Кафедра системого программирования СПбГУ

Бакалаврская работа

Разработка профайлера для процессоров ARC

Автор:
Ножкин Илья Игоревич, 16.Б09-мм
Научный рукводитель:
д.ф.-м.н., проф. Терехов А.Н.

18.06.2020

Текст:
Здравствуйте, меня зовут Илья и я расскажу о своем проекте, посвященном разработке профайлера для процессоров ARC.

Профайлинг

Профайлинг — это динамический анализ программы с целью измерения характеристик ее работы.

Например, подсчет:

- Длительности выполнения процедур
- Объема потребляемой памяти
- Событий, связанных с оборудованием (промахи кэша, ошибки в предсказании переходов и т.д.)

Текст:
Профайлинг — это измерение характеристик исполнения программы непосредственно во время работы с целью поиска наиболее затратных участков и выяснения причин неэффективного их выполнения. В качестве характеристик может выступать время выполнения функций, потребление памяти или некоторые специфичные для оборудования события, такие как, например, промахи кэша.

Процессоры ARC

- Разрабатываются компанией Synopsys
- Являются широко конфигурируемыми, почти все параметры могут сильно варьироваться
- Применяются во встраиваемых системах различных типов — от интернета вещей до автоматического управления автомобилем

Текст:
Ядра ARC, в свою очередь, это ядра, разрабатываемые компанией Synopsys и их основной особенностью является возможность включать в состав процессора различный набор модулей, уникальный для каждой решаемой задачи. Применяются они во встраиваемых системах различной сложности — от микроконтроллеров до систем, осуществляющих обработку видео в реальном времени при автоматическом управлении автомобилем.

Постановка задачи

Цель: Разработать инструмент для измерения характеристик исполнения программ на встраиваемых системах с процессорами ARC

Задачи:

- Выполнить обзор существующих методов профайлинга и изучить возможности для профилирования, предоставляемые процессорами ARC
- Рассмотреть сценарии использования профайлера пользователем
- Разработать и реализовать алгоритмы выполнения выделенных сценариев
- Протестировать полученное решение и оценить точность измерений

Текст:
Таким образом, целью данной работы было создание инструмента для измерения характеристик исполнения программ для встраиваемых систем на базе процессоров ARC. В задачи работы входило: провести обзор существующих техник профайлинга и оценить их применимость в случае встраиваемых систем; изучить возможности для разработки профайлера, предоставляемые процессорами ARC; рассмотреть ожидаемые сценарии использования профайлера; а также, непосредственно реализовать решение и провести замер точности.

Известные методы профилирования

Метод	Преимущества	Недостатки	Примеры
Инструментирование	Аккуратность	Замедление, потребление памяти	gprof, Orbit
Сэмплирование	Минимальное вмешательство	Неаккуратность	gprof, OProfile
Симуляция	Информативность	Существенное замедление	Callgrind

Текст:

Существует набор классических техник для реализации профайлера. Первая из них — инструментирование, которую во-многом используют gprof и Google Orbit. Она заключается во внедрении в код программы специальных измеряющих последовательностей инструкций. Преимуществом данного подхода является аккуратность, поскольку ни один участок кода не будет пропущен и статистика будет наиболее полной. Однако внедрение дополнительных инструкций приведет к замедлению работы, а также будет необходимо сохранять необработанные результаты в память. Другой подход — это сэплирование. По нему пошли такие профайлеры, как OProfile и Intel VTune, в том числе его использует и gprof. Он основан на периодической остановке выполнения и сборе информации, накопленной между остановками. Главное его преимущество — минимальное вмешательство в программу и поток ее выполнения, однако подробности могут быть упущены. Еще один известный подход заключается в запуске программы на виртуальной машине и перехвате всех важных событий. Так поступают, например, Callgrind и симуляторы. Это позволяет получить максимум информации, однако приводит к существенному замедлению работы. Однако все эти профайлеры предназначены для работы на классических системах, в случае же встраиваемых систем есть некоторые особенности и вызываемые ими трудности.

