Инструментарий для Профилирования и Анализа Графа Вызовов для RISC-архитектур на Основе Трасс Исполнения Программы

Шурыгин А. А.[1], Долгов А. А.[1], Петушков И. В.[1]

[1] *Московский физико-технический институт (национальный исследовательский университет)*
141701, Московская область, г. Долгопрудный, Институтский переулок, д.9.
shurygin.aa@phystech.edu

Аннотация. В работе решается проблема анализа поведения программ для RISC-архитектур на основе бинарных трасс исполнения. Существует много исследований, которые рассматривают применение трасс для анализа исполнения и оптимизаций, однако количество работ, которые делают генерацию профиля на основе трасс для RISC-архитектур сильно ограничено. В рамках работы был улучшен алгоритм генерации профиля, а также поддержана его последующая визуализация с помощью приложения KCachegrind с точностью до линейных участков кода. Алгоритм генерации спроектирован так, чтобы время профилирования в среднем линейно зависело от размера обрабатываемой трассы. В результате работы были получены точные профили исполнения приложений на наборе тестов производительности SPEC CPU 2017.

Ключевые слова: *профилирование; трассы исполнения; RISC; граф вызовов, граф потока управления.*

Для оптимального использования ресурсов вычислительной системы важно уметь производить анализ работы приложений. Одним из методов анализа служит динамическая бинарная инструментация[1], часто используемая для сбора данных о поведении программы. Бинарная трасса исполнения – один из наиболее полных вариантов собранной информации, который впоследствии можно использовать для изучения наиболее репрезентативных участков работы приложения. Однако обработка больших трасс – задача трудоемкая, в виду чего возникает необходимость использовать дополнительный инструментарий, который позволит разобрать трассу, проанализировать приложение на требуемом уровне представления и визуализировать результат, представив его в наиболее понятной для человека форме. Предыдущая работа по этой теме[2] была сосредоточена на проблеме генерации профиля на основе бинарных трасс и последующей визуализации графа вызовов, чего недостаточно для подробного изучения исполнения приложения. Для детального анализа профилировщик должен уметь собирать всю необходимую статистику о переходах, количестве исполненных инструкций и на выход выдавать граф вызовов или граф потока управления.

Одним из возможных инструментов профилирования является программа Callgrind[4] из пакета динамического анализа приложений Valgrind[3]. Использовать Callgrind напрямую для генерации профиля на основе трасс невозможно, так как Valgrind инструментирует исполняемые файлы, а конвертация бинарной трассы в исполняемый файл не позволит получить точный профиль исполнения программы. Кроме того Callgrind не всегда корректно обрабатывает передачу потока управления, что особенно критично для программ на RISC-архитектурах, в которых нет явных call-инструкций. При этом профилировщик обладает важным достоинством: собранный профиль исполнения можно визуализировать, используя KCachegrind[4] – одно из лучших средств визуализации с графическим интерфейсом и открытым кодом. Поэтому было принято решение взять формат вывода Callgrind, грамматика которого описана в документации, за эталонный как

для последующей визуализации профиля с помощью KCachegrind, так и для верификации сгенерированных профилей на синтетических тестах.

Решая проблему детального анализа исполнения приложений на основе бинарных трасс исполнения, мы разработали:

- Алгоритм генерации детального профиля, работающий с линейной асимптотикой, для RISC-архитектур на основе трассы.
- Алгоритм разбиения и визуализации линейных участков кода в приложении KCachegrind.
- Набор вспомогательных инструментов для дизассемблирования и декодирования машинного кода под RISC-архитектуру.

Алгоритм генерации профиля был разработан так, чтобы в результате одного прохода по трассе обработчик формировал граф вызовов, корректно соотнося вызывающие и вызываемые функции. При этом всем функциям сопоставляются их линейные участки кода без внесения дополнительной алгоритмической сложности выше линейной. Алгоритм был протестирован как на синтетических тестах, так и на наборе тестов производительности SPEC CPU 2017[5]. На рис. 1 представлены графики зависимости времени работы генератора профиля от числа исполненных инструкций согласно трассе.

