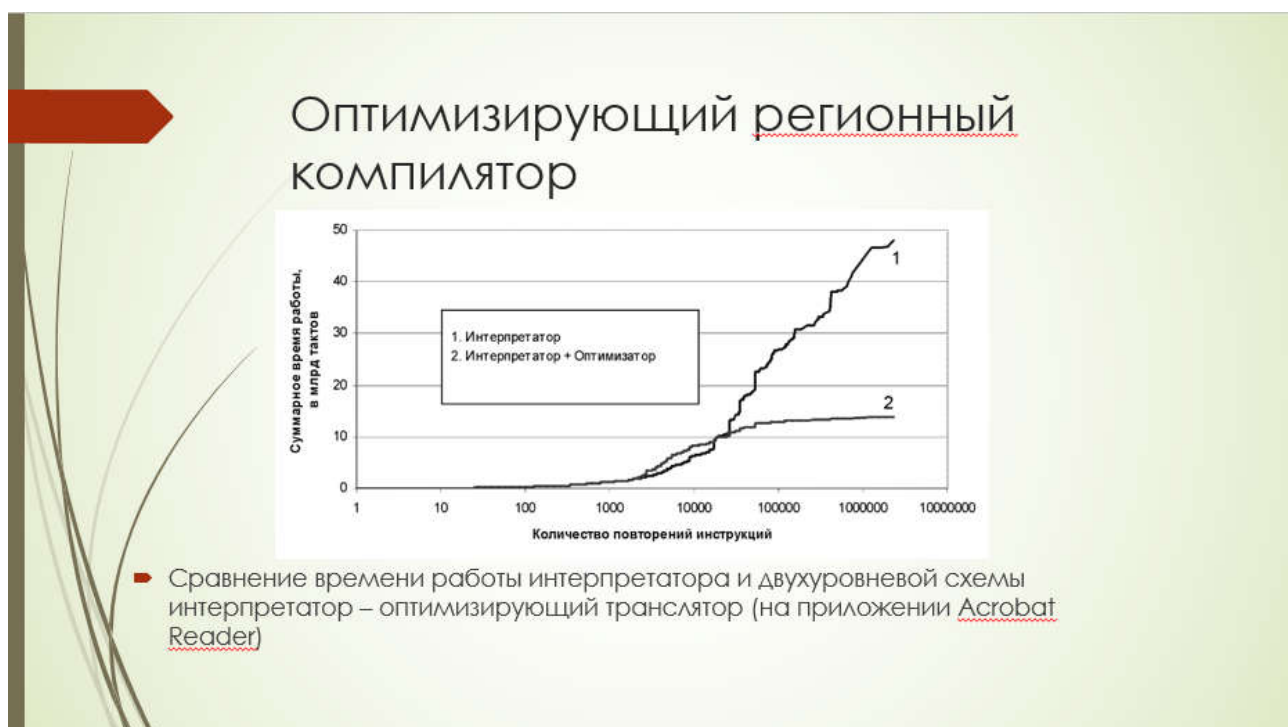


Рисунок 10. Слайд 10 – Оптимизирующий региональный компилятор

Оптимизирующий региональный компилятор

Оптимизирующий региональный компилятор является транслятором самого высокого уровня. В результате его работы получается максимально эффективный результирующий код, хотя при этом затрачивается много времени на саму трансляцию. По своему внутреннему устройству этот компилятор похож на классические языковые оптимизирующие компиляторы. Основными отличиями являются, во-первых, жесткие ограничения на скорость компиляции; во-вторых, при двоичной трансляции появляются новые семантические ограничения, накладываемые семантикой исходных двоичных кодов. Примером такого семантического ограничения является задача обеспечения точного контекста x86-ой машины при возникновении прерываний и исключений.

Его наиболее важные оптимизации

Основными оптимизациями, проводимыми оптимизирующим регион-

ным компилятором, являются: конвейеризация циклов, *if-conversion*, глобальное планирование, сокращение длины критических путей, удаление избыточных обращений в память, различные цикловые оптимизации (*unroll, peeling, nesting, tail duplication*), *operation strength reduction*, оптимизации, основанные на тождественных преобразованиях, *prefetch*. Кроме того, важную роль играют распределение регистров и планирование результирующего кода.

