Министерство образования и науки Российской Федерации
Московский физико-технический институт (национальный исследовательский университет)

Физтех-школа радиотехники и компьютерных технологий
Кафедра микропроцессорных технологий в интеллектуальных системах управления

Выпускная квалификационная работа бакалавра по направлению подготовки 03.03.01 «Прикладные математика и физика»

Улучшение метода подбора оптимальной конфигурации микроархитектуры процессора с параметрами низкого порядка

Студент Б01–909а группы
Державин А. А.

Научный руководитель
Петушков И. В.

Научный консультант
Лось Д. А.

Долгопрудный
2023

Аннотация

В работе предложен метод подбора оптимальной конфигурации микроархитектуры процессора, позволяющий варьировать большое число параметров малого порядка и эффективнее (по сравнению с существующими подходами) оценивать влияние этих параметров на производительность процессора. Для использования предложенного алгоритма реализована симуляторонезависмая среда моделирования с поддержкой параллельного запуска трасс исполнения. В результате тестовых запусков получено отклонение прироста производительности от максимально возможного (полученного методом полного перебора) на 3.2%, при этом прирост производительности составил 2.22%.

Содержание

Глава 1

Введение

Проектирование современных высокопроизводительных процессоров неразрывно связано с задачей подбора оптимальных параметров микроархитектуры, позволяющих добиться наилучшей производительности. Однако в силу большого числа конфигурационных параметров, формирующих пространство конфигураций; а также длительного времени симуляции на модели производительности процессора, данная задача становится нерешаемой за разумное время методом полного перебора всех точек пространства конфигураций. В связи с этим разрабатываются решения, позволяющие найти оптимальную конфигурацию процессора за меньшее время.

Большинство существующих подходов к решению задачи поиска оптимальной конфигурации микроархитектуры процессора можно разделить на два класса: решения, уменьшающие время одной симуляции и решения, ускоряющие сам алгоритм подбора. Методы первого класса предлагают использовать аппаратные решения [6], а также ряд оптимизаций на программном уровне [15]. Однако при значительном увеличении числа симуляций вследствие увеличения пространства конфигураций, ускорение, полученное данными методами, нивелируется. В то же время методы второго класса предлагают различные способы уменьшения числа симуляций, необходимых для нахождения оптимальной конфигурации, среди них присутствуют подходы с использованием машинного обучения [5, 12, 13], а также RPStacks [14].

В существующих работах, как правило, рассматривается варьирование микроархитектурных параметров высокого порядка (размер кэша, алгоритм предсказания условных переходов). Однако варьирование параметров микроархитектуры низкого порядка (к примеру, использование дополнительно-

го бита в алгоритме предсказания) представляет наибольшую сложность на практике. Изменение одного параметра микроархитектуры низкого порядка не приводит к такому значительному изменению производительности процессора, как при изменении параметра высокого порядка. Однако одновременное изменение большого числа параметров низкого порядка может приводить к существенному приросту производительности процессора. Существующие решения не позволяют эффективно находить оптимальные конфигурации процессора с большим числом параметров низкого порядка. Эмпирические модели машинного обучения, используемые для предсказания производительности процессора, из-за нетривиального характера взаимного влияния параметров низкого порядка требуют большого числа запусков симуляции для обучения. В то время как подходы, основанные на RPStacks, имеют высокую сложность реализации, так как стандартные стеки задержек не покрывают эффективно параметры микроархитектуры низкого порядка.

В данной работе представлен метод подбора оптимальной конфигурации микроархитектуры процессора, основанный на отображении набора статистических данных, получаемых при симуляции, на параметры конфигурации. В представленном методе сравнение статистических значений с пороговыми значениями является триггером для соответствующего изменения параметра конфигурации. Пороговые значения, а также используемые статистические данные определяются на основе запусков инициализации алгоритма подбора.

Алгоритм подбора конфигурации на каждой итерации определяет ряд отклонившихся от пороговых значений статистических данных и запрашивает смену зависимых параметров микроархитектуры. В случае обнаружения падения производительности производится откат изменений и выбирается альтернативное направление следования в пространстве конфигураций, чтобы пропустить рассмотрение конфигураций, приводящих к падению производительности.

Данный подход позволяет более эффективно оценивать влияние параметров микроархитектуры низкого порядка на производительность процессора. Кроме того, как правило, статистические данные, используемые в представленном алгоритме подбора, появляются в больших промышленных и академических симуляторах производительности процессора естественным образом в ходе работы над улучшениями микроархитектуры.

