Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б15-мм

Сбор статистики BPU в ОС на базе ядра Linux

Ефремов Алексей Александрович

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ст. преп. каф. ИАС, К. К. Смирнов

Консультант:
Инженер-исследователь, Лаборатория технологий программирования инфраструктурных решений СПбГУ, В. А. Кутуев

Санкт-Петербург
2024

Оглавление

Введение **3**

1. Постановка задачи **5**

2. Обзор **6**

2.1. Обзор проектов с удалённым запуском 6

2.2. Используемые технологии 7

3. Адаптация архитектуры **8**

3.1. Прошлая архитектура 8

3.2. Обновлённая архитектура 9

4. Запуск тестов на удалённой машине **10**

5. Параллелизм **11**

5.1. Background Buider 11

5.2. Параллельный запуск нескольких анализаторов 12

6. Расширение конфигурации нативного анализатора **14**

7. Апробация **15**

Заключение **16**

Список литературы **18**

Введение

С момента появления вычислительных систем всегда была необходимость повышения скорости выполнения программ. Одним из основных методов ускорения процессоров является конвейеризация инструкций, позволяющая одновременно обрабатывать несколько инструкций, разбивая их выполнение на отдельные этапы. Однако производительность этого метода не является постоянной. В частности, инструкции переходов (условные, безусловные, вызовы функций и возвраты из них) могут приводить к сбросу конвейера, вызывая простой отдельных компонентов процессора. Для устранения этой проблемы используется специализированный модуль — branch prediction unit (BPU). BPU пытается предсказать следующий исполняемый блок кода, чтобы минимизировать задержки процессора во время переходов.

Но эффективность различных реализаций BPU не является одинаковой [9, 10, 7]. Например, процессоры, произведенные компаниями с ограниченным опытом в их разработке, могут иметь более низкую производительность BPU по сравнению с передовыми процессорами от производителей, которые давно на этом специализируются [8, 3]. Изучение сценариев, в которых BPU конкретного процессора демонстрирует существенно худшие результаты по сравнению с BPU современных массовых процессоров, может способствовать улучшению архитектуры данного BPU посредством детального анализа таких сценариев.

Для выявления сценариев, подчеркивающих разницу в эффективности различных BPU, был выбран метод фаззинга. Главная идея метода заключается в многократном исполнении случайно сгенерированного кода для обнаружения сценариев, на которых BPU показывает аномально плохие результаты. На основе этого метода была начата разработка инструмента, состоящего из трёх частей:

- generate: генератор тестов на языке C;
- analyze: компонент, отвечающий за сборку тестов, их запуск и сбор данных со счетчиков процессора, хранящих информацию о работе

BPU;

- summarize: обработчик и визуализатор информации по запущенным тестам;

В рамках этой работы для данного инструмента была расширена функциональность этапа analyze: добавлен запуск тестов на удалённом компьютере, расширена конфигурация нативного анализатора и реализована возможность одновременного запуска нескольких анализаторов и параллельной сборки. Данная работа выполняется совместно с работой студента второго курса направления «Программная инженерия» Данила Слинчука, в рамках которой была сделана оставшаяся часть инструмента.

1. Постановка задачи

Целью работы является расширение функциональности этапа анализа тестов в проекте «Control hazard analyzer» [2]. Для её выполнения были поставлены следующие задачи:

1. произвести обзор реализации удалённого запуска в других проектах;
2. адаптировать архитектуру под удалённый анализатор;
3. реализовать запуск тестов на удалённой машине;
4. выполнить распараллеливание сборки тестов и их запуска для повышения производительности инструмента;
5. расширить возможности конфигурации нативного анализатора.

2. Обзор

2.1. Обзор проектов с удалённым запуском

В данном разделе рассматриваются фреймворки MPICH, Open MPI, Apache Hadoop, которые помогают организовать распределённые вычисления, для выбора подходящего способа запуска удалённой задачи и получения её результатов.

2.1.1. MPICH

MPICH [4] — это открытая высокопроизводительная реализация стандарта MPI (Message Passing Interface), разработанная для обеспечения масштабируемости и эффективности в параллельных вычислениях на суперкомпьютерах и кластерных системах.

