Санкт-Петербургский государственный университет

Печенев Данила Евгеньевич

Выпускная квалификационная работа

Разработка метода и инструментария для выделения характерных интервалов исполнения приложения

Уровень образования: бакалавриат

Направление *09.03.04 «Программная инженерия»*

Основная образовательная программа *СВ.5080.2021 «Программная инженерия»*

Научный руководитель:
старший преподаватель, к.т.н Ю. В. Литвинов

Консультант:
ведущий инженер-программист, ООО «КНС ГРУПП» Д. В. Донцов

Рецензент:
инженер-программист, ООО «КНС ГРУПП» О. В. Судаков

Санкт-Петербург
2025

Saint Petersburg State University

Danila Pechenev

Bachelor's Thesis

Development of a method and tools for identifying characteristic intervals of application execution

Education level: bachelor

Speciality *09.03.04 "Software Engineering"*

Programme *CB.5080.2021 "Software Engineering"*

Scientific supervisor:
C.Sc., senior lecturer Y. V. Litvinov

Consultant:
lead software engineer at KNS GROUP LLC D. V. Dontsov

Reviewer:
software engineer at KNS GROUP LLC O. V. Sudakov

Saint Petersburg
2025

Оглавление

Введение

Одним из направлений деятельности компании YADRO является разработка микропроцессоров и инструментов на базе открытой расширяемой архитектуры RISC-V [22]. В рамках этой деятельности имеется необходимость в тестировании производительности проектируемого процессора на приложениях, соответствующих реальным сценариям его использования.

Запуск комплексных индустриальных тестов, *бенчмарков*, производится на программных и аппаратных симуляторах с детальной имитацией процессорного конвейера. Скорость их работы многократно уступает реальному процессору, поэтому симуляция полного исполнения каждого приложения может занимать многие месяцы. Проблема усугубляется тем, что для выбора оптимального набора ключевых параметров архитектуры процессора инженерам требуется анализировать результаты симуляции множества приложений на целом ряде архитектурных конфигураций.

Для эффективности оценки ключевых метрик исполнения приложения используется идея о том, что процесс выполнения приложения состоит из *фаз*, в рамках которых оно ведет себя схожим образом, демонстрируя приблизительно одинаковые показатели производительности. Выделив из каждой фазы по одному интервалу и наделив его определенным весом, мы получаем возможность оценить метрики производительности всего приложения без симуляции его полного исполнения, то есть достаточно запустить на симуляторе *характерные интервалы* исполнения приложения и вычислить взвешенную сумму измеренных метрик.

Чтобы выделение характерных интервалов стало возможным, исполнение исследуемого бенчмарка делится на равные по количеству инструкций интервалы, после чего для каждого из них на симуляторе без имитации конвейера QEMU [33] собираются некоторые статистические данные. Они подаются на вход инструмента SimPoint [1], который выполняет кластеризацию интервалов и выделяет наиболее характер-

ные из них с заданием им весов.

Инструмент является открытым, однако его лицензия [17] не позволяет включать инструмент в проприетарные пакеты компании YADRO. Помимо этого, метод выделения характерных интервалов, описанный авторами инструмента в оригинальной статье [4], зачастую демонстрирует неудовлетворительное качество приближения оценки целевой метрики к её реальному значению, и у инженеров YADRO есть потребность в повышении точности этого метода.

1. Постановка задачи

Целью работы является разработка метода и инструментария для выделения характерных интервалов исполнения приложения для быстрой оценки качества архитектурных изменений в проектируемом процессоре.

Для достижения цели были поставлены следующие задачи.

1. Провести обзор существующего метода нахождения характерных интервалов.

2. Разработать новый метод нахождения характерных интервалов на основе существующего, включая выбор более подходящих для задачи способа характеризации интервалов и алгоритма кластеризации.

3. Разработать программную реализацию предложенного метода.

4. Провести экспериментальное исследование для сравнения существующего и нового методов нахождения характерных интервалов.

2. Обзор

В этом разделе приведено детальное описание существующего подхода к характеризации интервалов исполнения приложения и алгоритма работы инструмента SimPoint, рассмотрены особенности применения инструмента в компании YADRO.

2.1. Определения

- Интервал — это часть непрерывного исполнения приложения. В рамках этой ВКР интервалы всегда являются последовательными, непересекающимися и равными по количеству инструкций.

- Фазой является множество интервалов исполнения приложения со схожим поведением.

- Базовый блок — это последовательность инструкций процессора, имеющая ровно одну точку входа, ровно одну точку выхода и не содержащая инструкций передачи управления ранее точки выхода.

- QEMU [33] является известным программным эмулятором процессора без имитации конвейера. С его помощью можно исполнить приложение, предназначенное для целевой архитектуры, собрать статистику по исполненным инструкциям, однако нельзя получить какие-либо реальные метрики производительности. Скорость работы QEMU многократно превосходит скорость работы симуляторов с детальной имитацией конвейера.

- Трасса — это последовательность инструкций и данных, записанная в процессе работы приложения (или его части) и достаточная для его воспроизведения в будущем.

- Реплика является приложением, воссозданным по записанной трассе. Это приложение в точности повторяет действия ориги-

нального бинарного файла (инструкции, паттерны работы с памятью).

2.2. Существующий подход к характеризации интервалов

Перед использованием инструмента SimPoint необходимо поделить исполнение приложения на равные по количеству инструкций интервалы и характеризовать их некоторыми статистическими данными, на основе которых инструмент выполнит кластеризацию интервалов. Подход, описанный авторами инструмента в оригинальной статье [1], предлагает для каждого интервала собирать статистику по использованию в нем базовых блоков в формате <индекс базового блока>: <количество раз, которое он был исполнен в интервале * количество инструкций в этом базовом блоке>. Базовые блоки нумеруются по приложению в целом. Инструмент принимает на вход файл с векторами в следующем формате

T :<V1>:<N1> :<V2>:<N2> :<V3>:<N3> ...,

где T — это идентификатор начала строки, V{K} — индексы базовых блоков, N{K} — соответствующие им значения. Каждая такая строка содержит новый вектор. Векторы по порядку соответствуют последовательным интервалам исполнения приложения.

Стоит отметить, что такой формат входных данных, на самом деле, позволяет использовать для кластеризации интервалов различные статистические данные, собранные с интервалов. Так, вместо базовых блоков можно было бы использовать циклы или процедуры. Однако авторы инструмента SimPoint используют в своих экспериментах именно вектора базовых блоков и, что наиболее важно, определяют для этого типа данных оптимальные значения входных параметров инструмента. По этой причине для проведения первых экспериментов с технологией выделения характерных интервалов в компании YADRO был выбран именно такой подход.

Для сбора векторов базовых блоков (Basic Block Vectors, BBV) разработчиками был написан BBV-плагин для QEMU. Запустив эмуляцию приложения в QEMU с подключением этого плагина, мы на выходе получим файл, содержащий вектора базовых блоков для всех последовательных интервалов исполнения приложения. Количество инструкций в интервалах, на которые будет «нарезано» приложение, является опцией плагина. Для экономии памяти выходной файл сразу сжимается в .gz-архив с использованием алгоритма gzip.

2.3. Обзор инструмента SimPoint

2.3.1. Алгоритм работы

Инструмент получает на вход файл с векторами базовых блоков (или любыми другими векторами в заданном формате). Опция `-inputVectorsGzipped` отвечает за то, сжат ли файл в .gz-архив. Далее SimPoint выделяет различные фазы исполнения приложения и находит наиболее репрезентативные интервалы для каждой из фаз. Для этого выполняются следующие шаги.

Нормализация. Каждый частотный вектор нормализуется так, чтобы сумма значений в нем равнялась 1.

Сжатие размерности данных. Количество базовых блоков в индустриальных бенчмарках достигает десятков или сотен тысяч, в то время как количество интервалов в них достигает тысяч или десятков тысяч. Здесь мы сталкиваемся с широко известной в математике проблемой — «проклятием размерности». Этот термин в 1961 году ввел американский математик Ричард Беллман. Он описал ряд трудностей при обработке датасетов с большим количеством параметров. Среди них — возрастающая сложность вычислений, переобучение, высокий шум и сложность кластеризации.

Для решения этих проблем применяется уменьшение размерности данных. Существует два принципиально разных подхода к тому, как это можно делать: выбор признаков и проекция признаков. Выбор признаков просто удаляет все признаки (в нашем случае базовые блоки),

кроме небольшого числа тех, которые наилучшим образом характеризуют данные. Однако такой подход выбрасывает огромное число данных в проигнорированных измерениях. Проекция признаков, в свою очередь, уменьшает размерность данных путем создания нового пространства меньшей размерности и проекции каждой точки исходных данных в это пространство (измерения нового пространства не связаны напрямую с измерениями старого).

