Оптимизация использования памяти в двоичном трансляторе

Черемнов Андрей Валерьевич

Студент

Факультет ВМК МГУ имени М. В. Ломоносова, Москва, Россия

E-mail: cheremnov.av@yandex.ru

Научный руководитель *— Батузов Кирилл Андреевич*

Растёт конкуренция среди процессорных архитектур, поэтому возрастает потребность в эффективной эмуляции одной архитектуры на базе другой. Это обеспечивается динамической двоичной трансляцией, но излишнее потребление памяти может её замедлить.

В работе рассматриваются методы оптимизации использования памяти для двоичного транслятора x86-64 в ARM. Под памятью понимаются физические страницы. Накладные расходы на работу с памятью: выделение и заполнение нулями новой страницы при доступе в не отображённую память, копирование — особенно заметны в программах с большим количеством «холодного» кода.

Традиционно, «холодным» кодом называют редко исполняемый код [1], в противовес «горячему». При двоичной трансляции в «горячем» коде основные затраты процессорного времени приходятся на исполнение транслированного кода, в «холодном» — на трансляцию. Большая часть тестов производительности — таких, как SPECCPU [2] — ориентированы на «горячий» код. Но доля «холодного» кода в реальных приложениях — к примеру, содержащих большой объём JavaScript кода — часто оказывается существенной. Поэтому ограничиться в оценках эффективности только тестами «горячего» кода недостаточно — нужны и тесты, содержащие преимущественно «холодный» код.

При оценке эффективности важно знать, как меняется потребление памяти с течением времени. Существующие средства измерения памяти не учитывают специфику двоичной трансляции. Поэтому был разработан инструмент, который составляет профили потребления памяти, исходя из информации о внутреннем строении эмулятора. Задаётся набор событий, происходящих в эмуляторе — к примеру, начало трансляции участка кода, а по наступлению события инструмент подсчитывает статистику потребления памяти.

Профиль позволяет выяснить, какие структуры эмулятора потребляют больше всего памяти. Оказывается, что наибольшую долю памяти занимают три структуры: кэш транслированного кода, хра-

нилище мета-информации о трансляции, хэш-таблица трансляций, которая позволяет быстро найти трансляцию по гостевому адресу. Каждая структура находится в выделенном для неё участке памяти. Запрашивая трансляцию, эмулятор обращается к трём страницам: чтобы найти адрес в хэш-таблице, обратиться к кэшу транслированного кода, получить мета-информацию о трансляции.

В [3] описывалось сокращение объёма мета-информации, хранимой вместе с кэшем, для ускорения трансляции. Мы, напротив, разместим их в одном участке памяти, чтобы вместе с записью хэш-таблицы располагался код и информация о нём. Предположительно, это сократит затраты по памяти и повысит локальность доступов.

Действительно, потребление памяти сократилось, однако эмулятор не ускорился, а замедлился вдвое: трансляции стали чаще вытесняться из кэша. Максимальный размер участка, содержащего кэш, ограничен максимальным смещением в инструкции относительного перехода. Мета-информация занимает место, которое могла бы занять трансляция, тем самым уменьшая фактический размер кэша. Оптимизация неприменима на практике.

Теперь рассмотрим ленивую инициализацию. На основе профилей найдём те структуры, которые инициализировались, но более не использовались или перезаписывались. К примеру, эмулятор выделил место под массив структур, из которых большая часть оказалась невостребованной. Тогда будем инициализировать структуры, когда они начинают реально использоваться. Чтобы уменьшить накладные расходы, будем применять оптимизацию, только когда она сократит число используемых страниц — так, на той же странице не должно быть другой часто используемой структуры.

В результате применения ленивой инициализации, время работы на тестах «холодного» кода сократилось на 2%, а потребление памяти — на 5%. Время инициализации эмулятора сократилось вдвое.

Литература

1. Rokicki, Simon, Erven Rohou, and Steven Derrien. "Hardware-accelerated dynamic binary translation."Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. IEEE, 2017.
2. Henning, John L. "SPEC CPU2006 benchmark descriptions."ACM SIGARCH Computer Architecture News 34.4 (2006): 1-17.
3. Baiocchi, José A., et al. "Reducing pressure in bounded DBT code caches."Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems.2008.