Исследование исходного кода MPICH выявило, что удаленный запуск реализован посредством SSH. Для создания SSH-сессии MPICH использует функцию `execve()`, которая напрямую запускает SSH-клиент.

2.1.2. Open MPI

Open MPI [5] — это открытая реализация стандарта MPI, созданная для обеспечения гибкости и совместимости в параллельных вычислениях, активно поддерживаемая сообществом для использования на широком спектре вычислительных платформ.

Согласно документации по Open MPI, для удалённого запуска программ на нескольких компьютерах можно использовать различные инструменты удалённого управления. Однако SSH является классическим выбором, поскольку большинство документации и примеров ориентированы на его использование.

2.1.3. Apache Hadoop

Apache Hadoop [1]— это открытый фреймворк, разработанный для обработки и хранения больших объемов данных в распределенном ком-

пьютерном окружении, обеспечивая параллельную обработку данных на уровне кластера.

Для коммуникации между кластерами Apache Hadoop использует SSH. Кроме SSH в Apache Hadoop можно использовать другие механизмы аутентификации и управления, такие как Kerberos. Однако SSH остается популярным выбором благодаря своей простоте настройки, широкой доступности и надежности.

2.1.4. Вывод

Все рассмотренные в обзоре проекты используют SSH в качестве основного метода удаленного запуска целевых программ. Поэтому для разрабатываемого инструмента был также выбран SSH для выполнения тестов на удаленных машинах.

2.2. Используемые технологии

Для реализации SSH в разрабатываемом инструменте была выбрана библиотека Paramiko [6]. Этот выбор обусловлен тем, что для данной библиотеки нет особых требований со стороны проекта, кроме языка программирования, а Paramiko является ведущей библиотекой для реализации SSH-протокола в Python.

Библиотека Paramiko предоставляет широкий спектр функций для работы с SSH, включая аутентификацию, передачу файлов и выполнение команд на удаленных машинах. Она также хорошо документирована, что делает ее удобной для использования в академических и прикладных проектах.

3. Адаптация архитектуры

3.1. Прошлая архитектура

Для того чтобы получить значение счётчиков BPU, анализатор выполняет следующие шаги:

- подготавливает код теста к сборке, оборачивая его в специальные конструкции;
- собирает тест в исполняемый файл;
- запускает исполняемый файл и собирает статистику о его выполнении.

В предыдущей архитектуре было только два типа анализаторов: нативный анализатор и анализатор Gem5. Поскольку сборка тестов является общей задачей для обоих анализаторов, эта функциональность была выделена в отдельный класс `Builder`, который используется обоими анализаторами. Схематичное представление работы основного метода анализатора, `analyzer.analyze()`, в исходной архитектуре приведено на рисунке 1.

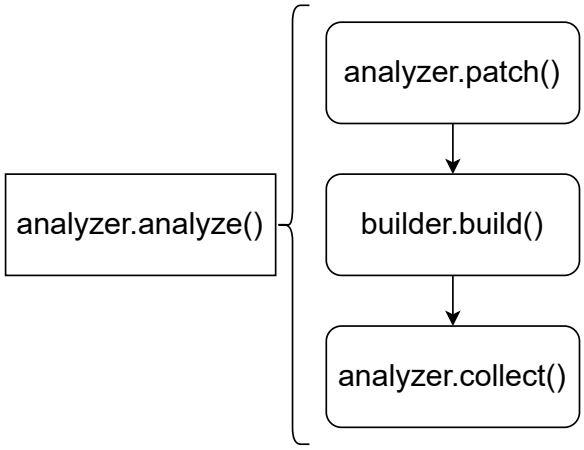

Рис. 1: Схема analyzer.analyze() для прежней архитектуры

3.2. Обновлённая архитектура

Предыдущая архитектура имела несколько недостатков, которые были устранены в новой архитектуре. Основная концепция новой архитектуры заключается в разделении обязанностей анализатора на отдельные классы. Это позволяет:

- упростить создание анализаторов на основе существующего кода;
- снизить количество дублируемого кода.

