Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Метод трансляции машинного кода из x86 в ARM»

Студент группы **<ИУ7-83Б>**		**М. Ю. Нитенко**
	(Подпись, дата)	(И.О. Фамилия)
Руководитель ВКР		**А. А. Оленев**
	(Подпись, дата)	(И.О. Фамилия)
Нормоконтролер		**Ю. В. Строганов**
	(Подпись, дата)	(И.О. Фамилия)

***2022** г.*

РЕФЕРАТ

Расчетно-пояснительная записка 20 с., 1 рис., 0 табл., X ист., X прил.

КЛЮЧЕВЫЕ СЛОВА

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **8**

1 Аналитическая часть **9**

1.1 QEMU 9

1.1.1 Сворачивание и оптимизация тривиальных выражений 9

1.1.2 Состояние процессора и блоки трансляции 10

1.1.3 Блок чейнинг?? 10

1.1.4 Поддержка самомодифицирующегося кода 11

1.1.5 Эмуляция MMU 11

1.2 box64 11

1.2.1 Трансляция 12

1.2.2 Поддержка родных библиотек 12

1.2.3 Процесс трансляции 13

1.2.4 JITs (mono) and self-modifying code. 14

1.3 FEX 14

2 Конструкторская часть **15**

3 Технологическая часть **16**

4 Исследовательская часть **17**

ЗАКЛЮЧЕНИЕ **18**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **19**

ПРИЛОЖЕНИЕ А **20**

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ВВЕДЕНИЕ

Процессоры архитектуры ARM занимают большую долю рынка, еще в 2015 году они составляли 35% от рынка процессоров, однако в основном они использовались в портативных устройствах [1]. С появлением процессоров M1 от компании Apple большое число людей начало пользоваться компьютерами на основе архитектуры ARM в домашней обстановке (типа personal computers). Однако, программы собранные под архитектуру x86 не смогут работать на таких компьютерах, им необходим или транслятор, такой как Rosetta 2, или виртуальная машина поддерживающая необходимую архитектуру.

Виртуальные машины как правило используют динамическую трансляцию, поэтому они намного более чувствительны к ее скорости.

1 Аналитическая часть

Проблемы эмулятора:

— управление кэшем транслированного кода;

— выделение регистров;

— оптимизация условных блоков;

— direct block chaining????? а по русски;

— управление памятью;

— поддержка самоизменяемого кода;

— поддержка исключений;

— поддержка аппаратных прерываний.

1.1 QEMU

кему это круто написать что это такое [2]

1.1.1 Сворачивание и оптимизация тривиальных выражений

В qemu определены свои опкоды, каждый из них может по своему сворачиваются. TCG ops (специальные операции TCG) оптимизируются и затем выполняются на хосте, таким образом портировать TCG становится проще. Операции поддерживаются над 32 и 64 битными целыми числами, указатели определены как алиас над соответствующим целым числом.

Пример операции TCG:

```
add_i32 t0, t1, t2 (t0 <- t1 + t2)
```

Одной из оптимизаций является игнорирование бессмысленных операций, например операция:

```
and_i32 t0, t0, $0xffffffff
```

выполняться не будет

Подавляются неиспользуемые перемещения данных, например из трех операций:

```
add_i32 t0, t1, t2
add_i32 t0, t0, $1
```

```
mov_i32 t0, $1
```

выполнится только последняя.

(оптимизации над инструкциями реализованы в файле tcg/optimize.c)

1.1.2 Состояние процессора и блоки трансляции

Блоком трансляции называется часть кода до момента изменения состояния процессора которое нельзя выяснить на этапе трансляции (например, некоторое ветвление).

Каждый регистр проверяется на используемость в определенном блоке трансляции, если регистр не используется в блоке трансляции связанные с ним операции оптимизируются, так как значения этих регистров (и связанное с ними состояние процессора) за блок не могут измениться. Если состояние виртуального процессора меняется, такой блок не будет выполняться пока состояние процессора не будет соответствовать необходимому для блока (например, другой уровень привилегий).

Пример: если на x86 регистры SS, DS и SS содержат в себе 0, транслятор не будет генерировать для них смещение.

1.1.3 Блок чейнинг??

После выполнения блока трансляции QEMU ищет следующий блок, для этого используется PC (эмулируемый program counter) и другая информация о статусе процессора (например регистр CS). (реализовано в функции tb_lookup в файле accel/tcg/cpu-exec.c)

Сначала блок ищется в кэше трансляции, если он там не найден он достается из хэш-таблицы и добавляется в кэш.

Для этого нужно выйти из цикла выполнения блока, пройти через эпилог процедуры, найти следующий блок, запустить его, пройдя через пролог процедуры. В качестве оптимизации предлагается связывать несколько блоков трансляции напрямую.

Для этого в конце блока вызывается tcg_gen_lookup_and_goto_ptr(), он в свою очередь вызывает helper_lookup_tb_ptr который ищет нужный блок и

генерирует инструкцию goto_ptr, которая либо продолжит управление в нужном блоке, либо вернется в основной цикл. (пример вызова для i386 в файле target/i386/tcg/translate.c в функции do_gen_eob_worker)

Еще одной оптимизацией является оптимизация ветвления. Если ветвление происходит напрямую, в пределах одной страницы QEMU выполняет поиск блока трансляции на который будет произведено ветвление, а затем сохраняет его адрес в транслированном коде. Таким образом при следующем выполнении этого блока нет необходимости в поиске следующего блока.

1.1.4 Поддержка самомодифицирующегося кода

Самомодифицирующийся код на x86 представляет особую проблему, так как нет механизма оповещения о изменении кода. При запуске в режиме пользователя QEMU помечает все страницы с транслированным кодом как защищенные от записи, при попытке записи в них поднимается сигнал SEGV, допускается запись, обнуляются все транслированные страницы и связь блоков. При запуске эмуляции системы программный MMU выполняет защиту от записи. Для эффективного выполнения этих операций хранится связанный список всех блоков трансляции и связей блоков.