В ходе работы была разработана многопоточная среда исполнения с поддержкой обобщённого интерфейса для симуляторов производительности процессора. В данной среде исполнения был реализован предложенный метод подбора оптимальной конфигурации микроархитектуры процессора. В качестве симулятора производительности использовалась модель суперскалярного процессора с внеочередным исполнением команд архитектуры ARM, основанная на симуляторе gem5 [1]. В результате анализа представленный метод подбора оптимальной конфигурации микроархитектуры процессора показал большую эффективность по сравнению с существующими решениями для подбора конфигураций с большим числом параметром микроархитектуры низкого порядка.

1.1 Постановка задачи

Задачей данной работы является разработка эффективного метода подбора оптимальной конфигурации микроархитектуры процессора с параметрами низкого порядка.

Глава 2

Обзор литературы

При обзоре литературы были изучены различные подходы к решению задачи подбора оптимальной конфигурации микроархитектуры процессора. В начале приводится теоретическое описание задачи, а также выводится полезная формула для оценки времени работы алгоритма, на основании которой в дальнейшем происходит классификация методов решения задачи подбора оптимальных параметров на два типа.

2.1 Теоретическая основа

Задача подбора оптимальной конфигурации микроархитектуры процессора сводится к задаче максимизации функции нескольких переменных в некой области многомерного пространства. В рассматриваемой задаче в качестве максимизируемой функции будет выступать среднее число инструкций исполненных за один такт — Instructions per cycle (далее — IPC) по определению равное

$$\text{IPC} = \frac{\text{число исполненных инструкций}}{\text{число прошедших тактов процессора}}. \quad (1)$$

Откуда нетрудно заметить, что $\text{IPC} > 0$ (неотрицательность IPC очевидна, а неравенство IPC нулю следует из того, что в данной задаче интересно рассмотрение только «реальных» трасс исполнения, содержащих ненулевое число инструкций). В некоторых работах вместо IPC используется среднее число тактов потраченное на одну инструкцию — Cycles per instruction (да-

лее — CPI), которое определяется как

$$\text{CPI} = \frac{\text{число прошедших тактов процессора}}{\text{число исполненных инструкций}} = \frac{1}{\text{IPC}}. \tag{2}$$

В данной задаче IPC рассматривается как функция:

$$\text{IPC} = f\left(x_1, x_2, \ldots, x_n\right).$$

При этом параметры микроархитектуры процессора (далее для простоты изложения могут называться просто параметрами) со значениями $x_1, x_2, \ldots, x_n$, вообще говоря, могут иметь зависимости между собой, обусловленные естественными законами (как например, зависимость числа промахов в кэше от его размера при прочих равных условиях), дополнительными ограничениями на процессор, его стоимостью (нельзя увеличивать все параметры бесконечно), а также иными причинами.

Исходя из вышесказанного, задача формулируется следующим образом:

$$\begin{cases} \text{IPC} = f\left(x_1, x_2, \ldots, x_n\right) \\ G = \left\{\vec{x} \in \mathbb{R}^n \middle| \, a_i \leqslant x_i \leqslant b_i,\ i = \overline{1 \ldots n}\right\} \\ f : G \to \mathbb{R}^+ \qquad\qquad (\text{из } (1)) \end{cases}$$

С дополнительными условиями:

$$\begin{cases} \varphi_i : G \to \mathbb{R},\ i = \overline{1 \ldots n} \\ \sum\limits_{i=1}^{m} \left(\varphi_i\left(x_1, x_2, \ldots, x_n\right)\right)^2 \equiv 0 \end{cases}, \tag{3}$$

где G называется **пространством конфигураций** или **конфигурационным пространством**. Требуется найти (при соблюдении условий (3))

$$\text{IPC}_{optimal} = \max_{\overrightarrow{(x_1, x_2, \ldots, x_n)} \in G} f\left(x_1, x_2, \ldots, x_n\right)$$

Либо же если задано пороговое значение IPC:

$$\text{IPC}_{optimal} = f\left(x_1^{optimal}, x_2^{optimal}, \ldots, x_n^{optimal}\right) \geqslant \text{IPC}_{threshold}$$

Тогда найденные параметры $x_1^{optimal}, x_2^{optimal}, \ldots, x_n^{optimal}$ задают оптимальную конфигурацию процессора.