В инструменте SimPoint используется *случайная линейная проекция* для создания малоразмерного пространства, в которое проецируются данные. Это простой и вычислительно эффективный способ уменьшить размерность данных, сохранив при этом их свойства. Если есть матрица веторов базовых блоков X размера $N_{intervals} \times D_{numbb}$, где D_{numbb} — это количество базовых блоков в программе, то для уменьшения размерности данных до D_{new} методом случайной линейной проекции необходимо:

- создать случайную матрицу проекции M размера $D_{numbb} \times D_{new}$;
- вычислить $X' = X \times M$, получив новый датасет размера $N_{intervals} \times D_{new}$, то есть данные в пространстве меньшей размерности.

Существуют разные способы задания значений случайной матрицы M. Авторы оригинального подхода в качестве значений матрицы выбирают случайные вещественные числа между -1 и 1. Порождающий элемент для генератора псевдослучайных чисел, используемого для инициализации матрицы проекции, задается опцией `-seedproj` (значение по умолчанию 2042712918). Стоит также отметить, что известной и общепринятой является гауссова случайная проекция, в которой элементы матрицы взяты из распределения $N(0, \frac{1}{D_{new}})$.

В своей статье [4] авторы инструмента проводят эксперименты на бенчмарках из набора SPEC CPU2000 [20] для поиска оптимального значения размерности уменьшенного пространства и находят, что при $D_{new} = 15$ в данных по-прежнему хорошо различимы фазы исполнения, однако при меньших значениях после проекции наблюдается потеря существенного количества информации. По этой причине входному

аргументу `-dim` инструмента SimPoint, отвечающему за размерность сжатых данных, присвоено значение по умолчанию 15.

Запуск алгоритма k-means на данных уменьшенной размерности. Для кластеризации SimPoint использует алгоритм k-means. Это хорошо известный и эффективный итеративный алгоритм кластеризации.

Алгоритм начинает работу со случайного задания k различных центров. Точки данных разбиваются на кластеры по близости к назначенным центрам. Далее происходит итеративный процесс: перевычисляется центр для каждого кластера, после чего векторы разбиваются на кластеры вновь в соответствии с тем, какой из новых центров оказался ближе по выбранной метрике расстояния. Процесс продолжается до тех пор, пока кластеры не перестанут меняться.

Полученная кластеризация может сильно зависеть от начального выбора центров. Поэтому для каждого k SimPoint выполняет кластеризацию несколько раз с различной случайной инициализацией центров в начале работы алгоритма и выбирает лучшую согласно заданному критерию качества. Количество раз задается с помощью опции `-numInitSeeds`, которая по умолчанию равна 5. Порождающий элемент для генератора псевдослучайных чисел, используемого для инициализации центров, задается опцией `-seedkm` (значение по умолчанию 493575226).

Алгоритм k-means завершает работу, когда он сошелся. Однако для ускорения работы алгоритма можно ограничивать максимальное число итераций на каждую кластеризацию. Для этого в инструменте SimPoint есть параметр `-iters`, который по умолчанию равен 100. Авторы оригинального подхода в своей статье [4] показали, что алгоритм часто сходится и завершает работу гораздо раньше. Только 1.1% из всех запусков на приложениях из SPEC CPU2000 не сошлись за 100 итераций. Значение опции «off» соответствует отсутствию максимального количества итераций и остановке алгоритма по достижению сходимости.

Количество кластеров может быть задано с помощью опции `-k`. Безусловно, главной проблемой в использовании алгоритма k-means явля-

ется необходимость предварительного задания количества кластеров, в то время как число фаз приложения инженерам почти всегда заранее неизвестно. Задача поиска подходящего значения k является действительно нетривиальной, и ей будет уделено особое внимание в рамках этой работы.

Поиск оптимального значения k (опционально). В инструменте SimPoint реализована возможность поиска оптимального числа кластеров. Однако эксперименты инженеров YADRO показали, что эта функциональность работает плохо. А именно, при попытке оценить время исполнения бенчмарков из набора SPEC CPU2006 [21] такой подход показал хорошую сходимость к реальному значению (то есть разницу менее 5%) только на 6 из 35 приложений. Даже тривиальный запуск с некоторым фиксированным k для всех приложений из набора позволяет достичь хорошей сходимости на бóльшем числе приложений. Поэтому инженеры компании не пользуются этой функциональностью, а вместо этого подбирают подходящее k вручную, например, с помощью профилирования приложения.

По этой причине мы не будем детально рассматривать алгоритм поиска оптимального k. Ознакомиться с ним можно в той же статье [4].

Выбор характерных интервалов для найденной кластеризации. Для каждого кластера выбирается один репрезентативный интервал, который в будущем будет детально симулирован для представления поведения всей фазы. В качестве интервала-представителя выбирается ближайший к центроиду (центру кластера) интервал. Каждому характерному интервалу также назначается вес, который равен отношению числа интервалов в кластере к общему числу кластеров приложения. Сумма весов всех характерных интервалов, соответственно, равна 1.

Номера интервалов сохраняются в файл, путь которого задан в опции `-saveSimpoints`, в следующем формате:

$<I_0>$ 0
$<I_1>$ 1
...

$<I_N>$ N

Веса интервалов сохраняются в файл, путь которого задан в опции `-saveSimpointWeights`, в следующем формате:

$<W_0>$ 0
$<W_1>$ 1
...
$<W_N>$ N

Характерные интервалы еще называют *точками симуляции*. Отсюда происходит название инструмента SimPoint как сокращение от simulation point — точка симуляции.

Отметим, что код инструмента написан на языке программирования C++.

2.3.2. Применение

Инструмент SimPoint был задуман с целью быстрой оценки *относительных* метрик приложения без полного его запуска. К относительным метрикам, например, относятся IPC (instructions per cycle, количество инструкций на цикл), L1D CMR (L1D cache miss rate, доля промахов в кэш L1D), Bad Speculation (доля слотов конвейера, потраченная впустую из-за неправильных спекуляций).

Рассмотрим, как выглядит применение инструмента SimPoint в процессе оценки качества архитектурных изменений в процессоре. Представим, что есть индустриальный бенчмарк, исполнение которого на симуляторе с детальной имитацией конвейера заняло бы неприемлемо большое количество времени. Мы хотим иметь возможность быстро узнать, как меняется относительная метрика производительности X на этом приложении вместе с изменением различных параметров архитектуры (иерархии кэшей, размера кэш-линии, алгоритма предсказания переходов и других). Для этого необходимо выполнить следующие шаги.

1. Задать размер интервала и собрать с приложения вектора базовых блоков с помощью BBV-плагина в QEMU (см. раздел 2.2).

2. На собранных данных запустить инструмент SimPoint, который сгенерирует файлы с характерными интервалами исполнения приложения и их весами (см. раздел 2.3.1).

3. С помощью tracer-плагина для QEMU, разработанного в компании YADRO, для выбранных интервалов записать их трассы.

4. Используя специальный инструмент, также разработанный внутри в YADRO, с записанных трасс сгенерировать реплики (бинарные файлы), пригодные для запуска на целевой архитектуре.

5. Для каждой архитектурной конфигурации выполняется симуляция реплик всех характерных интервалов на одном из симуляторов с детальной имитацией конвейера. На выходе мы имеем значения метрики X, вычисленные для каждой из точек симуляции. Теперь, чтобы узнать ожидаемое значение $\hat{X}$ метрики X на всем приложении, необходимо вычислить

$$\hat{X} = \sum_{i=0}^{N} X_i \cdot W_i,$$

где X_i — значение метрики X на i-ом характерном интервале, W_i — вес i-го характерного интервала.

Несмотря на то, что инструмент SimPoint изначально был разработан авторами для создания возможности быстрого оценивания относительных метрик приложения, можно легко распространить идею его использования и для оценки *абсолютных* величин, например, числа тактовых циклов исполнения приложения или количества появлений определенной инструкции. Для этого необходимо отмасштабировать веса, полученные после работы инструмента:

$$W_i^{scaled} = W_i \cdot I,$$

где W_i^{scaled} — масштабированный вес для i-го интервала, W_i — вес i-го интервала, I — количество интервалов приложения. Иными словами,

для оценки абсолютных величин в качестве веса характерного интервала берется количество интервалов в его кластере.

Главная цель разработчиков аппаратного обеспечения — максимально ускорить исполнение приложений на процессоре. Поэтому целевая метрика для них — это *время исполнения приложения (elapsed time)*. Это важнейший показатель, на который инженеры ориентируются в первую очередь при выборе оптимального набора параметров проектируемого процессора. А потому в рамках этой работы при проведении экспериментов мы будем оценивать именно эту метрику.

2.4. Симуляторы и оценка сходимости

Как было отмечено ранее, сбор векторов базовых блоков с приложения осуществляется в QEMU, а запуск реплик характерных интервалов — на симуляторах с детальной имитацией конвейера, которые позволяют узнать реальные метрики производительности системы. Рассмотрим, какие виды последних используются в компании YADRO.