В результате архитектура анализаторов была пересмотрена следующим образом:

- код подготовки тестов был выделен в классы с интерфейсом `Patcher`;
- код запуска бинарных файлов и сбора статистики был выделен в классы с интерфейсом `Collector`.

Схематичное представление работы метода `analyze` для пересмотренной архитектуры приведено на рисунке 2.

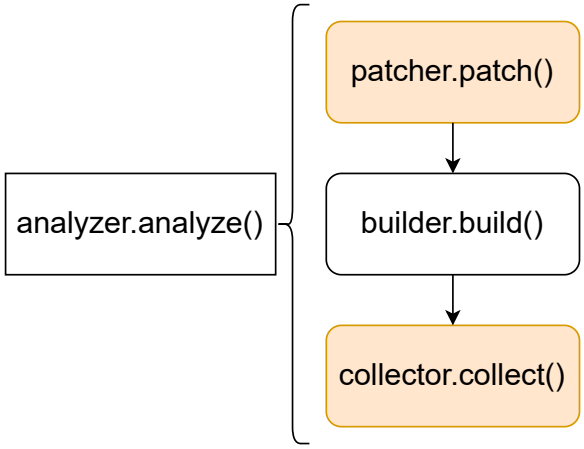

Рис. 2: Схема analyzer.analyze() для новой архитектуры

4. Запуск тестов на удалённой машине

Для удаленного анализатора был реализован только дополнительный `Collector` — `SSH Collector`. Создание отдельного `Patcher` не требовалось, поскольку для подготовки тестов можно использовать реализацию `Patcher` из нативного анализатора. Для `Builder` этого анализатора можно использовать тот же, что и для остальных анализаторов, так как он уже поддерживает кросс-компиляцию и позволяет собирать файлы под процессоры другой архитектуры.

Настройка `SSH Collector` требует указания адреса сервера, имени пользователя, пароля или ключа в зависимости от выбранного метода аутентификации. Указанный пользователь должен иметь права суперпользователя для выставления приоритета, планировщика процесса и получения значений счетчиков с процессора. В ходе своей работы `SSH Collector` передаёт целевой исполняемый файл теста на сервер по SFTP, затем запускает по SSH и собирает статистику по его выполнению (см. Рисунок 3).

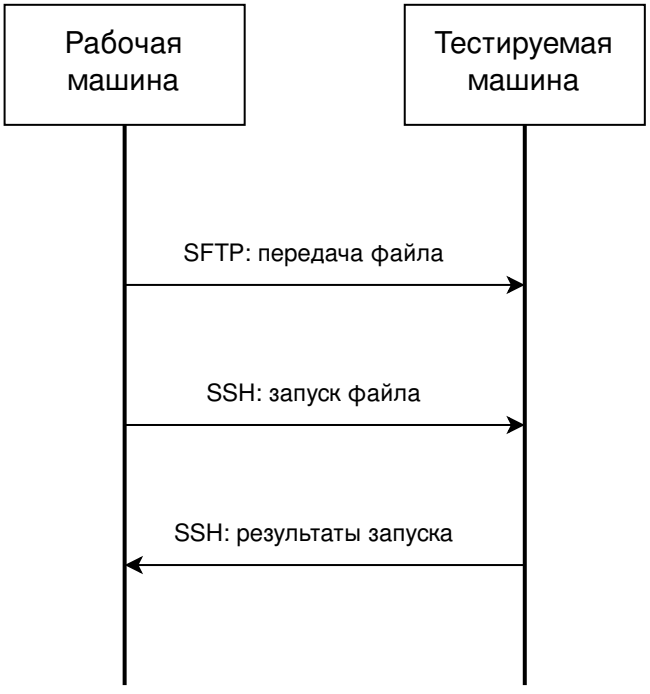

Рис. 3: Диаграмма работы SSH Collector

5. Параллелизм

Хотя на первый взгляд скорость работы инструмента может показаться несущественным параметром, учитывая, что для оценки эффективности исследуемого процессора достаточно одного запуска. Инструмент имеет еще один важный сценарий использования, при котором скорость работы важна — генерацию новых тестов.