Данную задачу можно проиллюстрировать следующей схемой (для случая $\mathbb{R}^3$):

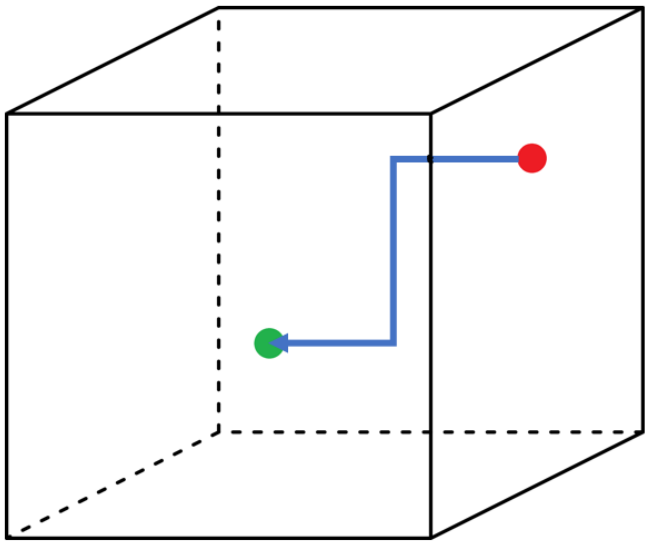

Рис. 1. Схема работы алгоритма подбора параметров

На схеме, изображённой на рис. 1, куб изображает пространство конфигураций, красная точка — начальную точку, отвечающей начальной конфигурации процессора, зелёная точка — желаемую точку, в которой достигается максимальное (либо требуемое) IPC.

Рассмотрим пространство конфигураций более подробно. Обычно параметры микроархитектуры процессора могут принимать лишь дискретный набор значений, следовательно число точек в пространстве конфигураций конечно и может быть вычислено по правилу умножения [18, с. 25]:

$$N_G = \prod_{i=1}^{n} |p_i| \,, \tag{4}$$

где p_i — множество значений параметра с индексом i, $|p_i|$ — его мощность (количество значений параметра с индексом i), при этом очевидно, что $\forall i \;\; x_i \in p_i$.

Пусть Δt — время однократного исполнения трассы на симуляторе, N — число точек конфигурационного пространства, в которых требуется исполнить трассу. Тогда полное время исполнения одной трассы на симуляторе в

N точках пространства конфигураций вычисляется по очевидной формуле:

$$T_N = N \cdot \Delta t. \tag{5}$$

Используя формулы (4) и (5), можно оценить время, требуемое для исполнения одной трассы во всех точках пространства конфигураций (метод полного перебора):

$$T_G = N_G \cdot \Delta t = \prod_{i=1}^{n} |p_i| \cdot \Delta t. \tag{6}$$

Решение методом полного перебора, очевидно, способно найти точку конфигурационного пространства с наибольшим IPC, однако из-за огромного числа параметров современного процессора, а также длительного времени симуляции на моделях производительности, время T_G из (6) становится значительно больше разумных сроков для решения данной задачи. В этой связи разрабатываются подходы, уменьшающие время работы симулятора для нахождения оптимальной конфигурации процессора. Исходя из формулы (5), можно разделить потенциальные методы решения задачи поиска оптимальной конфигурации микроархитектуры процессора на два класса:

- решения, уменьшающие Δt — методы ускорения симуляции
- решения, уменьшающие N — методы ускорения непосредственно алгоритма подбора через уменьшение числа точек пространства конфигураций, в которых требуется запустить симулятор

Перейдём к рассмотрению существующих методов – представителей вышеперечисленных классов.

2.2 Решения, уменьшающие время симуляции

Данный класс решений фокусируется на ускорении работы симулятора. Методы данного класса предлагают использовать аппаратные решения [6], а также ряд оптимизаций на программном уровне [4, 15].

Нетрудно заметить, что с увеличением числа параметров и количества их значений число симуляций N_G сильно растёт, и ускорение, получаемое

методами данного класса, нивелируется, не оказывая заметного влияния на возрастание T_G.