В качестве программного симулятора с детальной имитацией конвейера используется Gem5 [32]. Его неоспоримым преимуществом является возможность быстрого изменения параметров архитектуры, на которой проводятся эксперименты: часть ключевых характеристик процессора можно задавать опциями или в конфигурационных файлах (например, размер кэш-линии или ассоциативность кэша). При этом существенным недостатком является то, что время симуляции индустриальных бенчмарков в Gem5 может достигать нескольких месяцев.

В качестве аппаратного симулятора с детальной имитацией конвейера используется FPGA (*рус.* программируемая пользователем вентильная матрица). Это полупроводниковое устройство, которое может быть сконфигурировано инженерами так, чтобы оно вело себя практически как настоящий процессор. Главным преимуществом такого подхода перед Gem5 является гораздо меньшее время исполнения приложений, хотя оно всё еще многократно превосходит показатели реального процессора. В то же время на FPGA отсутствует возможность гибкого из-

менения параметров архитектуры — для этого требуется долгая работа по конфигурации интегральной схемы со стороны инженеров.

В ходе работы для оценки качества выбора характерных интервалов мы будем вычислять относительную разницу между оценкой времени исполнения приложения, полученной с помощью характерных интервалов, и его фактическим временем исполнения. Иными словами, мы будем определять *сходимость* оценки к реальному значению. Учитывая, что оригинал приложения (вместе с его входными данными) и реплики характерных интервалов запускаются на одном и том же симуляторе, в формулах вместо времени можно использовать *количество тактовых циклов*, за которое был исполнен бинарный файл. На практике это будет удобно ввиду наличия в процессоре аппаратного счетчика числа исполненных тактовых циклов.

Таким образом, для оценки сходимости мы будем вычислять относительное отклонение

$$\Delta = (\frac{\hat{X}}{X} - 1) * 100\%,$$

где X — количество тактовых циклов, за которое было исполнено оригинальное приложение с заданными входными данными, $\hat{X}$ — предполагаемое число тактовых циклов, рассчитанное при помощи точек симуляции. Мы будем стремиться к минимизации Δ.

В компании YADRO принято говорить, что если набор характерных интервалов позволяет оценить время исполнения приложения с $|\Delta|$ не более 5%, то этот набор показывает *хорошую* сходимость. Если же $|\Delta|$ не превосходит 10%, то говорят об *удовлетворительной* сходимости.

Ясно, что в ходе проведения экспериментов необходимо будет запускать на симуляторе с детальной имитацией конвейера не только реплики характерных интервалов, но и исходные приложения. Поэтому в рамках этой работы мы будем проводить все запуски оригинальных приложений и реплик характерных интервалов на FPGA, *зафиксировав архитектурную конфигурацию*. Использование Gem5 для этих целей заняло бы непозволительное количество времени.

3. Метод

В этом разделе представлен новый метод выделения характерных интервалов исполнения приложения.

3.1. Способ характеризации интервалов

Использование векторов базовых блоков (BBV) в качестве входных данных для инструмента SimPoint, без сомнений, имеет одно существенное преимущество: этот способ сбора статистики с интервалов приложения не зависит от процессорной архитектуры, для которой предназначено это приложение. Именно по этой причине такой подход был предложен авторами инструмента SimPoint в их оригинальной статье [4]. Однако в контексте оценки времени исполнения приложения он имеет два существенных недостатка.

Во-первых, BBV не учитывают характер исполненных инструкций. В двух разных базовых блоках могут исполняться практически одни и те же инструкции, и их время исполнения будет практически одинаково. Это значит, что кластеризация интервалов на основе базовых блоков может быть менее эффективна, чем на основе характера исполненных инструкций.

Во-вторых, вектора базовых блоков имеют очень большую размерность (от нескольких десятков до сотен тысяч), что ведет к «проклятью размерности» (обсуждалось в разделе 2.3.1) и невозможности визуализации. Человеческий глаз с экрана может воспринимать только пространства размерности 1 и 2. Но сжатие исходных векторов базовых блоков до такой малой размерности не является «безопасным» в том смысле, что будет потеряно слишком много информации, и вероятность сохранения относительных расстояний между точками будет неприемлемо мала. Для целей визуализации лучше всего подойдет такая статистика, собранная с интервалов, размерность векторов которой не превышает нескольких десятков.

Ввиду существования этих проблем предлагается выполнять кластеризацию интервалов не на основе векторов базовых блоков, а на ос-

нове векторов категорий исполненных инструкций (instruction category vectors, ICV). Под этим подразумевается сбор с каждого интервала приложения статистики о количестве раз, которое была исполнена каждая инструкция, с разбиением инструкций по категориям. Необходимость разбиения на категории обусловлена наличием большого числа сходных с точки зрения времени исполнения инструкций, которые не хотелось бы разделять между собой, например, `add` и `sub` (сложение и вычитание).

3.2. Кластеризация для фиксированного числа кластеров

Ключевой функциональностью инструмента SimPoint является выделение характерных интервалов для заданного числа кластеров k. Несмотря на то, что в большинстве случаев точное число фаз исполнения приложения неизвестно, во внутреннем инструменте-аналоге компании YADRO эту функциональность важно сохранить, поскольку в ряде случаев инженерам может потребоваться получить строго заданное число характерных интервалов, например, если известны детали исходного кода приложения или удалось узнать число фаз благодаря профилированию.

В рамках экспериментального исследования были проверены следующие алгоритмы, позволяющие решить задачу кластеризации для фиксированного k: k-means [12], bisecting k-means [10], spectral clustering [14], X-Means [8], fuzzy C-Means [11]. Каждый из них был протестирован с различными наборами значений входных параметров.

Использование алгоритма **bisecting k-means** позволило достичь хорошей сходимости на наибольшем числе приложений. Таким образом, предлагается использовать этот алгоритм для задачи кластеризации при фиксированном k. Рассмотрим подробнее схему его работы.

1. Считаем все данные одним кластером.

2. Используем алгоритм k-means с параметром k=2 для разделения

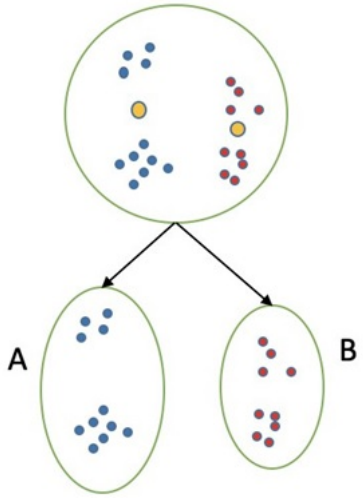

Рис. 1: Алгоритм bisecting k-means: шаг 2.

кластера на два (рис. 1).

3. Вычисляем внутрикластерное расстояние для каждого кластера (sum of squared errors, SSE):

$$\sum_{i=0}^{n} \left(X_i - \overline{X}\right)^2$$

4. Выбираем кластер с наибольшим внутрикластерным расстоянием и делим на два с помощью алгоритма k-means с параметром k=2 (рис. 2)

5. Повторяем шаги 2-4 до получения k кластеров (рис. 3)

Таким образом, для рис. 3 результатом выполнения алгоритма являются кластеры C, D, E, F.

Bisecting k-means является *иерархическим* алгоритмом кластеризации, то есть в рамках его работы создается иерархия (дерево) вложенных кластеров. Эта особенность обуславливает важное преимущество этого алгоритма, которого лишён алгоритм k-means, используемый в инструменте SimPoint. А именно, если получены реплики для набора из N характерных интервалов и нужно получить набор из N+1 интервала, то для этого необходимо собрать трассы и сгенерировать реплики лишь **не более чем для двух** новых интервалов. И действительно, в

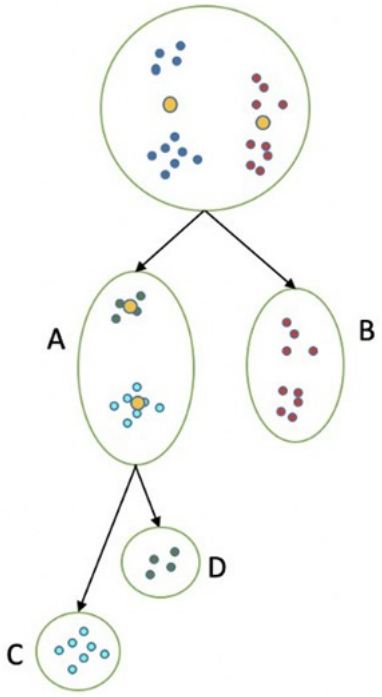

Рис. 2: Алгоритм bisecting k-means: шаг 4.

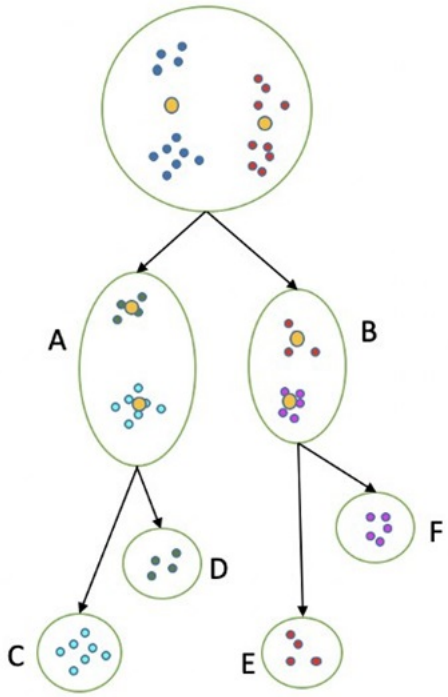

Рис. 3: Алгоритм bisecting k-means: шаг 5.