Генерация тестов для бенчмарка является время затратной задачей, требующей большого количества сгенерированных тестов и их результатов для выделения показательных. Кроме того, планируется внедрить в инструмент возможность генерации тестов с обратной связью, при которой результаты выполнения теста будут влиять на последующую генерацию. В этом случае увеличение скорости будет приводить к росту количества показательных тестов.

Для увелечения скорости работы было решено реализовать:

- класс Builder, который позволяет параллельно сборке тестов, запускать уже собранные;
- параллельный запуск анализаторов.

5.1. Background Buider

Прежняя реализация `Builder` была последовательной. Она сначала собирала все файлы, а затем передавала управление в SSH Collector. При удаленном тестировании это приводило к простою тестируемой машины во время сборки и рабочей машины во время работы теста на удалённой (см. Рисунок 4).

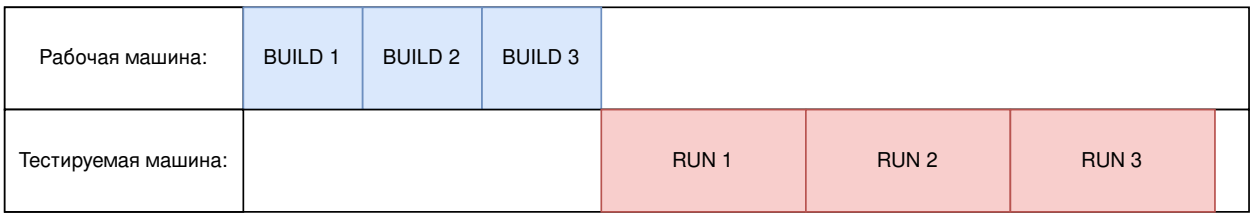

Рис. 4: График выполнения при последовательной реализации Builder

Для минимизации простоя была создана параллельная реализация `Builder` — `Background Builder`. `Background Builder` работает в отдельном потоке и по мере завершения сборки каждого файла добавляет его путь в очередь. `Controller` отслеживает очередь и извлекает из нее файлы для запуска тестов. Таким образом, сборка и запуск тестов выполняются почти параллельно (см. Рисунок 5).

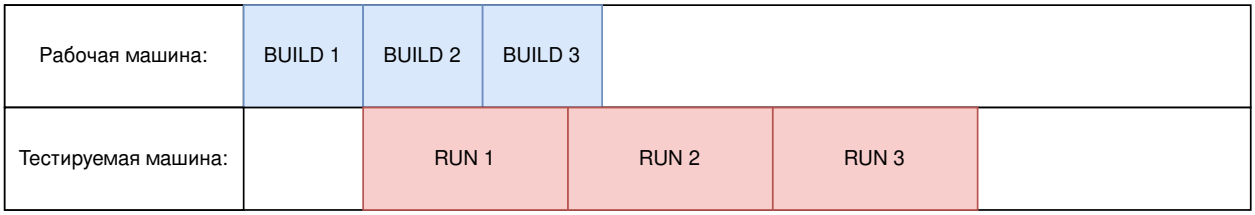

Рис. 5: График выполнения при параллельной реализации Builder

5.2. Параллельный запуск нескольких анализаторов

Драйвер инструмента `Aggregate` ранее поддерживал только последовательное выполнение анализаторов. Однако при использовании удаленных анализаторов это привело бы к простою всех тестируемых машин (см. Рисунок 6).

