Санкт-Петербургский государственный университет

Программная инженерия

Балашов Илья Вадимович

Оптимизация эмулятора RISC-V, порожденного по спецификации SAIL

Отчёт по учебной (проектно-технологической) практике

Научный руководитель:
к.ф.-м.н., доцент кафедры СП Д.В.Луцив
Консультант:
старший преподаватель кафедры СП Я.А.Кириленко

Санкт-Петербург
2023

Оглавление

Введение

Как известно, архитектуру процессора возможно задавать не только непосредственно в физическом виде, но и программно. Для этого в виде программного кода создаётся так называемая архитектура, заданная инструкциями (instruction set architecture, ISA). В ISA могут описываться возможные инструкции процессора, их поведение, логические правила, задержки, структура кешей и другое [1]. Такого описания рассматриваемой архитектуры, как правило, оказывается достаточно для создания программного эмулятора, позволяющего исполнять исполняемые файлы для целевой (гостевой) архитектуры на некой другой (хостовой) архитектуре [2]. Задачей эмулятора может являться, к примеру, тестирование особенностей нужной архитектуры без наличия физического процессора, который бы реализовывал эту архитектуру. Причем подобного физически существующего процессора может не быть вовсе, поэтому особенную важность приобретает задача генерации эмуляторов, инструментов верификации и тестирования и прочих исключительно из программной модели ISA.

Одной из архитектур, для которой наиболее актуально стоит задача генерации эмуляторов, является RISC-V. На данный момент сообщество и профильные компании создаю большое количество ядер RISC-V, которые не реализованы и в ближайшей перспективе не будут реализованы «в кремнии», поэтому создание эмуляторов для этой архитектуры является особенно актуальным. Одним из наиболее популярных инструментов для генерации эмуляторов является Sail.

По причине наличия больших накладных расходов на исполнения на другой архитектуре, исполнение программы на эмуляторе медленнее по времени, чем исполнения на физическом процессоре. Подобные задержки затрудняют проверку новых ядер RISC-V сообществом, работающим с Sail, а также в целом работу по созданию новых архитектур. Соответственно, особенно остро встает вопрос оптимизации эмуляторов Sail для RISC-V.

1. Постановка задачи

Целью данной работы является исследование возможностей оптимизации эмулятора архитектуры RISC-V, порожденного из её спецификации на языке Sail.

Для достижения указанной цели были поставлены указанные далее задачи.

- Выполнить обзор
  - основных понятий области, архитектуры RISC-V
  - языка и генератора эмуляторов SAIL
  - профилировщиков и бенчмарков производительности
  - подходящих для бенчмарков алгоритмов
- Выделить и реализовать подходы к оптимизации эмулятора RISC-V из SAIL
- Провести экспериментальное исследование данных подходов проанализировать результаты
- Сформулировать планы для дальнейшего исследования

2. Обзор предметной области

В данной главе будет представлен обзор основных понятий, алгоритмов и подходов, которые будут использованы для достижения цели работы.

2.1. Основные понятия

В данной главе будут представлены основные понятия, используемые в данной работе.

2.1.1. Инструмент Sail

Существует большое число инструментов, позволяющих генерировать эмуляторы из ISA процессоров [2]. Одним из подобных инструментов является Sail [3].

Sail был выбран в качестве объекта данной работы, поскольку он называется опрошенными практикующими инженерами одним из наиболее перспективных инструментов для генерации эмуляторов.

Язык Sail описания ISA представляет собой функциональный язык, по синтаксису схожий с Rust. На данный момент написаны модели различных архитектур, включая ARM [4], RISC-V [4], Power [3] и другие. По моделям на данном языке Sail генерирует эмулятор на языке Си (либо OCaml, который не рассматривается в рамках работы).

2.1.2. RISC-V

RISC-V — свободная архитектура процессора, впервые представленная в университете Беркли, США в 2010 году Разработана citewaterman2016de по концепции RISC [5], то есть обладает сравнительно небольшим набором инструкций с возможностью дополнения с помощью расширений. Отличительной особенностью является активное развитие на момент начала 2020х годов [6] [7] [8], что в рамках исследования дает доступ к широкому набору зрелых инструментов (компиляторы, эмуляторы и др.), упрощающий достижение цели работы.