2.3 Решения, ускоряющие алгоритм подбора

2.3.1 CPI stack подходы

В данных подходах используется метод CPI stack, в основе которого лежит идея о разложении среднего числа тактов, потраченных на одну исполненную инструкцию — CPI (2) на независимые слагаемые. Обычно выделяют базовую (обязательную), а также некоторый набор компонент, отражающих такты, «потерянные» из-за таких событий как ошибки предсказания переходов, промахи в кэшах и буферах ассоциативной трансляции (англ. TLB) и т. д. [8]:

$$\begin{aligned} \mathrm{CPI} &= \frac{\text{обязательные такты} + \text{такты на кэш} + \text{такты на TLB} + \ldots}{\text{число исполненных инструкций}} \\ &= \frac{\text{обяз. такты}}{\text{число исп. инструкций}} + \frac{\text{такты кэш}}{\text{число исп. инструкций}} + \ldots \\ &= \mathrm{CPI}_{\text{обяз}} + \mathrm{CPI}_{\text{кэш}} + \mathrm{CPI}_{\mathrm{TLB}} + \ldots \end{aligned} \tag{7}$$

При аккуратном подсчёте, CPI stack позволяет указать на узкие места в конфигурации процессора, сужая множество параметров – кандидатов на изменение. Например, в [14] рассматривается использование CPI stack полученного с помощью анализа критического пути в конвейере процессора. Такой подход позволяет более точно оценить причины потери циклов процессора, однако анализ критического пути является затратной (а также сложной в реализации) операцией, значительно увеличивающей время симуляции. Именно поэтому в [9] предлагается вместо усложнения алгоритма вычисления CPI stack, вычислять сразу несколько CPI stack на разных стадиях конвейера процессора. Данное представление даёт более полную картину о событиях, влияющих на производительность процессора, а также диапазон ожидаемых улучшений производительности процессора в случае ликвидации конкретного события простоя.

2.3.2 Подходы с использованием машинного обучения

Исходя из теоретического описания задачи, представленного в разделе 2.1, можно заключить, что рассматриваемая задача сводится к задаче оптимизации, для решения которой можно использовать методы машинного обучения.

Подходы с использованием машинного обучения [5, 12, 13] предлагают использовать модели, обученные на некотором относительно небольшом подмножестве пространства конфигураций, для предсказания IPC во всех остальных точках пространства. Например, в [13] предлагается решать задачу подбора параметров как задачу линейной регрессии:

$$y = \beta_0 + \sum_{i=1}^{m} \beta_i x_i + \sum_{i=1}^{m} \sum_{j=i+1}^{m} \beta_{i,j} x_i x_j + \sum_{i=1}^{m} \sum_{j=i+1}^{m} \sum_{k=j+1}^{m} \beta_{i,j,k} x_i x_j x_k \ldots + \beta_{1,2,\ldots,m} x_1 x_2 \ldots x_m, \quad (8)$$

где в качестве выходного параметра y выступает CPI (дополнительно проводится исследование с целью выбрать функцию от IPC в качестве выходного параметра, которая обладает наименьшей ошибкой), а подбираемые коэффициенты β играют роль важности соответствующего параметра. В работе также указывается, что для хорошей сходимости и высокой точности решения обычно достаточно не слишком большое множество параметров микроархитектуры процессора — достаточно лишь выделить некое подмножество.

В [12] применяется более сложный подход с использованием моделей нейронных сетей. В работе предлагается модель, которой достаточно обучится на 1% от всего пространства конфигураций, для того чтобы выдавать высокоточные предсказания. Перед передачей параметров процессора нейронной сети, производится их нормировка в интервал $(0, 1)$ с помощью min max scaling [11, с. 114], включая категориальные параметры (для них используется метод one-hot encoding [3]). Далее происходит тренировка некого набора нейронных сетей с применением перекрёстной валидации [17]. Также стоит отметить, что каждая точка конфигурационного пространства из тренировочного набора подается сетям с частотой, пропорциональной CPI в этой точке.

В противовес предыдущему решению, в [5] описывается несколько иной подход с использованием машинного обучения, решая вместо классической задачи нахождения экстремального значения, задачу классификации. Данная задача состоит в том, чтобы выделить из двух предоставленных конфигураций микроархитектуры процессора «лучшую». Такие попарные сравнения позволяют без дополнительных расходов на симуляцию увеличить размер тренировочного множества с n до $\frac{n\cdot(n-1)}{2} = \mathcal{O}\left(n^2\right)$, что существенно улучшает точность предсказания.