алгоритме bisecting k-means разбиение на N+1 кластер получается из разбиения на N кластеров путем разделения одного какого-то кластера на два. Далее мы воспользуемся этим преимуществом для ускорения поиска оптимального числа кластеров.

3.3. Поиск оптимального числа кластеров

Задача поиска оптимального k является нетривиальной. И, как уже было отмечено в разделе 2.3.1, инструмент SimPoint с ней справляется настолько неудовлетворительно, что инженерам YADRO приходится вручную подбирать оптимальное k. Это обуславливает необходимость создания принципиально нового подхода к поиску подходящего k.

Для этого, в первую очередь, был проведён ряд экспериментов для проверки следующих алгоритмов кластеризации с автоматическим определением числа кластеров: affinity propagation [9], mean-shift [13], HDBSCAN [6], OPTICS [7], G-Means [5]. Оказалось, что тривиальный запуск с фиксированным k=5 для всех приложений позволяет достичь хорошей сходимости на бóльшем числе приложений, чем каждый из этих алгоритмов.

Следующим шагом стала попытка создания способа автоматического определения оптимального k для выбранного алгоритма кластеризации для фиксированного k (bisecting k-means). А именно, были проверены критерии выбора k на основе следующих метрик модели: среднее внутрикластерное расстояние, силуэт [18], индекс Дэвиса-Булдина [16]. Как и в случае с алгоритмами, автоматически определяющими число кластеров, ни один из проверенных способов не показал результат лучше, чем тривиальный запуск с фиксированным k=5 для всех приложений.

Сложность определения подходящего k в контексте решаемой задачи обусловлена поиском малого количества характерных интервалов относительно общего числа интервалов исполнения приложения. Так, при работе с индустриальными бенчмарками число интервалов варьируется, как правило, от тысяч до десятков тысяч, а количество харак-

терных интервалов, которое мы стремимся выбрать, не превосходит 20. Это, в свою очередь, ведет к отсутствию какой-либо объяснимой зависимости сходимости от числа кластеров при столь малых значениях k, поскольку в таких случаях влияние отклонения от среднего в фазе значения оцениваемой метрики (например, времени исполнения) для каждого характерного интервала очень велико. Иными словами, из-за малых значений k на практике мы далеко не всегда наблюдаем реализацию интуитивного предположения о том, что увеличение k ведет к улучшению сходимости.

И тем не менее, для того чтобы облегчить инженерам YADRO работу по поиску оптимального k, были разработаны два новых подхода, которые можно использовать вместе.

3.3.1. Одновременное получение наборов характерных интервалов для диапазона значений k

Ввиду описанных выше проблем у инженеров зачастую возникает потребность в получении *наборов* характерных интервалов (под этим подразумеваются реплики характерных интервалов вместе с весами) для некоторого диапазона k, чтобы затем выбрать из этого диапазона такое k, на котором достигается хорошая сходимость. Для того чтобы выполнить это для k от M до N при помощи инструмента SimPoint, инженерам приходится запускать инструмент N-M+1 раз. Далее для каждой полученной *разметки* (файлы с интервалами и весами) необходимо записать трассы с интервалов и сгенерировать соответствующие им реплики. У такого подхода есть два существенных недостатка.

Во-первых, если какой-то интервал встречается в нескольких разметках, то работа по записыванию трассы и генерации реплики для него будет проделана столько раз, сколько он встретился.

Во-вторых, алгоритм кластеризации k-means, используемый в инструменте SimPoint, не даёт гарантий какой-либо связи разметок для N и N+1 интервалов. А это значит, что для диапазона k от M до N потенциально может быть нужно записать трассы и сгенерировать ре-

плики для

$$\sum_{k=M}^{N} k = \frac{(N - M + 1)(M + N)}{2}$$

различных интервалов. Такая сумма растет квадратично с увеличением N, что ведет к быстрому росту вычислительных и временных ресурсов, необходимых для получения готовых наборов реплик для k от M до N. Например, чтобы проверить сходимость при k от 1 до 15, может понадобиться получить реплики 120 различных интервалов.

Как было отмечено ранее, выбранный алгоритм bisecting k-means обладает следующим преимуществом: если получены реплики для набора из N характерных интервалов и нужно получить набор из N+1 интервала, то для этого необходимо собрать трассы и сгенерировать реплики лишь не более чем для двух новых интервалов. Это приводит нас к идее о том, что можно реализовать функциональность сбора единого файла с характерными интервалами для выбранного диапазона k и разметок, использующих ключи из этого общего файла. С одной стороны, такой подход позволит разом записать трассы и сгенерировать реплики для всех необходимых интервалов, причем без повторения одних и тех же действий для каких-либо интервалов, как это было в случае с инструментом SimPoint.

А с другой стороны, благодаря упомянутому преимуществу алгоритма bisecting k-means, для диапазона k от M до N понадобится получить не более чем

$$M + \sum_{k=M+1}^{N} 2 = M + 2 \cdot (N - M) = 2N - M$$

интервалов. А такая сумма растёт линейно с увеличением N. Так, чтобы проверить сходимость при k от 1 до 15, нужно будет получить реплики не более чем 29 различных интервалов, что в несколько раз меньше, чем для оригинального подхода.

3.3.2. Визуализация распределения интервалов и результатов кластеризации

В разделе 3.1 мы обсудили, что одним из преимуществ ICV-данных перед BBV-данными является возможность их адекватной визуализации. Полученные на выходе работы ICV-плагина вектора имеют размерность не более двух десятков, что позволяет «безопасно» (без потери существенного количества информации) уменьшать их размерность до 1 или 2 и использовать их для визуализации.

Предлагается использовать это преимущество для создания инструмента для визуализации распределения интервалов исполнения приложения и результатов кластеризации. Для построения графика распределения интервалов необходимо считать файл с векторами в стандартном формате, сжать их в пространство размерности 1, нормировать полученные данные и построить график зависимости значения в полученном пространстве от интервала. Предполагается, что чем меньше отличаются значения для некоторых двух интервалов, тем более похожий набор инструкций (с точки зрения категорий) в этих интервалах был исполнен.

Для визуализации результатов кластеризации на этом графике необходимо раскрасить разными цветами точки для разных кластеров, а также выделить центры кластеров.

Сценарии использования инструмента включают следующие.

- Визуальная оценка примерного количества фаз приложения.
- Визуальная оценка качества полученной кластеризации без сбора трасс характерных интервалов, генерации реплик, проведения экспериментов на симуляторах. Это позволяет быстро определить заведомо плохие кластеризации.
- Дополнительная проверка того, что выбранные характерные интервалы показывают хорошую сходимость не случайно, а действительно соответствуют разным фазам исполнения приложения и покрывают все или существенную часть из них.

4. Особенности реализации

В этом разделе приведены некоторые особенности программной реализации метода. На написанный код действует соглашение о неразглашении (NDA), в связи с чем он не может быть опубликован публично.

4.1. ICV-плагин для QEMU

Для сбора векторов категорий инструкций со всех интервалов исполнения приложения был разработан ICV-плагин для QEMU (instruction category vectors). Основной сложностью в рамках реализации стала ручная классификация инструкций по категориям. В начальной версии плагина были выделены следующие категории инструкций:

- Arithmetic — арифметические операции;
- Logical — логические операции;
- Shift — битовые сдвиги;
- BitManip — битовые манипуляции;
- ControlFlow — инструкции управления потоком исполнения;
- Call — инструкции вызова процедуры;
- Load — выгрузка из памяти;
- Store — загрузка в память;
- Atomics — атомарные инструкции;
- Integer — целочисленная арифметика;
- FloatingPoint — арифметика чисел с плавающей точкой;
- System — системные инструкции;
- Vector — векторные операции;

- Crypto — инструкции для криптографии.

Одна инструкция может относиться к нескольким категориям. Статистика собирается для всех пересечений вышеперечисленных категорий.

Реализация плагина подразумевает создание функции обратного вызова (англ. callback) на событие поступления нового базового блока. В этой функции, в свою очередь, регистрируются обратные вызовы для событий исполнения инструкций из поступившего базового блока. В последних, наконец, определяется принадлежность инструкции к категориям, и глобальная статистика обновляется. После работы плагина собранные данные записываются в файл такого же формата, как и в случае с BBV-плагином, а также сжимаются при помощи утилиты gzip.

4.2. Инструмент IEst

Для написания внутреннего аналога инструмента SimPoint был выбран язык программирования Python. Основной причиной этому послужило наличие исчерпывающих библиотек для машинного обучения и линейной алгебры, что позволяет не реализовывать алгоритмы сжатия размерности и кластеризации с нуля самостоятельно, а воспользоваться готовыми отлаженными функциями. Простота и быстрота внедрения изменений в код также являются преимуществами.