2.2. Алгоритмы для экспериментов

Для выполнения экспериментального исследования необходимо выбрать набор алгоритмов, который бы подавался на вход эмулятору, для тестирования влияния проведенной оптимизации.

Алгоритмы отбирались по следующим критериям:

- недолгое исполнение (десятки секунд на некотором наборе входных параметров);
- стабильная воспроизводимость результата и времени исполнения;
- возможность регулирования количества итераций в алгоритме параметрами;
- применимость алгоритма в прикладных сценариях;
- однопоточность — поскольку эмулятор Sail не поддерживает многопоточное исполнение;

Таким образом, были отобраны четыре представленных далее алгоритма, отвечающие всем перечисленным критериям.

1. SHA-256 [9] — алгоритм хеширования, применяется в криптографии.
2. Модель Блека-Шоулза [10] — алгоритм расчета стоимости опционов на европейском финансовом рынке.
3. Алгоритм Якоби — алгоритм решения систем линейных уравнений, применяется в численных вычислениях.
4. Алгоритм поиска пути на сетке — применяется в графовых вычислениях, также в FPGA [11].

2.3. Подходы к оптимизации

Из теории оптимизации алгоритмов можно выделить три основных уровня оптимизации, представленные ниже [12].

- Архитектурный.
- Алгоритмический.
- Низкоуровневый.

В случае с Sail архитектурная оптимизация не представляется возможной, поскольку эмулятор состоит ровно из одного модуля. Алгоритмическая (на уровне алгоритмов) оптимизация теоретически возможна, поскольку рассматриваемый эмулятор представляет собой автомат с конечным количеством состояний, то есть применимы оптимизации для графовых и автоматных алгоритмов. Тем не менее, после анализа кода было выяснено, что в эмуляторе не наблюдается очевидных мест для оптимизации алгоритмов, поэтому потенциал подобного подхода оценивается как низкий. Наиболее многообещающим для ускорения эмулятора выглядит применение низкоуровневых оптимизаций — попадание в кеш-линию, улучшение качества спекуляции процессора, оптимизация аллокаций памяти и другое.

Таким образом, на данном этапе работы перспективным направлением для оптимизации выбрана оптимизация на низком уровне.

2.4. Подходы к замеру производительности

Для оценки ускорения эмулятора в результате проведенных оптимизаций необходим инструмент, позволяющий воспроизводимо и статистически корректно измерить время выполнения эмулятора. Более подробно, нужен инструмент, обладающий следующими свойствами:

- позволяет проводить замер статистически, по нескольким исполнениям программы;
- позволяет оценить погрешность измерений;
- может работать с исполняемыми бинарными файлами.

Были рассмотрены три часто используемых в прикладных задачах инструмента, перечисленных ниже.

- bash time — встроенная в GNU bash утилита замера времени исполнения. Позволяет работать с бинарными файлами, однако не имеет возможностей для статистических замеров и оценки погрешности.

- Google Benchmark [1] — свободных инструмент для проведения бенчмарков, представлен в виде библиотеки.. Позволяет измерять время исполнения по нескольким исполнениям и оценивать погрешность, но не предоставляет возможностей для работы с уже собранными программами (только в виде кода).

- hyperfine [2] — также свободных инструмент для проведения бенчмарков, представлен в виде утилиты командной строки. Позволяет производить статистические замеры времени исполнения, а также предназначен для работы с бинарными файлами (интерфейс схож с bash time).

Подводя итог, из рассмотренных альтернатив для проведения эксперимента выбран инструмент hyperfine.

Кроме того, для проведения профилирования программ в рамках подготовки к экспериментам использовался инструмент Intel Vtune [13]. Инструмент был выбран по причине наличия следующих возможностей:

- проведение анализа «горячих мест» программы;

- наличие GUI для упрощения интерпретации результатов;

- возможность работы на удаленной виртуальной машине, что важно по причине наличия доступа к вычислительным ресурсам только в таком виде.