Особенности встраиваемых систем

Предельное использование ресурсов

Маленький объем памяти

Часто медленный отладочный канал

Чувствительны к вмешательствам в код

Сложно хранить результаты локально

Дорого производить остановку

Текст:
Во-первых, при разработке системы стараются сделать ее максимально дешевой, маленькой и энергоэффективной из-за чего выжимают все имеющиеся ресурсы из оборудования. Из-за этого встраиваемые системы чувствительны к вмешательствам в код программы и поток выполнения, что затрудняет полноценное инструментирование. Во-вторых, у них часто небольшой объем оперативной памяти и при длительной работе сложно сохранять необработанные результаты профайлинга. Это затрудняет подход, предложенный в gprof. В-третьих, взаимодействие разработчика со встраиваемыми системами идет по внешнему медленному каналу, что затрудняет остановки, используемые в сэмплирующем профайлинге. Однако, именно из-за первого пункта профайлинг так важен для встраиваемых систем и поэтому эти проблемы нужно решать.

Задействованные средства, предоставляемые ARC

Счетчики производительности:

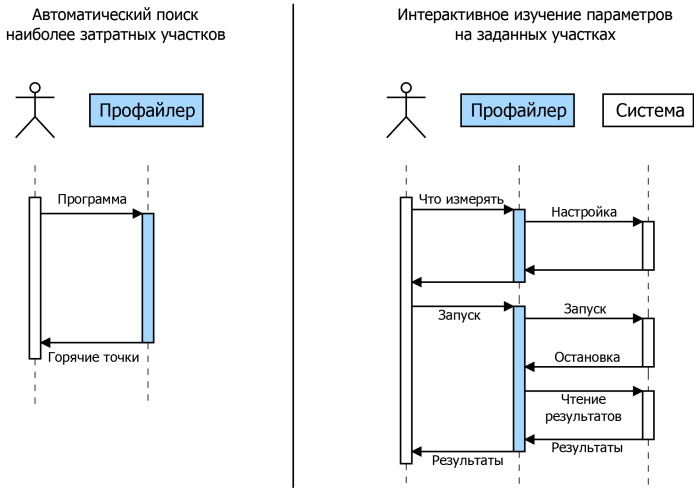
- Раздельно конфигурируются на срабатывание при одном из условий, например, при чтении из памяти, при промахе кэша, каждый цикл
- Могут совместно конфигурироваться на срабатывание в диапазоне адресов

Отладочный интерфейс, позволяющий, в частности:

- Произвести остановку
- Читать регистры и писать в них
- Читать из памяти и писать в нее

Текст:
Основная идея их решения состоит в использовании встроенных в оборудование средств. В данной работе, как наиболее гибкие, были выбраны встроенные счетчики производительности. Они могут считать различные события. Включаются и отключаются через запись в регистры и могут считать в заданном диапазоне. Для конфигурирования счетчиков во время профилирования используется отладочный интерфейс.