В [7] сообщается о двух недостатках, которыми обладают некоторые вышеописанные модели:

- При смене тестовых наборов требуется заново собирать и обучать модель
- Для обучения существующих предсказателей обычно требуется большое число симуляций

Предлагаемая модель фокусируется на поведении самой микроархитектуры. В основе метода лежит идея о разложении нового пространства, представляющего новую программу на на линейную комбинацию пространств тестовых программ, где в роли метрики выступает производительность. На рис. 2 изображена схема предлагаемого решения. Решение работает следующим образом. Сначала происходит обучение N моделей, каждая на своём приложении, далее при для предсказания поведения новой программы, собирается результат малого числа R симуляций новой программы, а также результаты предсказаний моделей с этих симуляций, после чего используется метод линейной регрессии в следующем виде:

$$\overrightarrow{y} = \beta_0 + \sum_{i=1}^{m} \beta_i \cdot \overrightarrow{X}_i,$$

где $\overrightarrow{X}_j$ — вектор из R ответов от модели j, $\overrightarrow{y}$ — вектор, состоящий из ответов новой программы на R симуляциях. Таким образом, для получения предсказания для новой программы на очередной точке конфигурационного пространства, достаточно подать R новых точек пространства на обученные предсказатели (модели), получая тем самым нужные R ответов.

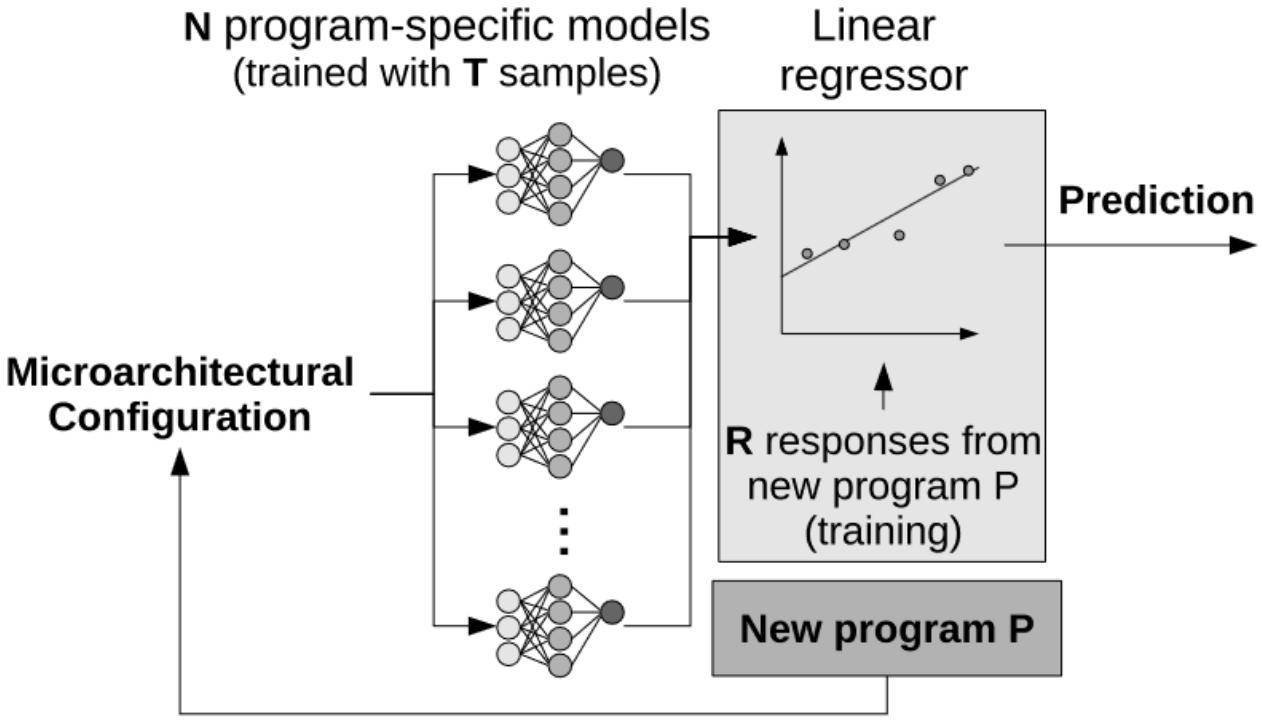

Рис. 2. Схема алгоритма [7]

Перечисленные методы позволяют значительно сократить число симуляций N.