Известно, что программы на Python могут в разы уступать в производительности программам, написанным на C/C++. Однако в нашем случае это не станет проблемой, поскольку бóльшая часть вычислительной нагрузки будет проходиться на модули, написанные на C/C++ и обернутые в Python (в частности, на те, что содержатся в библиотеке NumPy [15]). Кроме того, учитывая, что работа инструмента в целом занимает совсем небольшое время (от нескольких секунд до минут) в то время как запись трасс и генерация реплик занимают несколько часов, при таких величинах возможные задержки для нас не имеют значения.

Новый инструмент был назван IEst, от английского Intervals for Estimation (интервалы для оценки). Для фиксированного k алгоритм

работы IEst отличается от SimPoint (рассмотрен в разделе 2.3.1) использованием *разреженной случайной линейной проекции* на этапе уменьшения размерности данных и применением алгоритма кластеризации bisecting k-means вместо k-means (особенности выбранного алгоритма обсуждались в разделе 3.2).

Сжатие с помощью разреженной случайной матрицы является альтернативой проекции при помощи плотной матрицы, которая гарантирует аналогичное качество сжатия, будучи при этом гораздо более эффективной по памяти и позволяющей более быстрое вычисление сжатых данных.

Для разреженной матрицы задается параметр плотности $density$. Тогда если обозначить за $s = \frac{1}{density}$, то элементы случайной матрицы задаются как

$$\begin{cases} -\frac{\sqrt{s}}{\sqrt{D}} & \text{с вероятностью } \frac{1}{2s} \\ 0 & \text{с вероятностью } 1 - \frac{1}{s} \\ \frac{\sqrt{s}}{\sqrt{D}} & \text{с вероятностью } \frac{1}{2s}, \end{cases}$$

где D — это размерность целевого пространства. Мы будем брать в качестве значения плотности $density = \frac{1}{\sqrt{N}}$, где N — размерность исходного пространства, как рекомендовано в статье [3].

Опции инструмента SimPoint, которые используются и в новом инструменте, были названы так же. Была добавлена опция `-saveScaledWeights`, с помощью которой можно задать путь к файлу для сохранения масштабированных весов (см. раздел 2.3.2). Кроме того, была добавлена опция `-saveLabels` для задания пути к файлу, куда для каждого интервала будут записаны кластер, к которому он относится, и расстояние до центра этого кластера.

Также была реализована функциональность одновременной генерации разметок для диапазона значений k (см. раздел 3.3.1). Для этого необходимо указать ключ `-kRange`, а в опции `-kMin` и `-kMax` передать минимальное и максимальное значения диапазона соответственно. После чего в папке, переданной в опции `-kRangeFolder`, будет создана структура, включающая файл all.intervals со всеми интервалами, которые встретились в разметках для k от kMin до kMax, а также папки с

названиями-числами для каждого k из диапазона. Папка для каждого k содержит три файла: выбранные интервалы (simpoints.intervals), веса (simpoints.weights), масштабированные веса (simpoints.scaled_weights). В этих файлах используются не ключи из диапазона от 0 до k-1, как это происходит при запуске для фиксированного k, а ключи из общего файла all.intervals.

Значения по умолчанию для параметров `-iters` (300) и `-numInitSeeds` (5) были подобраны в рамках проведенных экспериментов. Для остальных опций значения по умолчанию выставлены из общих соображений и могут часто меняться от запуска к запуску.

4.3. Инструмент для визуализации распределения интервалов и результатов кластеризации

На языке Python с помощью модуля pyplot библиотеки matplotlib [19] был написан инструмент для визуализации распределения интервалов исполнения приложения и результатов кластеризации. Алгоритм его работы и сценарии использования были рассмотрены в разделе 3.3.2.

Приведем список опций этого инструмента.

- `-loadFVFile` <path> — файл с векторами в заданном формате.
- `-inputVectorsGzipped` — указывает на то, что файл с векторами был сжат при помощи утилиты gzip.
- `-labelsFile` <path> — файл, соответствующий опции `-saveLabels` инструмента IEst.
- `-spIntervalsFile` <path> — файл, соответствующий опции `-saveSimpoints` инструмента IEst.
- `-saveImage` <path> — путь для сохранения построенного графика (опционально).

- `-seedproj` <n> — инициализирующее ядро для генератора случайных чисел, используемого в случайной линейной проекции.
- `-showClusters` — выделять ли на графике интервалы, относящиеся к разным кластерам, разными цветами.
- `-showCenters` — выделять ли на графике центры кластеров.
- `-noDisplay` — если указано, построенный график не будет показан (но будет сохранен по пути, указанному в опции `-saveImage`).

4.3.1. Примеры визуализации

На рисунке 4 приведена визуализация распределения интервалов исполнения приложения 435.gromacs. На оси абсцисс отложены интервалы исполнения приложения, на оси ординат — некое число в пространстве размерности 1, полученное после проекции и нормализации. Напомним, что нумерация интервалов соответствует порядку их появления при исполнении приложения и начинается с 0.

На графике отчётливо видны две линии, на которых сконцентрированы практически все точки, соответствующие интервалам. Поскольку значения точек тем ближе друг к другу, чем ближе характер исполненных инструкций в соответствующих интервалах, каждая такая линия отражает фазу исполнения приложения. Таким образом, хорошо заметны две фазы. Количество остальных точек незначительно относительно общего числа интервалов (более 25000). Профилирование приложения, выполненное коллегами, также показывает два «горячих» участка кода. Таким образом, без каких-либо дополнительных действий (записи трасс, генерации реплик, запусков на FPGA), мы понимаем, что оптимальное количество кластеров для этого приложения равно двум.

Отметим, что важно рисовать интервалы на графике именно в порядке их появления во время исполнения приложения. Сама по себе близость интервалов по значению оси ординат еще не говорит о том, что они относятся к одной фазе и демонстрируют приблизительно одинаковые показатели производительности. Однако одновременная бли-

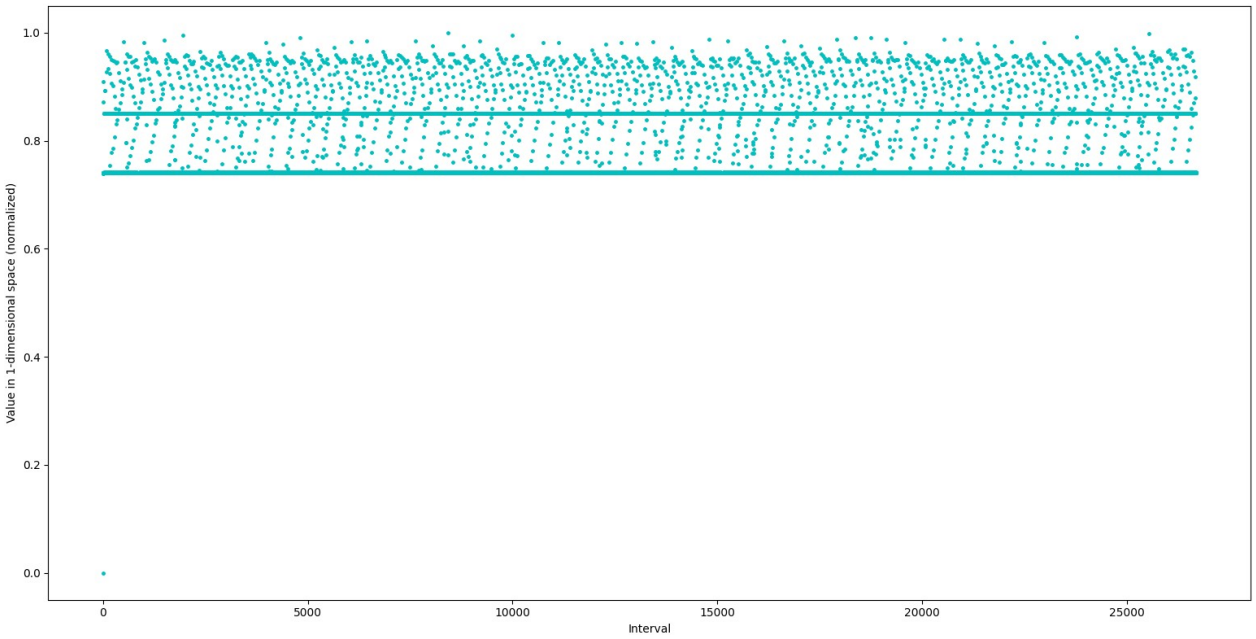

Рис. 4: Распределение интервалов для 435.gromacs.

зость интервалов и по оси абсцисс (то есть по появлению во время исполнения), и по оси ординат, с гораздо большей степенью уверенности свидетельствует о принадлежности интервалов к одной фазе.