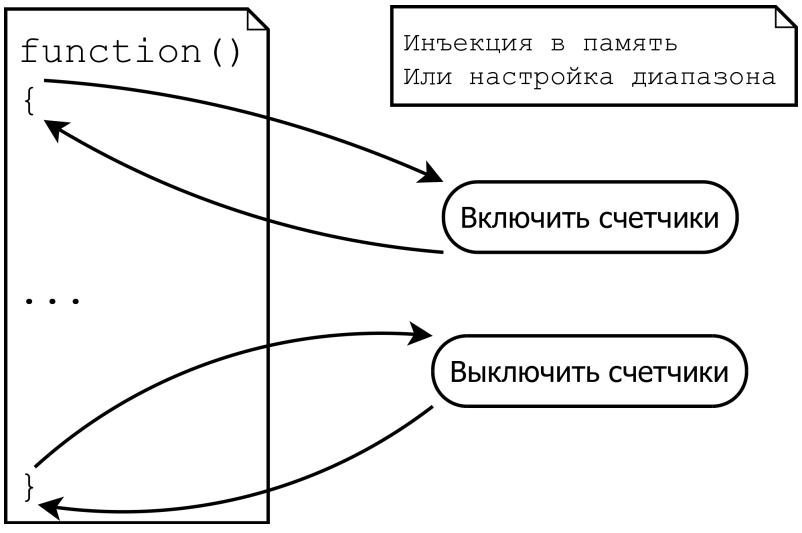
Сценарии использования



Текст:

Рассмотрим сценарии использования профайлера. Во-первых, классический сценарий — автоматический поиск горячих точек. Однако более важен и несколько иной сценарий, когда пользователь примерно знает, на каких именно участках программа работает неэффективно, и хочет уточнить результат и узнать, почему именно он такой. В данном случае важна интерактивность и возможность сообщения профайлеру пожеланий пользователя, а также минимальное вмешательство в выполнение программы со стороны профайлера.

Применение счетчиков для анализа



Текст:
Чтобы проанализировать участок кода, нужно включать счетчики при входе в него и отключать при выходе на протяжении всего времени измерений. В данной работе реализованы два метода такой активации: внедрение кода, управляющего счетчиками, в память работающего процесса и настройка диапазона счетчиков.

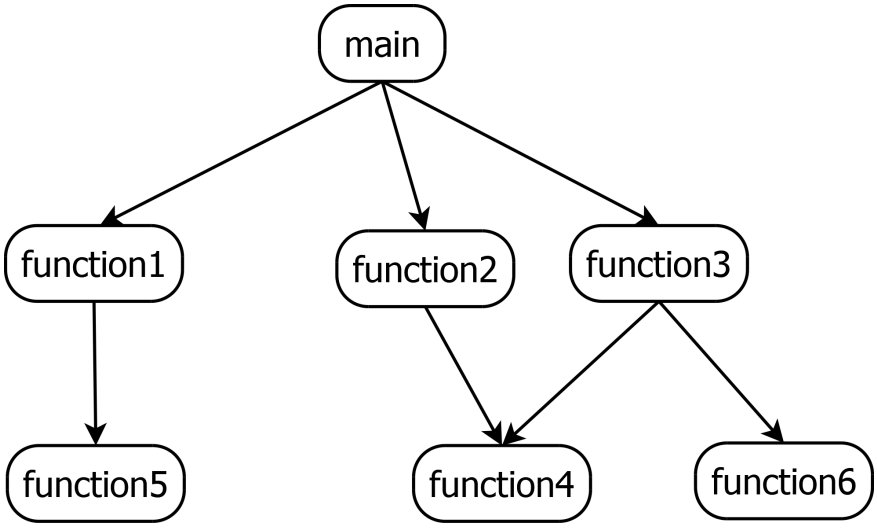
Характеристики методов активации

Оба метода позволяют сконфигурировать программу один раз и далее не вмешиваться в ее выполнение

	Задание диапазона	Инъекции
Потребление памяти	Отсутствует	Только хранение обработчиков
Влияние на точность	Отсутствует	Несколько дополнительных инструкций, единоразовая инвалидация части кэша
Ограничения	Один диапазон за запуск	Не всегда есть подходящее место для 2-х инъекций