2.4 Анализ решений

В существующих работах, как правило, рассматривается варьирование параметров высокого порядка[1], такие как размер кэша, алгоритм предсказания условных переходов и т. п. В то время как наибольшую сложность на практике представляет варьирование параметров низкого порядка (дополнительный бит в алгоритме предсказания, дополнительный бит тэга).

С одной стороны, исходя из определения, изменение одного параметра микроархитектуры низкого порядка не может привести к такому значительному изменению производительности процессора, как при изменении одного параметра микроархитектуры высокого порядка. С другой стороны, при одновременном изменении нескольких параметров микроархитектуры низкого порядка, можно добиться существенного прироста производительности процессора.

[1]Параметры, небольшое изменение которых приводит к значительному изменению производительности

Существующие решения не позволяют эффективно находить оптимальные конфигурации микроархитектуры процессора при рассмотрении большого числа параметров низкого порядка:

- Подходы основанные на RpStacks имеют высокую сложность реализации, в силу того, что стандартные данные о задержках, предоставляемые разработчиками симуляторов, не покрывают эффективно параметры микроархитектуры низкого порядка.
- Эмпирические модели машинного обучения, которые предлагается использовать для предсказания производительности процессора, из-за нетривиального характера взаимного влияния параметров низкого порядка требуют намного больше данных для обучения, что выражается в большем числе запуска симулятора, отрицательно сказываясь на времени работы

Глава 3

Методология

В данной главе приводится описание предлагаемого решения. Сначала описывается непосредственно алгоритм подбора параметров микроархитектуры процессора, логика его работы. Далее приводится обзор среды моделирования, на которой производились запуски алгоритма подбора. В конце главы рассматривается инфраструктура, позволяющая упростить работу среды моделирования со статистикой.

3.1 Описание алгоритма подбора

Предлагаемый алгоритм подбора оптимальной конфигурации основан на отображении набора статистических данных, получаемых в результате симуляции, на параметры конфигурации процессора. В рассматриваемом алгоритме используются статистические данные, которые, как правило, появляются в больших промышленных и академических симуляторах производительности естественным образом в ходе работы над улучшениями микроархитектуры; либо несложным образом выражаются через стандартные статистические данные.

Каждому статистическому значению сопоставляется определённое множество параметров (непосредственно влияющих на данное статистическое значение) процессора, которым, в свою очередь, сопоставляются наборы возможных (варьируемых) значений. Упомянутые наборы определяют **пространство конфигураций**.

На каждой итерации алгоритма выполняются следующие действия:

1. Запустить симулятор, собрать и усреднить статистические данные по трассам исполнения

2. Вычислить изменение IPC по сравнению с предыдущей итерацией

3. Если изменение отрицательное, произвести откат последнего изменения параметра и исключить зависимую статистику из списка кандидатов до тех пор, пока переходом к другой точке пространства конфигураций не будет получено положительное изменение IPC (рис. 3)

4. Выбрать из списка кандидатов имеющее наибольшее относительное отклонение от порогового[1] статистическое значение

5. Запросить изменение параметра конфигурации, от которого зависит выбранное статистическое значение

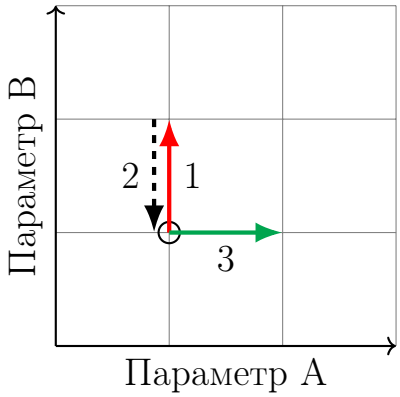

Рис. 3. Принцип работы отката изменения

Описанный выше порядок действий повторяется до тех пор, пока не выполнится одно из условий:

- Достижение требуемого улучшения производительности (если было задано)
- Исчерпание возможных направлений модификаций
- Превышение лимита числа итераций (если был задан)