Поскольку выбор профилировщика не оказывает непосредственного влияния на результаты экспериментов, инструмент Vtune был выбран по перечисленным выше причинам и исходя из предпочтений автора.

[1] Репозиторий Google Benchmark: https://github.com/google/benchmark (дата обращения: 08.03.2023)
[2] Репозиторий hyperfine: https://github.com/sharkdp/hyperfine (дата обращения: 08.03.2023)

3. Постановка эксперимента

В данном разделе будут описаны проведенные эксперименты, а также способы их постановки. На этапе учебной практики будет описан лишь один эксперимент.

3.1. Оптимизация аллокаций памяти

По результатам анализа профилировщиком исполняемого файла эмулятора RISC-V, порожденного инструментом Sail, было выявлено, что около 80% времени исполнения программы в эмуляторе затрачивается на аллокации памяти функцией malloc, и её освобождение методов free. Указанные методы реализованы в стандартной библиотеке языка Си libc.

Таким образом, наиболее перспективной низкоуровневой оптимизаций выглядела подмена стандартных реализаций malloc/free на более оптимальные. В качестве подменных реализаций были рассмотрены следующие:

- mimalloc;
- rpmalloc;
- jemalloc.

Три указанных аллокатора были выбраны по причине их лидерства в наиболее свежих из обнаруженных синтетических тестах производительности [3]. Подмена аллокатора производилась динамически, то есть подключением библиотеки на этапе исполнения эмулятора с подменой всех вызовов malloc/free на оптимизированные. В качестве алгоритма на данном этапе работы был рассмотрен SHA-256. Внутри алгоритма были зафиксированы итерации от одной до 50 с целью минимизации влияния накладных расходов на запуск программы на итоговые результаты.

[3]Бенчмарк: https://github.com/microsoft/mimalloc#benchmark-results-on-a-16-core-amd-5950x-zen3 (дата обращения: 08.03.2023)

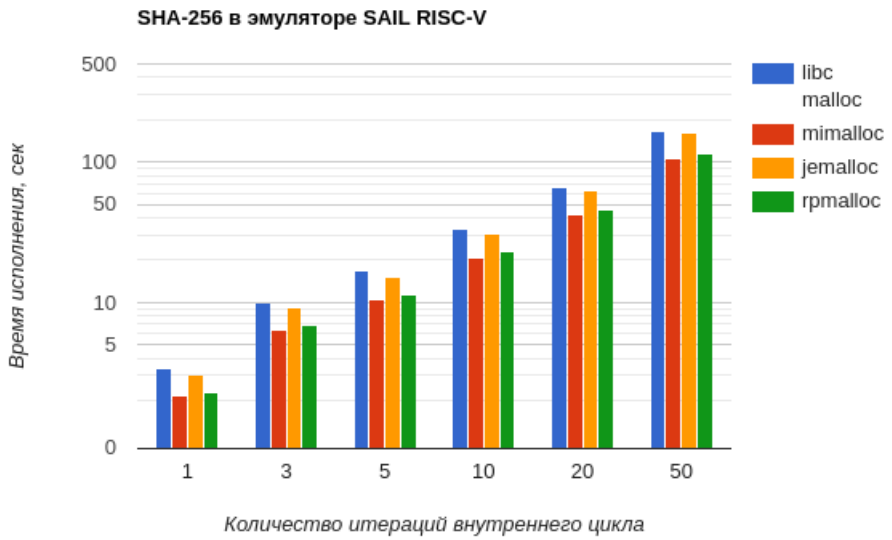

Рис. 1: Результаты замеров производительности SHA-256 в эмуляторе Sail RISV-V с разным количеством внутренних итераций

В качестве тестового стенда исползовалась следующая конфигурация: Yandex Cloud, Intel Xeon Gold 6338 8x2.0Ghz (гарантированный процент CPU — 100%), 32GB RAM, Ubuntu 22.04.

В результате запуска под инструментов hyperfine были получены результаты, представленные на картинке 1.

Подводя итог, было получено ускорение около 50% на аллокаторе mimalloc, около 45% на аллокаторе rpmalloc. Ускорение на аллокаторе jemalloc в пределах погрешности. Статистическая погрешность — около 2%.