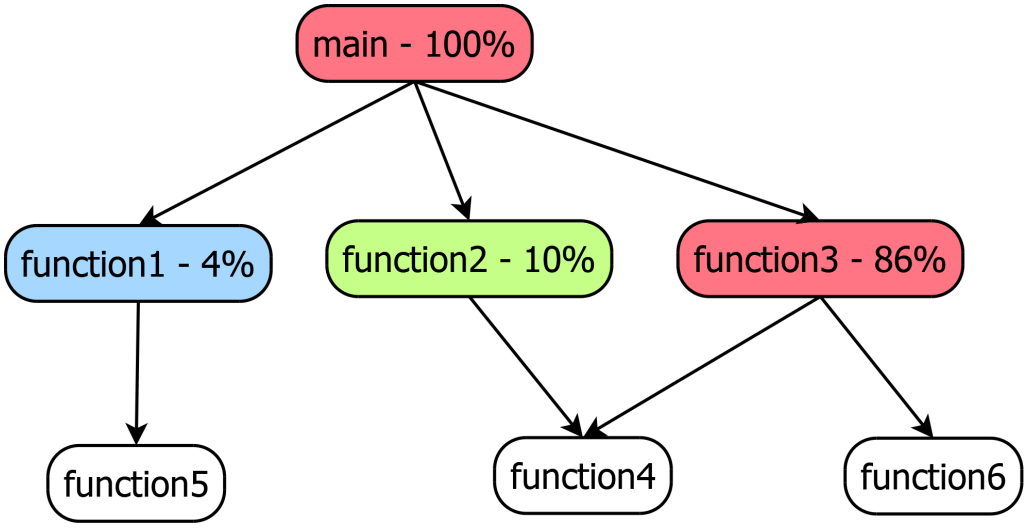
Текст:
Оба этих метода позволяют сконфигурировать программу единожды и далее не вмешиваться в ее выполнение до окончания анализа, что реализует интерактивный сценарий исследования программы. При этом, задание диапазона не вносит абсолютно никаких дополнительных расходов, однако позволяет проверить не более одного диапазона за раз. Инъекции же позволяют проверить за раз столько диапазонов, сколько имеется в распоряжении счетчиков, однако они требуют небольшого количества памяти для внедренного кода и вносят небольшие погрешности в измерения. Кроме того, не все функции могут быть проинструментированы. Теперь, чтобы провести поиск горячих точек, можно прибегнуть к наивному подходу: разбить множество функций на подмножества размером с количество счетчиков и переконфигурировать счетчики до тех пор, пока все функции не будут проанализированы. Однако, есть более оптимальный алгоритм.

Поиск затратных участков



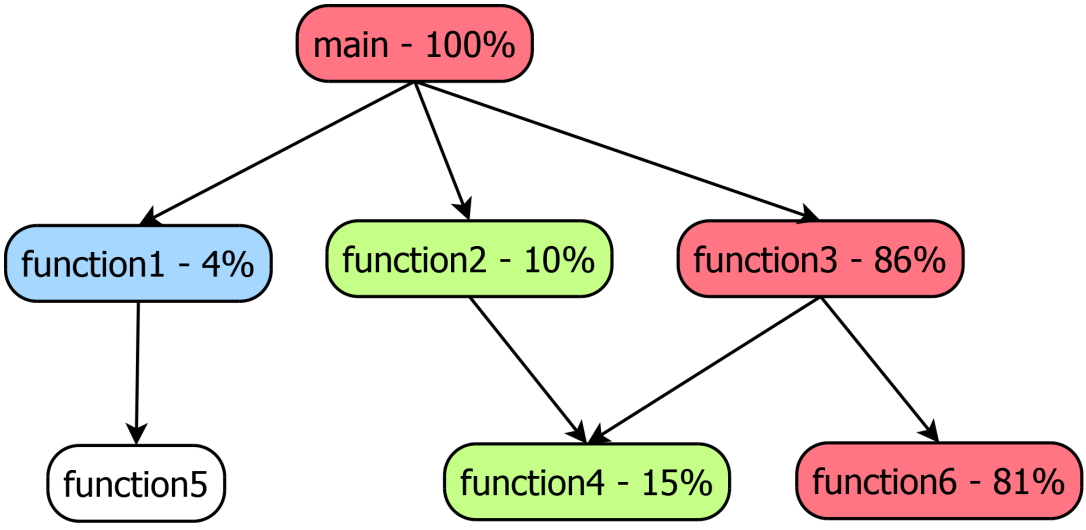
Текст:
Сначала извлекается граф вызовов и выполняется топологическая сортировка.