[1]Подбираются на основе предварительных инициализирующих запусков

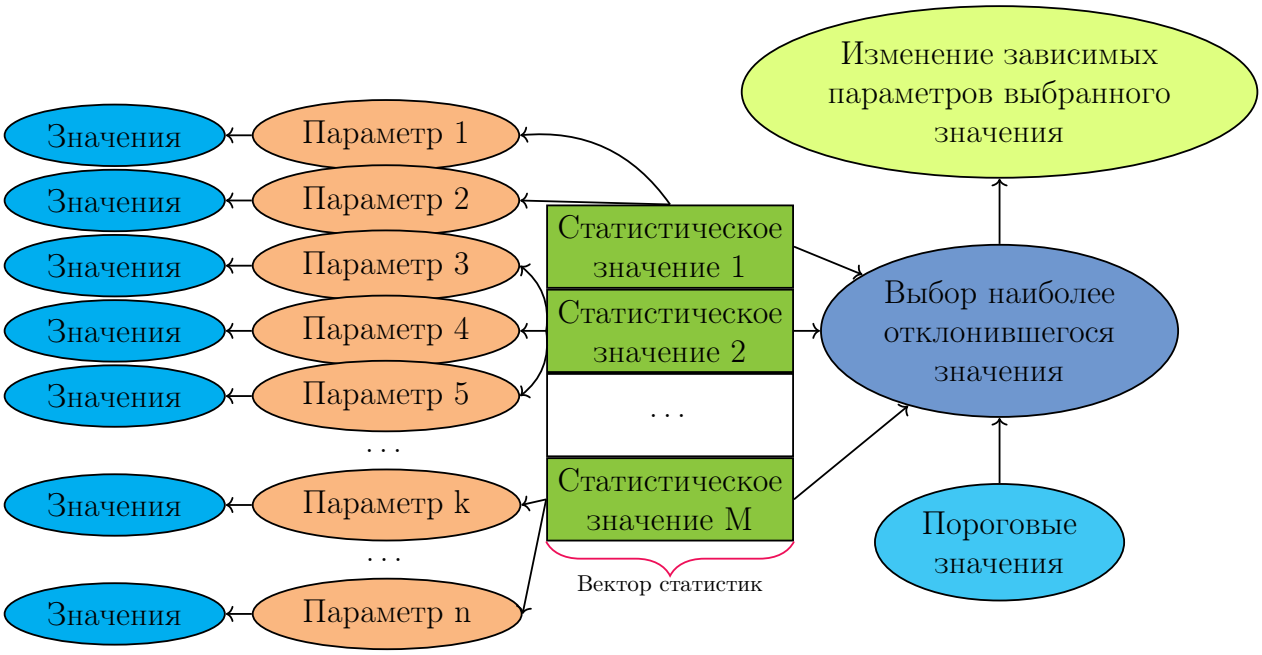

Рис. 4. Схема работы алгоритма

3.2 Обзор среды моделирования

Для тестирования предложенного алгоритма требовалось разработать среду моделирования, которая бы позволяла взаимодействовать с симулятором (например, для изменения параметров), а также производила сбор статистики полученной с трасс исполнения.

В силу большого разнообразия симуляторов, ориентированных под различные нужды, и как следствие значительных отличий в организации работы с ними, была поставлена задача сделать среду моделирования **симуляторонезависимой** — способной выступать в роли связующего звена между конкретным симулятором и алгоритмом, не вмешиваясь в работу последнего. Кроме того, из-за большого числа трасс исполнения в тестовых наборах приложений, было необходимо, чтобы среда моделирования поддерживала запуск нескольких трасс исполнения в параллельных потоках для уменьшения время работы алгоритма и лучшей утилизации вычислительных ресурсов кластерных систем, на которых и предполагается запуск реализованного алгоритма.

Схема работы реализованной среды моделирования представлена на рис. 5.

Далее представлено описание элементов схемы.

- **Обобщённый интерфейс** позволяет абстрагироваться от конкретно-

го симулятора, обеспечивая взаимодействие с ним в обобщённом виде. Для добавления поддержки нового симулятора достаточно лишь описать ключевые моменты в работе с ним (процесс запуска, сбор статистики, изменение параметров), при этом не требуется вносить изменения в остальную («алгоритмическую») часть среды моделирования.

- **Алгоритм подбора** через обобщённый интерфейс получает от симулятора статистические данные запуска трасс исполнения. Далее происходит усреднение статистических данных, а также прироста IPC по трассам исполнения. Для того, чтобы получить осмысленное среднее значение прироста IPC, производится предварительная нормировка каждого значения IPC на значение на начальной итерации (под номером 0) по формуле

$$\mathrm{IPC}_{ij}^{norm} = \frac{\mathrm{IPC}_{ij}}{\mathrm{IPC}_{0j}},$$

где i — номер итерации, j — номер трассы исполнения. Тогда относительный прирост IPC на итерации i для трассы j может быть вычислен как

$$\Delta\mathrm{IPC}_{ij} = \mathrm{IPC}_{ij}^{norm} - 1.$$

В итоге среднее значение относительного прироста IPC вычисляется как

$$\langle \Delta\mathrm{IPC}_i \rangle = \sqrt[M]{\prod_{j=1}^{M} \Delta\mathrm{IPC}_{ij}}$$