График на слайде сравнивает следующую схему работы, выбирается некоторый порог количества исполнений инструкций в интерпретаторе, после которого исходный код транслируется оптимизирующим компилятором. Из графика, изображенного на рис. 5, видно, что двухуровневая схема позволяет достичь существенного уменьшения времени работы.

3.9 Слайд 11 - Методы уменьшения накладных расходов на трансляции



Рисунок 11. Слайд 11 - Методы

Методы уменьшения накладных расходов на трансляцию

Несмотря на использование нескольких уровней, на трансляцию кодов тратится существенное время. Рассмотрим методы, позволяющие его сократить.

База кодов

Первым способом уменьшения накладных расходов является переиспользование ранее транслированных кодов. Для этой цели в системе двоичной трансляции предусматривается возможность долговременного хранения транслированного кода в энергонезависимой памяти (например, жёсткий диск) – базе кодов – и его переиспользования после перезагрузки системы. Для обеспечения быстрого поиска регионов в базе кодов используется хэш-таблица.

Трансляция в параллельном потоке

Другим способом сокращения накладных расходов на трансляцию является выполнение трансляции оптимизирующим компилятором на отдельном процессоре. В этом случае компиляция проводится одновременно с выполнением кодов. В такой схеме после отправки транслирующему процессору информации о компилируемом участке кода выполнение этого кода может быть продолжено с текущим уровнем оптимизаций.

3.10 Слайд 12 – Экспериментальные результаты

Экспериментальные результаты

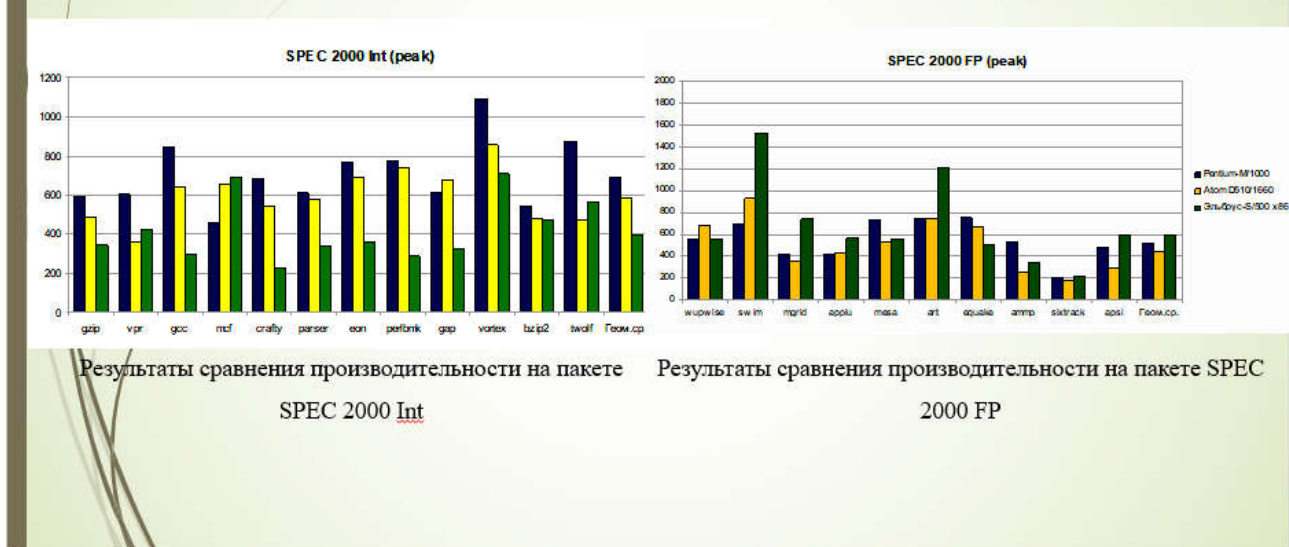


Рисунок 11. Слайд 12 - Экспериментальные результаты

В заключение приведём результаты сравнения производительности системы полной двоичной трансляции, работающей на микропроцессоре «Эльбрус-S» (частота 500 МГц) с двумя x86-микропроцессорами: Pentium-M (частота 1000 МГц) и Atom D510 (частота 1660 МГц). Сравнение проводилось на пакете тестов SPEC 2000. На рис. 8 и 9 приведены результаты целочисленных и вещественных задач, соответственно.