На рисунке 5 приведена визуализация результатов кластеризации для приложения 403.gcc при k=3. Отчетливо видно, что k=3 для этого приложения недостаточно, поскольку некоторые очевидно разные фазы, то есть линии разной высоты, были отнесены к одному кластеру (раскрашены в один цвет). Для сравнения рисунок 6 демонстрирует результаты кластеризации для этих же интервалов при k=15. Видно, что в данном случае хорошо отличимые друг от друга фазы отнесены к разным кластерам, что означает, что такую кластеризацию стоит пробовать использовать для оценки метрик исполнения приложения. Но в то же время такая визуализация совершенно не гарантирует хорошей сходимости.

Используя инструмент, можно также строить графики распределения интервалов в пространстве размерности 2. Они лишены информации о близости интервалов по их появлению, поэтому определить примерное число фаз с помощью таких графиков гораздо сложнее. По этой причине мы опустим рассмотрение таких графиков в этой работе.

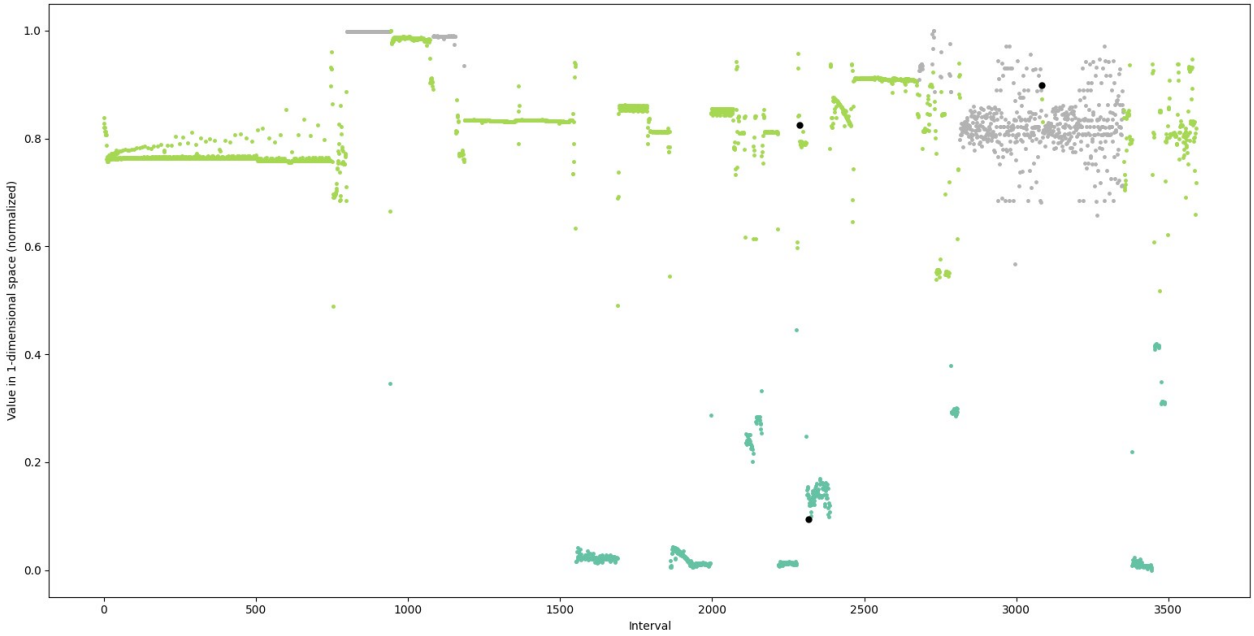

Рис. 5: Результаты кластеризации для 403.gcc при k=3.

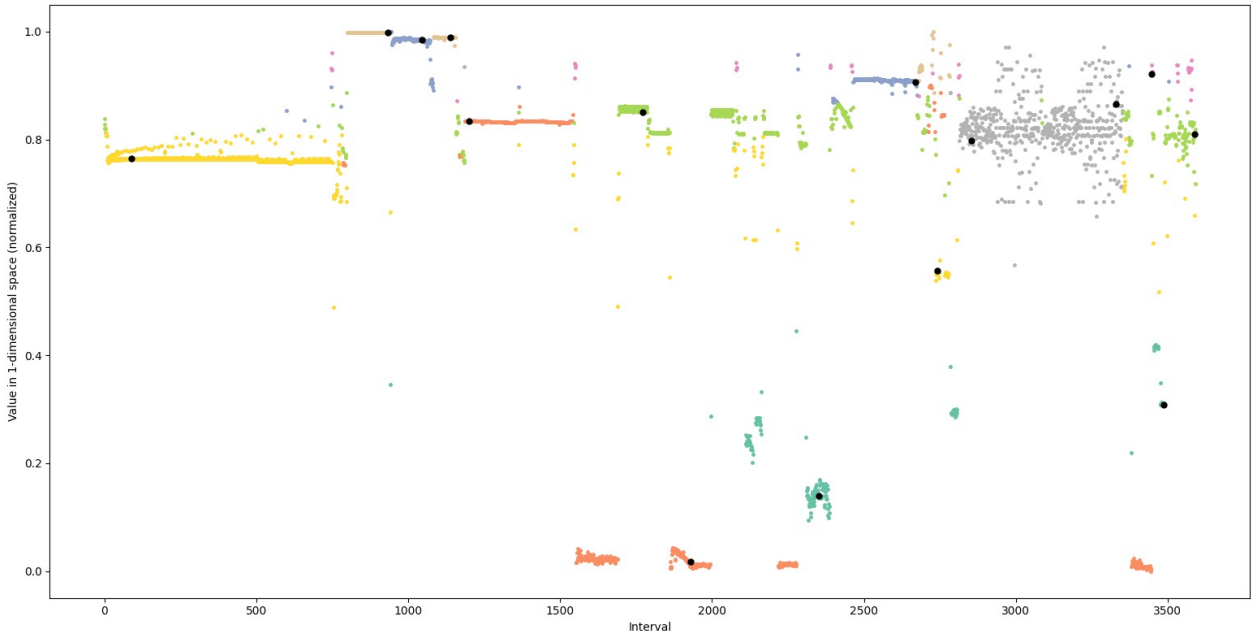

Рис. 6: Результаты кластеризации для 403.gcc при k=15 (интервалы те же, что и на рис. 5).

5. Экспериментальное исследование

Этот раздел посвящен экспериментальному сравнению существующего и нового методов выделения характерных интервалов исполнения приложения. Заданы исследовательские вопросы, рассмотрен набор используемых бенчмарков, разработан план проведения экспериментов и проанализированы полученные результаты.

5.1. Исследовательские вопросы

Целью экспериментального исследования является ответ на следующие два исследовательских вопроса.

RQ1 : Правда ли, что предложенный способ характеризации интервалов (ICV) работает лучше существующего подхода (BBV), а именно позволяет в среднем достичь лучшей сходимости и демонстрирует хорошую/удовлетворительную сходимость на бóльшем числе приложений из выбранного набора?

RQ2 : Правда ли, что выбранный алгоритм bisecting k-means в инструменте IEst работает лучше существующего алгоритма k-means в инструменте SimPoint, а именно позволяет в среднем достичь лучшей сходимости и демонстрирует хорошую/удовлетворительную сходимость на бóльшем числе приложений из выбранного набора?

5.2. Бенчмарки

Для оценки производительности процессоров используются наборы индустриальных бенчмарков. Некоммерческая организация SPEC (Standard Performance Evaluation Corporation) занимается разработкой и публикацией таких наборов тестов. Тестовые пакеты SPEC являются стандартами для оценки производительности современных компьютерных систем.

Тесты SPEC создаются на основе кода, поступающего из разных источников, и максимально приближены к насущным жизненным про-

блемам. К примеру, пакет тестов для Java тестирует как простейшие вычисления (SPECjbb), так и всю систему в целом, включая Java EE, базу данных, диск и сеть. Набор тестов SPEC CPU для тестирования процессоров включает в себя компилятор gcc, химическую программу GAMESS, программу для предсказания погоды WRF и утилиты для решения задач молекулярной и гидродинамики.

Тестовые пакеты SPEC написаны на языках программирования C, Java и Fortran и поставляются в виде исходных кодов, что позволяет портировать тесты на множество платформ.

Для проведения экспериментов мы будем использовать бенчмарки из набора SPEC CPU2006 [21], а именно подгруппу CINT2006, измеряющую производительность целочисленных вычислительных задач. Набор CINT2006 содержит 35 вариантов.

К каждому приложению из набора поставляется несколько вариантов входных данных (может быть один). Приложение вместе с заданными входными данными создает определенную нагрузку на систему. Каждую из них мы будем называть *вариантом*. Варианты приложения нумеруются, начиная с 0. Каждый вариант соответствует определенному набору входных данных. Говоря об оценке времени исполнения приложения, мы всегда имеем в виду оценку времени исполнения какого-то из его вариантов, то есть приложения при определенных входных данных.