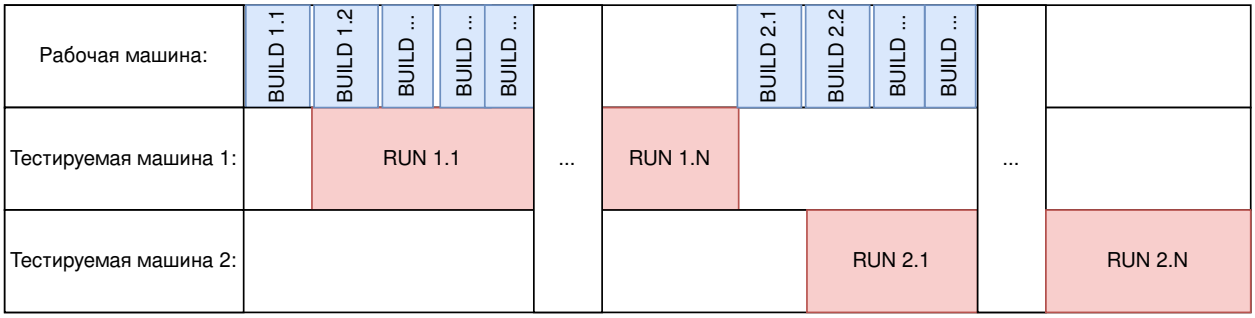

Рис. 6: График выполнения при последовательном запуске анализаторов

Драйвер был дополнен возможностью запуска всех анализаторов одновременно, что устранило простой удаленных машин (см. Рисунок 7). Это достигается путем запуска каждого анализатора в отдельном потоке. По завершении работы анализаторы передают свои данные в оче-

редь. После завершения работы всех анализаторов данные из очереди обрабатываются и передаются на вход `Summarize`.

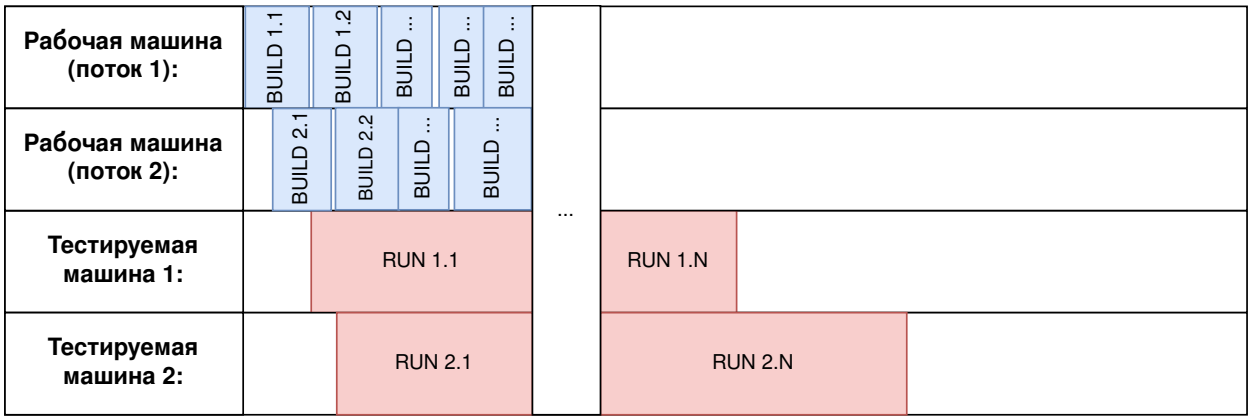

Рис. 7: График выполнения при параллельном запуске анализаторов

6. Расширение конфигурации нативного анализатора

В нативном анализаторе на этапе подготовки тестов код оборачивается в конструкцию, использующую системный вызов `perf_event_open` для получения необходимых результатов. Существует набор классических событий для этого системного вызова, которые ядро гарантирует предоставить по определенному номеру. Эти события включают счетчики, представляющие интерес для анализа. Однако процессор может не поддерживать подсчет определенного классического события, но быть способным подсчитывать другое событие, по которому можно вычислить значение нужного. К примеру, процессор «ARM Cortex-A72» для «Raspberry Pi 4b» не может собирать информацию по общему числу переходов, но может считать число угаданных переходов, событие не являющееся классическим. Такое дополнительное событие будет иметь собственный номер, уникальный для данного процессора. Кроме того, не все процессоры поддерживают все возможности `perf_event_open`.

Для решения этих проблем реализована конфигурация `perf_event_open`. Она позволяет считывать дополнительные процессорозависимые события и отключать определенные возможности системного вызова. В параметрах анализатора пользователь может выбрать один из готовых заголовочных файлов, настраивающих `perf_event_open` для конкретных процессоров. Допускается создание собственного заголовочного файла, если для исследуемого процессора нет готового и классические настройки не подходят.