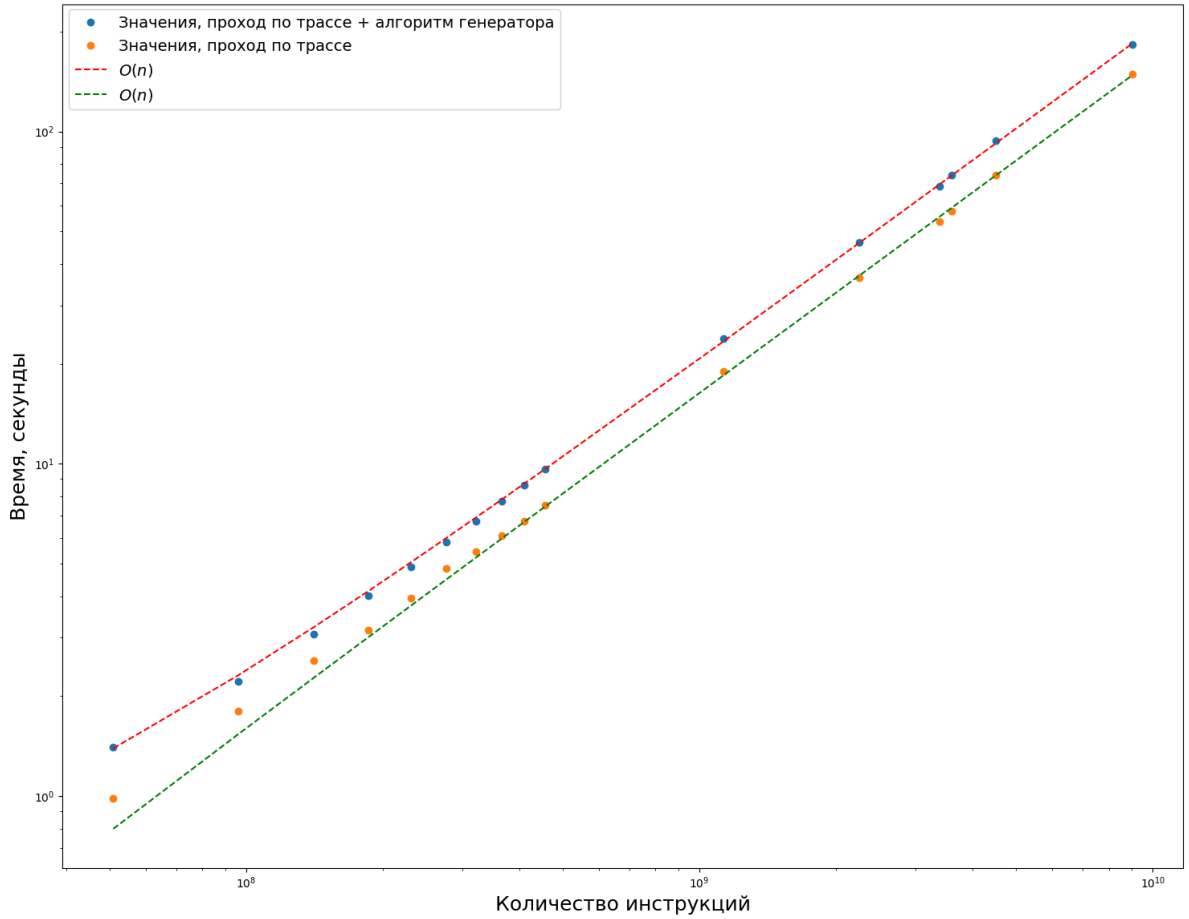

Рис. 1. График сравнения асимптотики работы алгоритма и линейного прохода по трассе

По обеим осям логарифмический масштаб, поскольку количество инструкций варьируются в широком диапазоне. На одних и тех же тестах были собраны данные двух сценариев работы: обычный линейный проход по трассе, линейный проход со

включенным алгоритмом генератора. Из рис. 1 видно, что генератор не вносит дополнительные накладные расходы, влекущие за собой нелинейную асимптотику.

На рис. 2 представлена визуализация профиля одного из синтетических тестов.