Поиск затратных участков



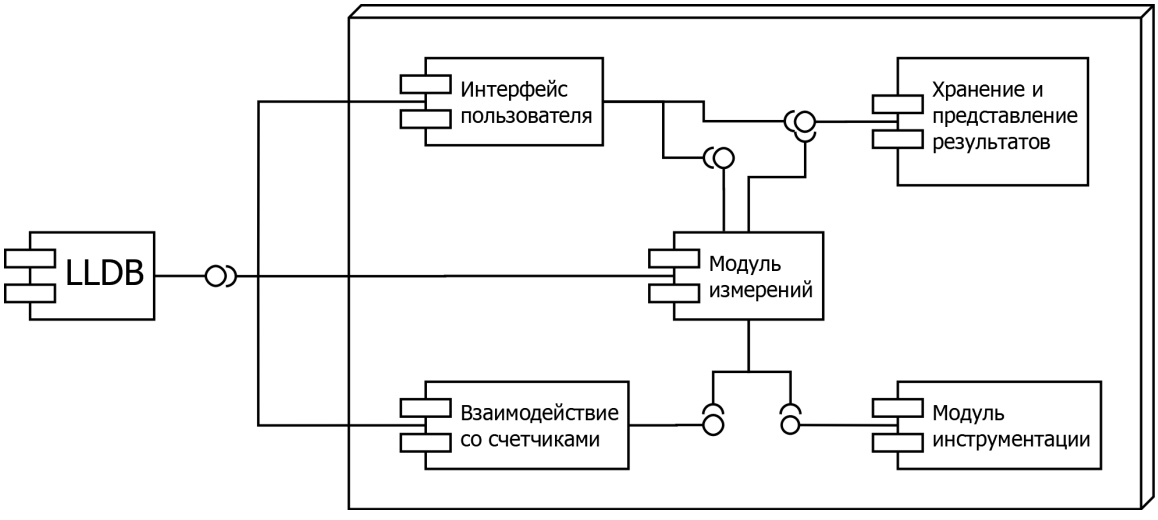
Текст:
Далее анализируются начальные функции. С использованием их результатов рассчитывается верхняя граница значений метрик у их потомков. Потомки, чья оценка мала, исключаются из анализа.

Поиск затратных участков



Текст:
Далее процесс продолжается для оставшихся функций. В итоге, наиболее затратные участки выделяются за меньшее число запусков, чем при полном переборе.

Реализация



Текст:

Сам профайлер реализован на Python в виде модуля, подключаемого к отладчику для процессоров ARC на базе LLDB. При этом, отладчик реализует управление запуском кода и взаимодействие с оборудованием, но никак не вмешивается в работу во время исполнения программы. Профайлер состоит из нескольких компонентов, из которых базовые — это управление счетчиками и инструментирование. Инструментирующий модуль также содержит небольшой ассемблер для генерации кода в реальном времени. Модуль измерений управляет процессом анализа и порождает первичные результаты, которые обрабатываются модулем представления результатов. Измерения и просмотр результатов выведены в интерфейс пользователя, представленный в виде команд для внутреннего интерпретатора LLDB.