5.3. Планирование экспериментов

Для каждого из двух сравниваемых подходов будут получены разметки, записаны трассы, сгенерированы и запущены на FPGA реплики и посчитаны относительные отклонения оценки времени исполнения (Δ в разделе 2.4) для всех 35 вариантов. Затем для каждого варианта сравниваются абсолютные значения Δ двух подходов (то есть не имеет значения, ниже или выше оказалась оценка, главное — насколько она близка к реальному значению). Если разница абсолютных значений не превосходит 2%, то мы будем говорить, что качество приближения оди-

наково (далее обозначаем как EQL). В противном случае мы говорим, что тот подход, который показал **отклонение меньше**, **отработал лучше** на рассматриваемом варианте.

Мы будем говорить, что **подход A лучше подхода B**, если выполнены четыре следующих условия.

1. Количество вариантов, на которых подход A отработал лучше подхода B, не меньше количества вариантов, на которых подход B отработал лучше A.

2. Количество вариантов, на которых подход A показал хорошую сходимость ($\Delta < 5\%$), не меньше количества вариантов, на которых подход B показал хорошую сходимость.

3. Количество вариантов, на которых подход A показал удовлетворительную сходимость ($\Delta < 10\%$), не меньше количества вариантов, на которых подход B показал удовлетворительную сходимость.

4. Для хотя бы двух условий из 1-3 выполнено строгое неравенство.

Для ответа на RQ1 будет запущен инструмент SimPoint на BBV и ICV данных, собранных со всех вариантов, для k=5, 7, 10. Для BBV установим `-dim`=15, как это рекомендовано в статье [4], а для ICV не будем использовать уменьшение размерности из-за малой размерности данных; `-iters`=300; значения остальных параметров взяты по умолчанию. Это позволит в равных условиях сравнить два способа характеризации интервалов: BBV и ICV.

Для ответа на RQ2 будут запущены инструменты SimPoint (`-iters`=300) и IEst на ICV данных для k=5, 7, 10 (значения остальных параметров взяты по умолчанию). Это позволит в равных условиях сравнить работу существущего и нового инструментов.

Поскольку для ответа на RQ1 и RQ2 инструменты SimPoint и IEst будут запущены с одинаковыми параметрами, это даст нам возможность провести итоговое сравнение существующего (BBV и SimPoint) и нового (ICV и IEst) методов выделения характерных интервалов для k=5, 7, 10.

5.4. Анализ результатов

По результатам проведенных экспериментов были сформированы подробные таблицы с результатами. Они были выложены на GitHub [2] с целью экономии места в работе, здесь же будут приведены только итоговые таблицы для сравнения подходов. Отметим, что в выложенных таблицах отсутствуют абсолютные значения времени исполнения, поскольку информация о реальной производительности проектируемого процессора является строго конфиденциальной.

5.4.1. RQ1

В таблицах [27], [28] и [26] приведены результаты экспериментов по сравнению BBV и ICV для k=5, 7 и 10 соответственно.

Ниже приведены обобщенные результаты для k=5. Согласно критерию сравнения, определенному в разделе 5.3, ICV-подход отработал лучше.

Лучший подход	Количество	В %
ICV	21	74%
EQL	5	
BBV	9	26%
Всего	35	100%

	BBV	ICV
$\|\Delta\| < 5\%$	3	7
$\|\Delta\| < 10\%$	12	18

Ниже приведены обобщенные результаты для k=7. Согласно критерию сравнения, ICV-подход отработал лучше.

Лучший подход	Количество	В %
ICV	18	66%
EQL	5	
BBV	12	34%
Всего	35	100%

	BBV	ICV
$\|\Delta\| < 5\%$	6	12
$\|\Delta\| < 10\%$	13	16

Ниже приведены обобщенные результаты для k=10. Согласно критерию сравнения, ICV-подход отработал лучше.

Лучший подход	Количество	В %
ICV	17	69%
EQL	7	
BBV	11	31%
Всего	35	100%

	BBV	ICV
$\|\Delta\| < 5\%$	3	7
$\|\Delta\| < 10\%$	11	14

Таким образом, из полученных результатов можно сделать уверенный вывод о том, что предложенный способ характеризации интервалов (ICV) работает лучше существующего (BBV) и позволяет достичь хорошей/удовлетворительной сходимости на бóльшем числе приложений.

Необходимо понимать, что тот факт, что для конкретного значения k BBV-подход отработал лучше ICV на некотором числе вариантов, не говорит о том, что он всегда работает лучше на этих вариантах. Вместе с изменением k меняются и те варианты, на которых BBV работает лучше или хуже. В этом смысле гораздо важнее убедиться, что при разных k ICV работает лучше на бóльшем числе вариантов, чем BBV.

5.4.2. RQ2

В таблицах [30], [31] и [29] приведены результаты экспериментов по сравнению инструментов SimPoint и IEst для k=5, 7 и 10 соответственно.

Ниже приведены обобщенные результаты для k=5. Согласно критерию сравнения, инструмент IEst с алгоритмом bisecting k-means отработал лучше.

Лучший подход	Количество	В %
IEst	21	77%
EQL	6	
SimPoint	8	23%
Всего	35	100%

	SimPoint	IEst
$\|\Delta\| < 5\%$	7	13
$\|\Delta\| < 10\%$	18	21

Ниже приведены обобщенные результаты для k=7. Согласно критерию сравнения, инструмент IEst отработал лучше.

Лучший подход	Количество	В %
IEst	16	69%
EQL	8	
SimPoint	11	31%
Всего	35	100%

	SimPoint	IEst
$\|\Delta\| < 5\%$	12	12
$\|\Delta\| < 10\%$	16	20

Ниже приведены обобщенные результаты для k=10. Согласно критерию сравнения, инструмент IEst отработал лучше.

Лучший подход	Количество	В %
IEst	20	77%
EQL	7	
SimPoint	8	23%
Всего	35	100%

	SimPoint	IEst
$\|\Delta\| < 5\%$	6	14
$\|\Delta\| < 10\%$	13	23

Таким образом, полученные результаты подтверждают, что новый инструмент IEst с алгоритмом bisecting k-means позволяет более эффективно выделять характерные интервалы, чем инструмент SimPoint.

5.4.3. Итоговое сравнение существующего и нового методов

В таблицах [24], [25] и [23] приведены результаты итогового сравнения существующего (BBV и SimPoint, обозначен за Original) и нового (ICV и IEst, обозначен за New) методов выделения характерных интервалов для k=5, 7 и 10 соответственно.

Ниже приведены обобщенные результаты для k=5. Для 80% вариантов новый подход отработал не хуже существующего. В разы выросло количество вариантов, на которых получилось достичь хорошей сходимости: с 3 до 13. А число вариантов, на которых сходимость оказалось удовлетворительной, возросло с 12 до 21.

Лучший метод	Количество	В %
New	21	80%
EQL	7	
Original	7	20%
Всего	35	100%

	Original	New
$\|\Delta\| < 5\%$	3	13
$\|\Delta\| < 10\%$	12	21

Ниже приведены обобщенные результаты для k=7. Для 74% вариантов новый подход отработал не хуже существующего. Количество вариантов, на которых получилось достичь хорошей сходимости, выросло в два раза: с 6 до 12. А число вариантов, на которых сходимость оказалось удовлетворительной, возросло с 13 до 20.

Лучший метод	Количество	В %
New	20	74%
EQL	6	
Original	9	26%
Всего	35	100%

	Original	New
$\|\Delta\| < 5\%$	6	12
$\|\Delta\| < 10\%$	13	20

Ниже приведены обобщенные результаты для k=10. Для 74% вариантов новый подход отработал не хуже существующего. В разы выросло количество вариантов, на которых получилось достичь хорошей сходимости: с 3 до 14. А число вариантов, на которых сходимость оказалось удовлетворительной, возросло более чем в два раза: с 11 до 23.

Лучший метод	Количество	В %
New	20	74%
EQL	6	
Original	9	26%
Всего	35	100%

	Original	New
$\|\Delta\| < 5\%$	3	14
$\|\Delta\| < 10\%$	11	23

Также важно отметить, что с помощью нового метода всего за три проведенных запуска (k=5, 7, 10) на 23 вариантах из 35 была получена хорошая сходимость и на 29 вариантах — удовлетворительная. И лишь только на одном варианте (403.gcc.3) существующий метод отработал лучше нового при всех трёх k=5, 7, 10.

По результатам проведенного экспериментального исследования можно сделать однозначный вывод о том, что предложенный метод выделения характерных интервалов исполнения приложения работает лучше существующего и позволяет достичь хорошей/удовлетворительной сходимости на значительно большем (в разы) числе приложений, чем существующий.

6. Апробация решения

У коллег из соседней команды возникла потребность в получении наборов характерных интервалов для приложений из CINT2006 (см. раздел 5.2), собранных с другими опциями компилятора, а именно с использованием битовых операций. Для этого им было предложено воспользоваться ICV-плагином для QEMU, инструментом IEst и его функциональностью по одновременному получению разметок для k от M до N, а также инструментом для визуализации.