1.1.5 Эмуляция MMU

Каждый доступ к памяти проходит через MMU, в нем адреса преобразуются из виртуальных к реальным. Для ускорения трансляции адреса хранится TLB-кэш.

Все блоки трансляции в кэше индексируются при помощи их физического адреса, таким образом нет необходимости очищать кэш при смене страницы. [3]

1.2 box64

box64 это норм написать что такое

В отличии от QEMU box64 не может эмулировать полную систему (то есть на нем нельзя, например, запустить Windows), однако можно запускать linux-приложения собранные под x86_64.

1.2.1 Трансляция

Трансляция в box64 производится при помощи dynarec или при помощи эмулятора процессора. Для ARM реализован полноценный dynarec, для всех остальных архитектур написан общий эмулятор процессора интерпретирующий инструкции. Dynarec написан на C, на ассемблере реализованы пролог и эпилог.

1.2.2 Поддержка родных библиотек

Главным механизмом позволяющим добиться хорошей производительности является использование родных для архитектуры библиотек. Таким образом в графически требовательных приложениях производительность близка к 100%, однако в приложениях не использующие сторонние библиотеки (то есть полностью транслируемые) производительность около 50%.

При эмуляции Quake 3 (графического приложения с большим количеством повторно используемых функций) производительность была около 85%.

Необходимые для работы приложения библиотеки либо транслируются, либо используется родная библиотека (в таком случае производительность выше). Вызовы функций перехватываются для вызова родных функций.

В ELF содержатся специальные символы называемые перемещениями (relocati они используются при линковке для установки адреса необходимой функции. При вызове родной функции в качестве адреса выставляется адрес заранее подготовленного кода, он состоит из CC 53 43 и двух указателей. Первый указатель рассматривается как указатель на оберточную функции, а второй как указатель на обертываемую (родную) функцию. Оберточной функции передается структура с состоянием процессора и указатель на вызываемую функцию, эта структура распаковывается и вызывается переданная функция. По завершению работы оберточная функции завершается через ret или retn.

Поиск необходимой функции осуществляется при помощи файлов находящихся в src/wrapped, их необходимо определять в ручную, так как названия

функций не сохраняются после компиляции программы. Сигнатура функции состоит из символов определяющих тип возвращаемого значения и типов аргументов разделенных буквой F. Пример сигнатуры показан на рисунке 1.

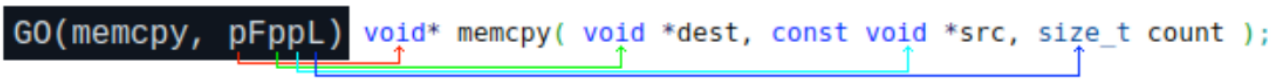

Рисунок 1 – Пример сигнатуры функции memcpy

После определения функция заносится в таблицу функций библиотеки, а затем находится при помощи разных методов (// посмотреть что за методы).

Одной из оптимизаций в box64, по сравнению с box86, является определение простых функций, такие функции не нуждаются в оберточной функции и вызываются напрямую, тем самым экономя время. [5]

1.2.3 Процесс трансляции

Так же как и в QEMU код разбивается на блоки трансляции, блок заканчивается когда после нее нет других инструкций (например jump, call или ret) и когда на последнюю инструкцию не ссылается какое-либо ветвление из этого блока. Исключения: multi-bytes NOP are handled, а неизвестная инструкция останавливает процесс выполнения блока.

Каждый блок транслируется в 4 прохода:

— Первый проход считает все количество транслируемых x86 инструкций. Для каждой инструкции выделяется память;

— На втором проходе обрабатываются все инструкции ветвления, проверяется где находится адрес: в этом же блоке или в каком-либо другом. В случае если адрес находится в другом блоке его необходимо связать (// LinkNext() ?) с текущим. Также рассчитываются флаги, не нужные флаги предлагается не рассчитывать, так как инструкция может использовать/выставлять не все флаги;

— На третьем проходе считается количество необходимых ARM инструкций в блоке;

— На четвертом проходе генерируются необходимые инструкции.

После генерации блока он записывается в таблицу сгенерированных блоков в которой также содержится отображение эмулированных адресов в физические. [4]

1.2.4 JITs (mono) and self-modifying code.

1.3 FEX

напишем

2 Конструкторская часть

3 Технологическая часть

4 Исследовательская часть

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Annual Report 2015: Strategic Report [Электронный ресурс]. – Режим доступа: https://media.corporate-ir.net/media_files/IROL/19/197211/2016CustomWork/ARM_Strategic_Report.pdf, свободный – (24.11.2021) (преза арма как оформить??)

2. Bellard F. QEMU, a Fast and Portable Dynamic Translator [Текст] / Bellard F. // FREENIX Track: 2005 USENIX Annual Technical Conference. – 2005. – С. 41-42.

3. QEMU: Translator Internals [Электронный ресурс]. – Режим доступа: https://qemu.readthedocs.io/en/latest/devel/tcg.html, свободный – (11.12.2021)

4. box64: Inner workings [Электронный ресурс]. – Режим доступа: https://box86.org/2021/07/inner-workings-a-high%e2%80%91level-view-of-box86-and-a-low%e2%80%91level-view-of-the-dynarec/, свободный – (11.12.2021)

5. box64: A deep dive into library wrapping [Электронный ресурс]. – Режим доступа: https://box86.org/2021/08/a-deep-dive-into-library-wrapping/, свободный – (11.12.2021)

ПРИЛОЖЕНИЕ А