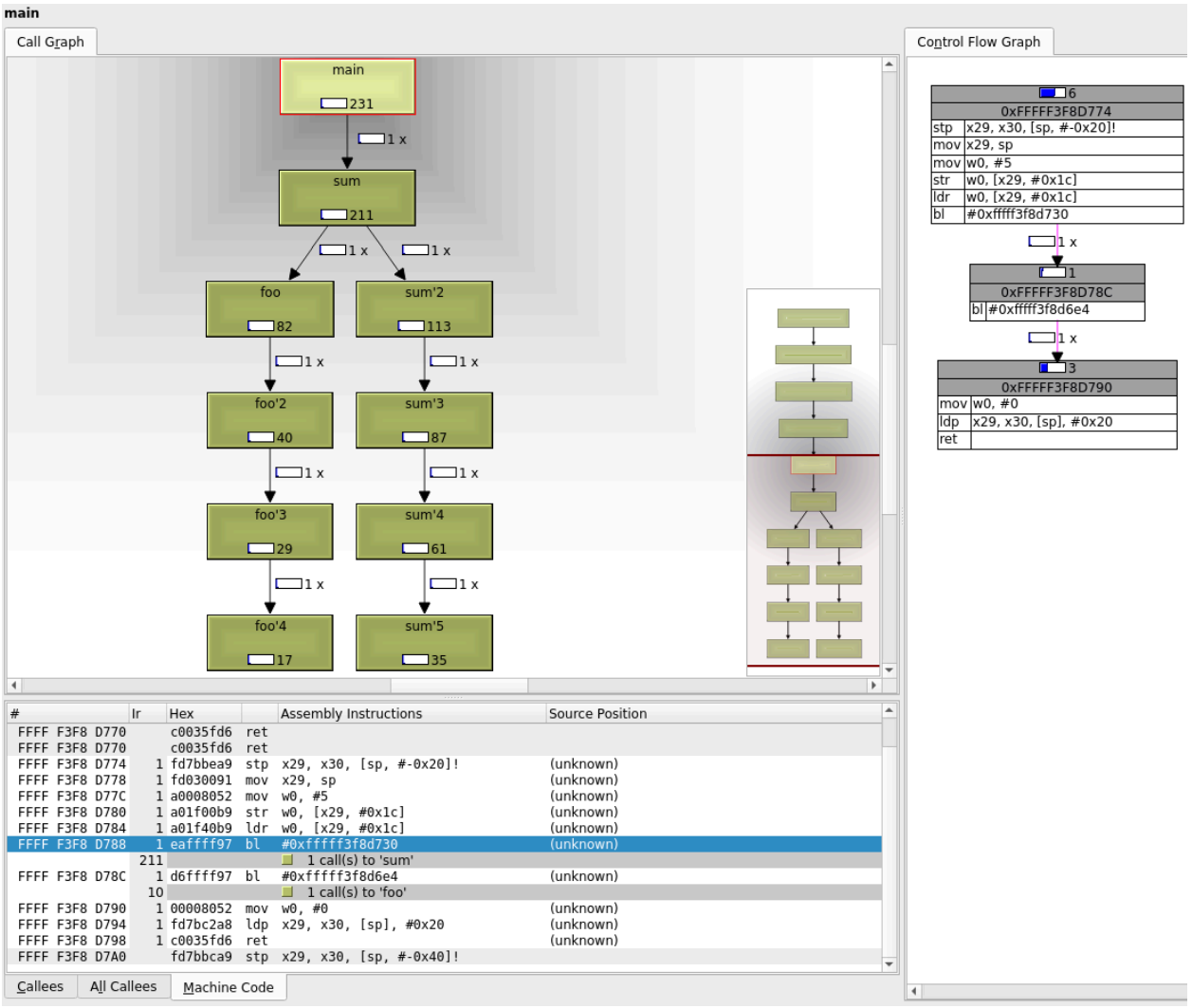

Рис. 2. Визуализация графа вызовов по функциям и по линейным участкам кода в программе KCachegrind

Инструментарий позволяет корректно сопоставлять машинный код, полученный от аналога objdump, разработанного для анализа бинарных трасс, с каждой функцией из графа вызовов.

Разбиение машинного кода на линейные участки для дальнейшей визуализации реализовано в пределах функций. Понятие линейного участка совпадает с таковым из теории компиляторов[6], то есть линейный участок – последовательность инструкций, управление в которую может войти только в начале, а выйти только в конце. Классически вызовы функций не являются инструкциями-терминаторами, но в нашем алгоритме это не так, поскольку в RISC-архитектурах вызов функции – разновидность безусловного перехода. Основными трудностями, с которыми пришлось столкнуться в процессе разработки алгоритма, оказались отсутствие во внутреннем представлении KCachegrind информации о том, что та или иная инструкция возвращает управление вызывающей стороне, а также неточности подсчёта числа переходов в профилях, полученных от Callgrind. Преодолеть обе трудности оказалось возможным, используя число исполнений

инструкций для разграничения линейных участков и как более приоритетную метрику, чем число переходов при изменении потока управления.

Таким образом, был реализован инструментарий, позволяющий сгенерировать и проанализировать точный профиль на основе бинарной трассы исполнения приложения. Специфика работы с трассами позволяет за линейное время получить профиль наиболее репрезентативных участков без необходимости повторного запуска программы. Данное решение может быть полезно для технологии симуляции на основе трасс в рамках микроархитектурных оптимизаций и исследований. Разработанный алгоритм профилирования учитывает особенности изменения потока управления согласно документации архитектуры ARM[7], как пример наиболее популярной RISC-архитектуры, а визуализация линейных участков кода позволяет разработчику лучше понимать сценарий исполнения программы на уровне машинных инструкций.

Литература

1. Nethercote N. Dynamic binary analysis and instrumentation. – University of Cambridge, Computer Laboratory, 2004. – №. UCAM-CL-TR-606.
2. Шурыгин А.А, Петушков И.В. Генерация детального профиля и его визуализация для RISC-архитектур на основе трасс исполнения программы //Труды 65-й Всероссийской научной конференции МФТИ в честь 115-летия Л.Д.Ландау, 3–8 апреля 2023 г. Радиотехника и компьютерные технологии. — М: Физматкнига, 2023. – С. 23-24.
3. Nethercote N., Seward J. Valgrind: a framework for heavyweight dynamic binary instrumentation //ACM Sigplan notices. – 2007. – Т. 42. – №. 6. – С. 89-100.
4. Weidendorfer J. Sequential performance analysis with callgrind and kcachegrind //Tools for High Performance Computing: Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing, July 2008, HLRS, Stuttgart. – Springer Berlin Heidelberg, 2008. – С. 93-113.
5. Bucek J., Lange K. D., v. Kistowski J. SPEC CPU2017: Next-generation compute benchmark //Companion of the 2018 ACM/SPEC International Conference on Performance Engineering. – 2018. – С. 41-42.
6. Ахо А., Сети Р., Ульман Д. Компиляторы //Принципы, технологии, инструменты. М.: Вильямс. – 2003.
7. ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile. ARM DDI 0487B.a ID033117. Arm Ltd., 2013–2017.