Инженерами были получены разметки для k от 1 до 20 для всех 35 вариантов, и для каждого варианта потребовалось лишь один раз записать трассы и собрать реплики. После этого полученные реплики были запущены на FPGA, и была составлена таблица сходимости по всем вариантам и по всем k от 1 до 20.

Для абсолютного большинства вариантов инженеры смогли найти среди 20 полученных наборов характерных интервалов такой, который показывает хорошую сходимость. Более того, чаще всего для варианта было несколько таких наборов. Чтобы выбрать наилучший из них, коллеги воспользовались инструментом для визуализации результатов кластеризации. По итогу было выбрано такое k, для которого построенные кластера лучше всего соответствуют визуально отличимым фазам.

Для 31 из 35 вариантов был получен набор характерных интервалов, демонстрирующий хорошую сходимость, для 33 — удовлетворительную. Теперь соответствующие реплики будут использоваться для симуляции с целью быстрой оценки архитектурных изменений в процессоре. Для получения желаемой сходимости для остальных вариантов, вероятно, необходимо изменить величину интервала.

По результатам апробации решения коллеги отметили, что новый метод точнее и удобнее существующего. Он позволяет многократно ускорить получение наборов характерных интервалов, демонстрирующих приемлемую сходимость. Разработанное ПО будет активно использоваться в дальнейшем с целью получения наборов характерных интервалов для других индустриальных бенчмарков.

Заключение

В рамках дипломной работы были достигнуты следующие результаты.

1. Проведен анализ существующего метода нахождения характерных интервалов, в том числе исследованы способ характеризации интервалов исполнения приложения и алгоритм работы инструмента SimPoint.

2. Разработан новый метод нахождения характерных интервалов исполнения приложения.

3. Разработан инструментарий, реализующий предлагаемый метод, а именно:

   - ICV-плагин для QEMU для сбора статистики по количеству исполненных инструкций по категориям;
   - инструмент IEst для выбора характерных интервалов на основе собранной в QEMU статистики;
   - инструмент для визуализации распределения интервалов исполнения приложения и результатов кластеризации.

4. Проведено экспериментальное исследование для сравнения существующего и нового методов выделения характерных интервалов. Установлено, что предложенный метод работает лучше существующего, а именно позволяет в среднем достичь лучшей сходимости и демонстрирует хорошую/удовлетворительную сходимость на бóльшем (до нескольких раз) числе приложений из выбранного набора.

Еще раз упомянем, что на написанный код действует соглашение о неразглашении (NDA), в связи с чем он не может быть опубликован. Разработанное программное обеспечение было покрыто тестами и прошло код-ревью в компании YADRO.

Список литературы

[1] GitHub-репозиторий инструмента SimPoint. — URL: https://github.com/hanhwi/SimPoint/tree/master (дата обращения: 20 мая 2025 г.).

[2] GitHub-репозиторий с результатами экспериментов. — URL: https://github.com/Danila-Pechenev/thesis-exp-results (дата обращения: 20 мая 2025 г.).

[3] Li Ping, Hastie Trevor, Church Kenneth. Very sparse random projections. — Vol. 2006. — 2006. — 08. — P. 287–296.

[4] SimPoint 3.0: Faster and More Flexible Program Phase Analysis / Greg Hamerly, Erez Perelman, Jeremy Lau, Brad Calder // Journal of Instruction-Level Parallelism. — 2005. — 09. — Vol. 7. — P. 1–28.

[5] Алгоритм G-Means из Python-библиотеки pyclustering. — URL: https://pyclustering.github.io/docs/0.9.2/html/d8/d3c/classpyclustering_1_1cluster_1_1gmeans_1_1gmeans.html (дата обращения: 20 мая 2025 г.).

[6] Алгоритм HDBSCAN из Python-библиотеки scikit-learn. — URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.HDBSCAN.html (дата обращения: 20 мая 2025 г.).

[7] Алгоритм OPTICS из Python-библиотеки scikit-learn. — URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html (дата обращения: 20 мая 2025 г.).

[8] Алгоритм X-Means из Python-библиотеки pyclustering. — URL: https://pyclustering.github.io/docs/0.9.2/html/dd/db4/classpyclustering_1_1cluster_1_1xmeans_1_1xmeans.html (дата обращения: 20 мая 2025 г.).

[9] Алгоритм affinity propagation из Python-библиотеки scikit-learn. — URL: https://scikit-learn.org/stable/modules/generated/

sklearn.cluster.AffinityPropagation.html (дата обращения: 20 мая 2025 г.).

[10] Алгоритм bisecting k-means из Python-библиотеки scikit-learn. — URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.BisectingKMeans.html (дата обращения: 20 мая 2025 г.).

[11] Алгоритм fuzzy C-Means из Python-библиотеки pyclustering. — URL: https://pyclustering.github.io/docs/0.9.0/html/d2/d6a/classpyclustering_1_1cluster_1_1fcm_1_1fcm.html (дата обращения: 20 мая 2025 г.).

[12] Алгоритм k-means из Python-библиотеки scikit-learn. — URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (дата обращения: 20 мая 2025 г.).

[13] Алгоритм mean-shift из Python-библиотеки scikit-learn. — URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html (дата обращения: 20 мая 2025 г.).

[14] Алгоритм spectral clustering из Python-библиотеки scikit-learn. — URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html (дата обращения: 20 мая 2025 г.).

[15] Библиотека NumPy. — URL: https://numpy.org/ (дата обращения: 20 мая 2025 г.).

[16] Индекс Дэвиса-Булдина из Python-библиотеки scikit-learn. — URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.davies_bouldin_score.html (дата обращения: 20 мая 2025 г.).

[17] Лицензия инструмента SimPoint. — URL: https://github.com/hanhwi/SimPoint/blob/master/README.txt (дата обращения: 20 мая 2025 г.).

[18] Метрика силуэт из Python-библиотеки scikit-learn. — URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html (дата обращения: 20 мая 2025 г.).

[19] Модуль pyplot Python-библиотеки matplotlib. — URL: https://pyclustering.github.io/docs/0.9.0/html/d2/d6a/classpyclustering_1_1cluster_1_1fcm_1_1fcm.html (дата обращения: 20 мая 2025 г.).

[20] Набор бенчмарков SPEC CPU2000. — URL: https://www.spec.org/cpu2000/ (дата обращения: 20 мая 2025 г.).

[21] Набор бенчмарков SPEC CPU2006. — URL: https://www.spec.org/cpu2006/ (дата обращения: 20 мая 2025 г.).

[22] Процессорная архитектура RISC-V. — URL: https://riscv.org/about/ (дата обращения: 20 мая 2025 г.).

[23] Результаты итогового сравнения существующего и нового метода выделения характерных интервалов для k=10. — URL: https://github.com/Danila-Pechenev/thesis-exp-results/blob/main/original-new-k10.xlsx (дата обращения: 20 мая 2025 г.).

[24] Результаты итогового сравнения существующего и нового метода выделения характерных интервалов для k=5. — URL: https://github.com/Danila-Pechenev/thesis-exp-results/blob/main/original-new-k5.xlsx (дата обращения: 20 мая 2025 г.).

[25] Результаты итогового сравнения существующего и нового метода выделения характерных интервалов для k=7. — URL: https://github.com/Danila-Pechenev/thesis-exp-results/blob/main/original-new-k7.xlsx (дата обращения: 20 мая 2025 г.).

[26] Результаты сравнения BBV и ICV для k=10. — URL: https://github.com/Danila-Pechenev/thesis-exp-results/blob/main/BBV-ICV-k10.xlsx (дата обращения: 20 мая 2025 г.).

[27] Результаты сравнения BBV и ICV для k=5. — URL: https://github.com/Danila-Pechenev/thesis-exp-results/blob/main/BBV-ICV-k5.xlsx (дата обращения: 20 мая 2025 г.).

[28] Результаты сравнения BBV и ICV для k=7. — URL: https://github.com/Danila-Pechenev/thesis-exp-results/blob/main/BBV-ICV-k7.xlsx (дата обращения: 20 мая 2025 г.).

[29] Результаты сравнения инструментов SimPoint и IEst для k=10. — URL: https://github.com/Danila-Pechenev/thesis-exp-results/blob/main/SimPoint-IEst-k10.xlsx (дата обращения: 20 мая 2025 г.).

[30] Результаты сравнения инструментов SimPoint и IEst для k=5. — URL: https://github.com/Danila-Pechenev/thesis-exp-results/blob/main/SimPoint-IEst-k5.xlsx (дата обращения: 20 мая 2025 г.).

[31] Результаты сравнения инструментов SimPoint и IEst для k=7. — URL: https://github.com/Danila-Pechenev/thesis-exp-results/blob/main/SimPoint-IEst-k7.xlsx (дата обращения: 20 мая 2025 г.).

[32] Симулятор Gem5. — URL: https://www.gem5.org/ (дата обращения: 20 мая 2025 г.).

[33] Эмулятор QEMU. — URL: https://www.qemu.org/ (дата обращения: 20 мая 2025 г.).