Данные для микропроцессора Pentium-M были взяты с официального сайта SPEC. Результаты на микропроцессорах Atom и «Эльбрус» получены авторами работы [6], при этом для обоих измерений брались одинаковые коды. Система двоичной трансляции x86 → «Эльбрус» работала со всеми описанными в данной статье технологиями и была собрана оптимизирующим языковым компилятором с высоким уровнем оптимизаций.

3 – Заключение

3.1 – Слайд 13 - Заключение

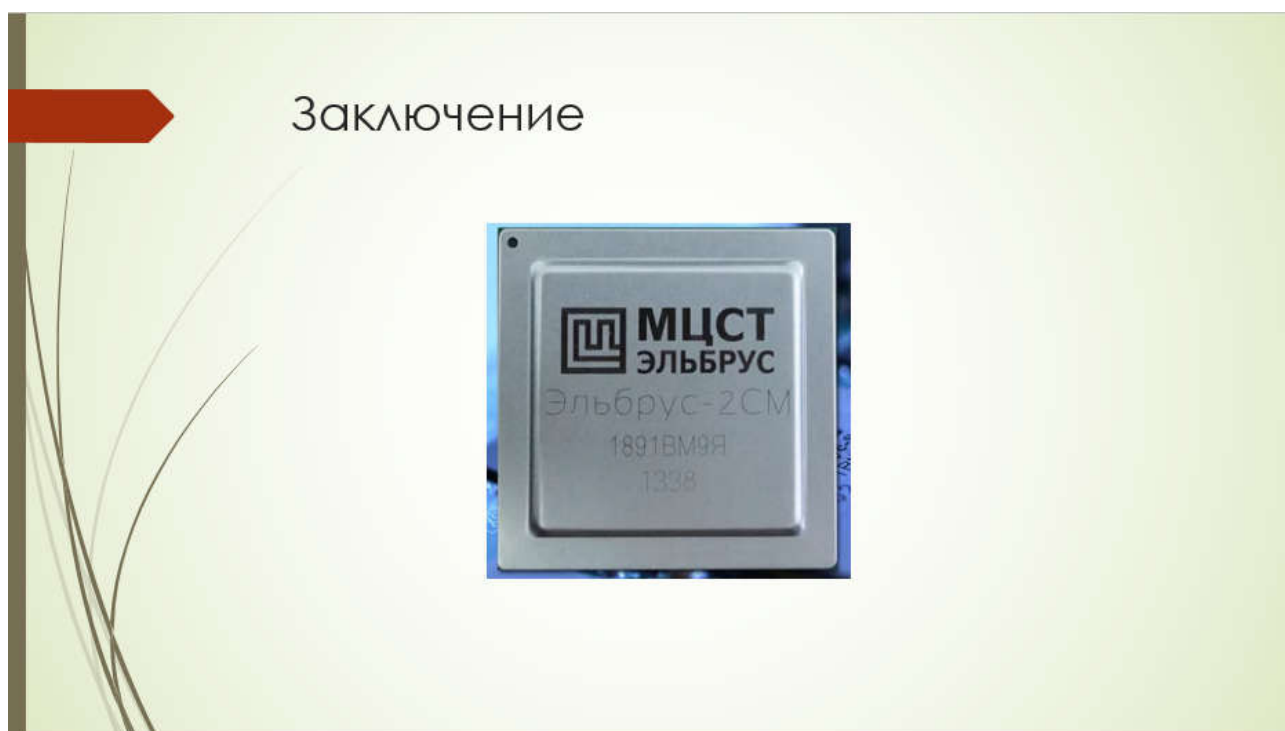


Рисунок 13. Слайд 13 – Заключение

Технология двоичной трансляции в современном мире является востребованной, так как позволяет исполнять двоичный код одной архитектуры на процессорах других архитектур. При этом при создании кода целевой платформы использование оптимизирующей компиляции способно существенно увеличить работу приложения. Важным моментом является создание адаптивных многоуровневых оптимизирующих двоичных трансляторов, позволяющих регулировать линейку и агрессивность применяемых оптимизаций в зависимости от частоты исполнения оптимизируемого кода.