4. Дальнейшие планы

В качестве дальнейших планов можно выдвинуть описанные далее задачи.

- Реализовать остальные упомянутые алгоритмы для бенчмарков
- Выделить и реализовать
  - Низкоуровненые оптимизации процента попадания в кеш и качества спекуляции
  - Алгоритмические оптимзации эмулятора
- Сделать Pull Request в репозиторий SAIL

5. Результаты

По результатам работы, проведённой в рамках учебной практики, были выполнены все поставленные задачи, перечисленные ниже.

- Выполнен обзор:
  - основных понятий области, архитектуры RISC-V;
  - языка и генератора эмуляторов SAIL;
  - профилировщиков и бенчмарков производительности;
  - подходящих для бенчмарков алгоритмов.
- Выделены основные подходы к оптимизации эмулятора RISC-V из SAIL.
- Проведено экспериментальное исследование низкоуровневой оптимизации с malloc.
  - Получено ускорение в 1.5 раза.
- Сформулированы планы для дальнейшего исследования.

Список литературы

[1] Cambricon: An instruction set architecture for neural networks / Liu Shaoli, Du Zidong, Tao Jinhua, Han Dong, Luo Tao, Xie Yuan, Chen Yunji, and Chen Tianshi // ACM SIGARCH Computer Architecture News. — 2016. — Vol. 44, no. 3. — P. 393–405.

[2] Roelke Alec, Stan Mircea R. Risc5: Implementing the RISC-V ISA in gem5 // First Workshop on Computer Architecture Research with RISC-V (CARRV). — 2017. — Vol. 7.

[3] The sail instruction-set semantics specification language : Rep. / Technical report published by Cambridge University ; executor: Gray Kathryn E, Sewell Peter, Pulte Christopher et al. : 2017.

[4] ISA semantics for ARMv8-a, RISC-v, and CHERI-MIPS / Armstrong Alasdair, Bauereiss Thomas, Campbell Brian, Reid Alastair, Gray Kathryn E, Norton-Wright Robert, Mundkur Prashanth, Wassell Mark, French Jon, Pulte Christopher, et al. — 2019.

[5] Aletan Samuel O. An overview of RISC architecture // Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing: technological challenges of the 1990's. — 1992. — P. 11–20.

[6] The case for RISC-V in space / Di Mascio Stefano, Menicucci Alessandra, Furano Gianluca, Monteleone Claudio, and Ottavi Marco // Applications in Electronics Pervading Industry, Environment and Society: APPLEPIES 2018 6 / Springer. — 2019. — P. 319–325.

[7] A comparative survey of open-source application-class RISC-V processor implementations / Dörflinger Alexander, Albers Mark, Kleinbeck Benedikt, Guan Yejun, Michalik Harald, Klink Raphael, Blochwitz Christopher, Nechi Anouar, and Berekovic Mladen //

Proceedings of the 18th ACM International Conference on Computing Frontiers. — 2021. — P. 12–20.

[8] Giorgi Roberto, Mariotti Gianfranco. Webrisc-v: A web-based education-oriented risc-v pipeline simulation environment // Proceedings of the workshop on computer architecture education. — 2019. — P. 1–6.

[9] Standard Secure Hash. FIPS Pub 180-1 // National Institute of Standards and Technology. — 1995. — Vol. 17, no. 180. — P. 15.

[10] MacBeth James D, Merville Larry J. An empirical examination of the Black-Scholes call option pricing model // The journal of finance. — 1979. — Vol. 34, no. 5. — P. 1173–1186.

[11] Zapletina Mariya A, Zheleznikov Daniil A, Gavrilov Sergey V. Improving Pathfinder Algorithm Perfomance for FPGA Routing // 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus) / IEEE. — 2021. — P. 2054–2057.

[12] Knuth Donald Ervin. The art of computer programming. — Pearson Education, 1997. — Vol. 3.

[13] Reinders James. VTune performance analyzer essentials. — Intel Press Santa Clara, 2005. — Vol. 9.