Апробация

Тест	Длительность теста (в циклах)	Интервал абсолютных погрешностей по функциям (в циклах)	Интервал относительных погрешностей по функциям (%)	Кол-во вызовов в тесте	Дополнительные расходы на вызов (в циклах)	Сумм. Дополнительные расходы (%)	Сумм. Дополнительные расходы gprof (%)
aha_mont64	34734	13-859	0.2-2.47	21	40.9	2.47	22.69
crc32	194253	14-81	0.01-0.04	2	40.5	0.04	113.17
cubic	201084	18-18	0.01-0.01	1	18.0	0.01	21.51
edn	553189	4-253	0.0-0.84	9	28.1	0.05	0.35
huffbench	907810	818-3277	0.35-3.39	96	34.1	0.36	1.07
matmult_int	1253556	61-7412	0.0-5.81	803	9.2	0.01	0.08
minver	23416	27-379	0.33-2.57	11	34.5	0.43	12.1
nettle_aes	138350	12-659	0.02-1.76	12	54.9	0.48	2.47
nettle_sha256	14077	13-499	0.34-12.05	9	55.4	3.54	19.44
slre	92523	77-45949	13.44-65.09	1168	39.3	49.66	308.34
st	701853	9-29018	0.05-6.48	613	47.3	4.13	21.32
statemate	4403	110-239	2.86-5.43	5	47.8	5.43	69.54
ud	17212	19-70	0.15-0.41	2	35.0	0.41	2.53
zlib	234445	1-298	0.0-4.83	28	10.6	0.04	-

Текст:

Оценка точности и дополнительных расходов проводилась следующим образом. Был собран бенчмарк для встраиваемых систем Embench, а также простая программа, использующая библиотеку zlib. Далее, были проанализированы функции, входящие в тесты, с использованием xCAM — симулятора процессора на уровне логических вентилей. Также были получены результаты представленного решения и рассчитано отклонение от результатов xCAM. Полученные оценки приведены в таблице, изображенной на слайде. В среднем, относительная погрешность в циклах колеблется около 3-х процентов, однако существенно увеличивается на часто вызываемых маленьких функциях. Дополнительные расходы при этом близки к константным на один вызов измеряемой функции и в большинстве случаев существенно меньше длительности выполнения самой функции. В сумме, дополнительные расходы на инструментирование всех функций в тесте находятся в интервале от 0.01% до 5%. Что, например, в несколько раз меньше, чем аналогичный показатель для профайлера gprof.

Сравнение с существующими инструментами

	xCAM	nSIM	gprof	Разработанное решение
Используемый метод	Симуляция на уровне логических вентилей	Симуляция семантики инструкций	Инструментация + сэмплирование	Инструментация + счетчики производительности
Может использоваться на реальном оборудовании	Нет	Нет	Да	Да
Требует предварительной подготовки	Нет	Нет	Да	Нет
Может анализировать всю программу за один проход	Да	Да	Да	Нет
Замедление	На порядки	В разы	На 1-20% в среднем В 4 раза максимум	На 0.01-5% в среднем В 1.5 раза максимум
Точность	Абсолютная	Результаты по функциям коррелируют с реальными	Результаты по функциям коррелируют с реальными при длительном профилировании	Погрешность $\sim 3\%$ относительно циклов

Текст:

Для итогового сравнения были выбраны следующие три существующих инструмента для профилирования: xCAM (симулятор на уровне логических вентилей), nSIM (симулятор на уровне инструкций) и gprof. В сравнении с ними представленное решение отличается возможностью использования на реальном оборудовании, что невозможно при использовании симуляторов. Оно не требует предварительной подготовки программы в отличие от gprof, который требует скомпилировать программу со специальным флагом. Однако данное решение не может анализировать программу целиком. При этом, оно существенно меньше замедляет выполнение по сравнению с остальными инструментами и позволяет достаточно точно измерить реальное количество событий, происходящих во время выполнения того или иного участка. gprof и nSIM, в свою очередь, могут использоваться для быстрого поиска горячих точек, но не гарантируют точный результат в циклах, а данное решение занимает нишу интерактивного уточняющего инструмента.