Поскольку подготовка тестов для нативного и удаленного анализаторов совпадает, все возможности конфигурирования доступны и для удаленного анализатора.

7. Апробация

Для апробации была проведена оценка различных BPU с использованием новых функциональных возможностей инструмента. Из сотен сгенерированных тестов были отобраны семьдесят пять, в которых BPU демонстрировали нестандартное поведение в работе. Эти тесты были объединены в бенчмарк и запущены на процессорах различных архитектур и на симуляторе Gem5. Бенчмарк и результаты его запуска для каждого процессора и модели BPU загружены в отдельную ветку проекта на GitHub[1]. Более подробно о проведённой оценке можно узнать в работе Данила Слинчука.

[1]Ветка с результатами бенчмарка:
https://github.com/osogi/control-hazard-analyzer/tree/syrcose (дата обращения: 25 мая 2024 г.)

Заключение

В ходе этой работы были выполнены все задачи и получены следующие результаты:

1. проведён обзор реализации удалённого запуск в других проектах: MPICH, Open MPI и Apache Hadoop;
2. переработана архитектура анализаторов, разделена на более мелкие элементы;
3. реализован запуск тестов на удалённой машине с использованием ssh;
4. реализован параллельный запуск нескольких анализаторов и параллельная сборка тестов;
5. расширенны возможности конфигурации нативного анализатора.

Результаты совместной работы с Данилом Слинчуком были представлены на:

- конференции «Современные технологии в теории и практике программирования», опубликованы в сборнике входящем в РИНЦ[2];
- конференции SYRCoSE, сборник пока не опубликован.

Репозиторий инструмента на GitHub (пользователь: osogi): https://github.com/osogi/control-hazard-analyzer (дата обращения: 25 мая 2024 г.).

Дальнейшее развитие инструмента

- Запустить на компьютерах с различными процессорами при помощи инструмента существующе бенчмарки, например: Coremark, Dhrystone.

[2]Ссылка на сборник: https://www.elibrary.ru/item.asp?id=66237522 (дата обращения: 25 мая 2024 г.

- Добавить анализатор для компьютеров без операционной системы.

Список литературы

[1] Apache Hadoop. — официальный сайт. — URL: https://hadoop.apache.org/ (дата обращения: 25 мая 2024 г.).

[2] Control hazard analyzer. — GitHub. — URL: https://github.com/osogi/control-hazard-analyzer (дата обращения: 25 мая 2024 г.).

[3] Fast sort on cpus, gpus and intel mic architectures / Nadathur Satish, Changkyu Kim, Jatin Chhugani et al. // Intel Labs. — 2010. — P. 77–80.

[4] MPICH. — официальный сайт. — URL: https://www.mpich.org/ (дата обращения: 25 мая 2024 г.).

[5] Open MPI. — официальный сайт. — URL: https://www.open-mpi.org/ (дата обращения: 25 мая 2024 г.).

[6] Paramiko. — официальный сайт. — URL: https://www.paramiko.org/ (дата обращения: 25 мая 2024 г.).

[7] Quadri Syed Ali Imran, Jahangir Mohd Ziauddin. Design, Implementation and Performance Comparison of Different Branch Predictors on Pipelined-CPU // 2017 International Conference on Computer, Electrical & Communication Engineering (ICCECE) / IEEE. — 2017. — P. 1–7.

[8] Roberts-Hoffman Katie, Hegde Pawankumar. ARM cortex-a8 vs. intel atom: Architectural and benchmark comparisons // Dallas: University of Texas at Dallas. — 2009. — Vol. 5.

[9] Yeh Tse-Yu, Patt Yale N. A comparison of dynamic branch predictors that use two levels of branch history // Proceedings of the 20th annual international symposium on computer architecture. — 1993. — P. 257–266.

[10] Youssif A, Ismail N, Torkey F. Comparison of branch prediction schemes for superscalar processors iceec 2004 // Electrical, Electronic

and Computer Engineering, 2004. ICEEC'04. 2004 International Conference on. — 2004. — P. 257–260.