3.2 Используемая литература

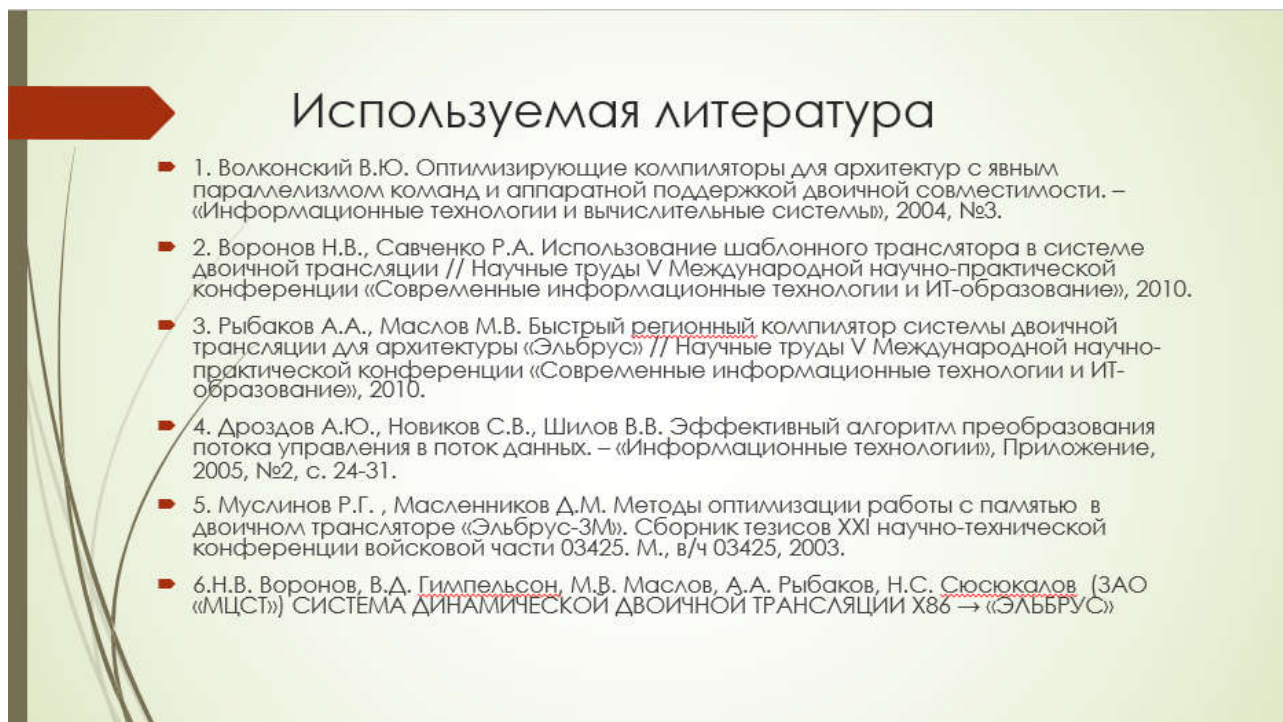


Рисунок 14. Слайд 14 – Литература

Литература

- 1. Волконский В.Ю. Оптимизирующие компиляторы для архитектур с явным параллелизмом команд и аппаратной поддержкой двоичной совместимости. – «Информационные технологии и вычислительные системы», 2004, №3.
- 2. Воронов Н.В., Савченко Р.А. Использование шаблонного транслятора в системе двоичной трансляции // Научные труды V Международной научно-практической конференции «Современные информационные технологии и ИТ-образование», 2010.
- 3. Рыбаков А.А., Маслов М.В. Быстрый регионный компилятор системы двоичной трансляции для архитектуры «Эльбрус» // Научные труды V Международной научно-практической конференции «Современные информационные технологии и ИТ-образование», 2010.

- 4. Дроздов А.Ю., Новиков С.В., Шилов В.В. Эффективный алгоритм преобразования потока управления в поток данных. – «Информационные технологии», Приложение, 2005, №2, с. 24-31.
- 5. Муслинов Р.Г., Масленников Д.М. Методы оптимизации работы с памятью в двоичном трансляторе «Эльбрус-3М». Сборник тезисов XXI научно-технической конференции войсковой части 03425. М., в/ч 03425, 2003.
- 6. Н.В. Воронов, В.Д. Гимпельсон, М.В. Маслов, А.А. Рыбаков, Н.С. Сюсюкало (ЗАО «МЦСТ») СИСТЕМА ДИНАМИЧЕСКОЙ ДВОИЧНОЙ ТРАНСЛЯЦИИ X86 → «ЭЛЬБРУС».