Результаты

- Выполнен обзор существующих техник профайлинга. В качестве основы для реализации выбрано использование инструментации и счетчиков производительности
- Выделены два основных сценария использования профайлера
- Разработаны алгоритмы: анализа отдельных участков и поиска горячих точек в графе вызовов. Решение реализовано в качестве расширения для существующего отладчика
- Решение протестировано. Точность оценена на примерах zlib и Embench, погрешность измерения циклов составила в среднем 3%, однако она существенно увеличивается при анализе коротких функций

Текст:
Таким образом, в ходе работы были получены следующие результаты: рассмотрены существующие техники профайлинга, из них выбрана комбинация динамической инструментации и счетчиков производительности. Выделены два сценария использования. Разработаны и реализованы алгоритмы выполнения сценариев и пользовательский интерфейс. Решение протестировано, проведены его апробация и сравнение.

Вопросы научного руководителя и рецензента

1. В тексте упомянуто сравнение дополнительных затрат, вносимых описанным решением, и возникающих при использовании gprof, однако было бы лучше также привести более подробные результаты измерений в отдельной таблице и в целом добавить практическое сравнение с gprof на рассмотренном бенчмарке.
2. Интересно узнать о возможностях интеграции текущего решения с существующими утилитами и средствами (RTT, xCAM, nSIM) для обмена информацией между ними и расширения возможностей для анализа.

Ответы:

1. Дополнительные затраты, вносимые gprof, добавлены как отдельный столбец на слайде с оценкой точности.

2. xCAM и nSIM обладают заведомо более широкими возможностями для анализа и при работе с ними предложенное решение не даст дополнительных преимуществ. Однако можно использовать их результаты для более быстрого извлечения графа вызовов и для первичного поиска наиболее затратных участков программы, а далее использовать предложенное решение на реальном оборудовании. Наличие же Real-Time Tracing решит многие проблемы предложенного подхода. В частности, позволит реализовать извлечение графа вызовов в реальном времени, а также даст возможность анализировать все функции программы за один проход посредством отправки значений счетчиков в вывод трассировщика вместо хранения их в ограниченном количестве регистров счетчиков.

Вопросы научного руководителя и рецензента

1. В работе используется подход на основе динамической инструментации кода программы. Для полной оценки преимущества данного подхода над подходом, основанном на остановке выполнения программы и конфигурированием/считыванием данных Performance Counters регистров, следовало бы измерить производительность обоих методов на различных сценариях и привести таблицу с результатами сравнения.
2. Сбор статистики потребления ресурсов функциями для циклических программ предполагает, что пользователь предоставляет профайлеру машинный адрес инструкции, на которой можно безопасно произвести остановку. Однако, пользователю, разрабатывающему программное обеспечение на языке высокого уровня, было бы удобнее указывать строку кода или имя функции, на которых можно остановить процесс (подобно параметрам точек останова). Такое улучшение может быть выполнено на следующей стадии развития проекта.

Ответы:

1. Подход с остановкой и конфигурированием/считыванием счетчиков через отладочный интерфейс рассматривался на ранних этапах работы. Однако, он был исключен после измерений, которые показали задержку выполнения в 3-4 раза превосходящую таковую при динамической инструментации. Кроме того, остановка процессора приводит к сбросу конвеера, что также негативно влияет на результаты. В остальном, результаты были бы очень похожи.

2. Это хорошее замечание к пользовательскому интерфейсу и будет принято к сведению. Изначально данная возможность не была реализована из-за особенностей API LLDB и была временно отложена. Однако, принципиальных препятствий для этого нет.

Вопросы научного руководителя и рецензента

1. Извлечение графа вызовов производится не самым оптимальным способом. Тем не менее, альтернативный и более эффективный вариант упомянут, и также может быть реализован в дальнейшем развитии проекта.

Ответы:

1. Действительно, реализованный в работе способ является наивным. Однако, данный тип анализа не был главной целью работы и был реализован как легкий способ получения максимально достоверных данных для использования в дальнейших измерениях. Планируется обеспечить интеграцию с другими способами извлечения графа вызовов, таких как использование статического анализа, трассировки, симуляторов, профайлера gprof. В том числе, есть возможность использовать динамическую инструментацию для извлечения графа вызовов самим процессором и сохранения его в оперативную память.