При этом среднее значение статистического значения вычисляется по формуле

$$\langle \mathrm{Stat}_i \rangle = \frac{\sum_{j=1}^{M} \mathrm{Stat}_{ij}}{M},$$

где Stat_{ij} (по аналогии с IPC) есть статистическое значение с именем Stat, полученное с трассы исполнения с номером j на i-ой итерации.

На основании анализа статистики, через обобщённый интерфейс осуществляется запрос изменения выбранного зависимого параметра.

- **Кластер** (либо иная среда с возможностью параллельного исполнения) осуществляет параллельный запуск трасс исполнения на симуляторе.

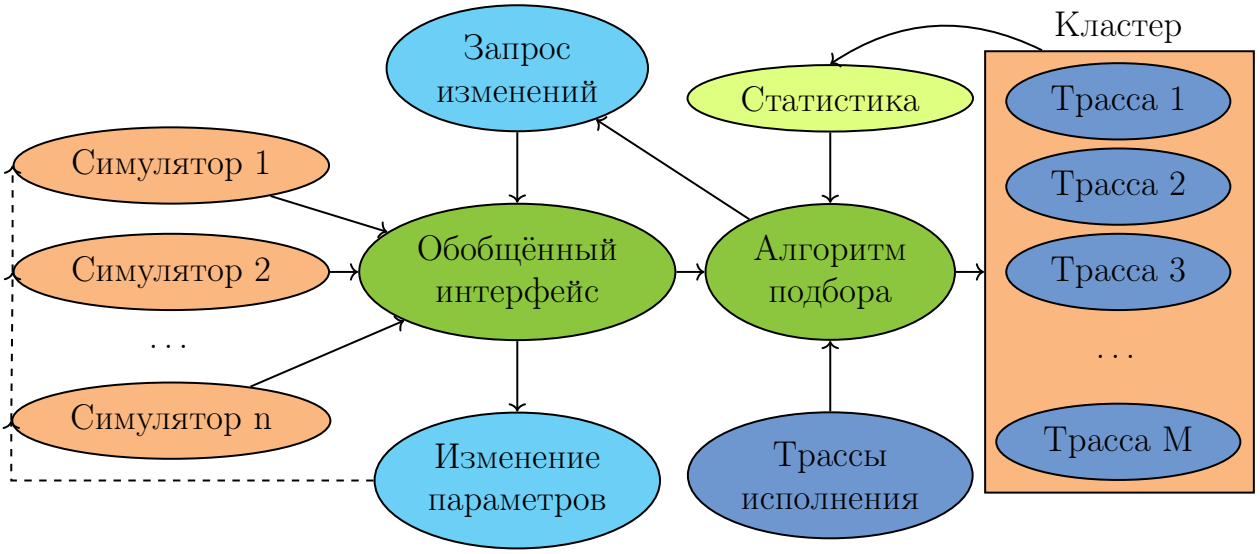

Рис. 5. Схема работы среды моделирования

В среде моделирования реализована поддержка двух симуляторов производительности: gem5 [1] и ChampSim [10].

Для валидации работы алгоритма был добавлен режим полного перебора, в котором осуществляется проход по всем точкам конфигурационного пространства с запуском на них симулятора для нахождения максимально возможного IPC. По этой причине при выборе параметров для варьирования требовалось ограничиться небольшим числом параметров, чтобы алгоритм полного перебора успел просчитать все точки конфигурационного пространства за разумное время.

3.3 Работа со статистикой в обобщённом виде

В силу того, что формат хранения статистических данных сильно отличается от симулятора к симулятору (начиная с разных форматов файлов статистики и заканчивая различными названиями статистических данных и параметров), требовалось спроектировать инфраструктуру, которая бы обеспечила единообразный подход к анализу статистики с точки зрения алгоритма подбора.

Каждый симулятор сопровождается файлом в формате JSON [2], который называется **файлом отображений** (англ. mappings file).

В листинге 1 представлен пример такого файла. В нём можно заметить две основные секции: определение множества варьируемых параметров с их

Листинг 1 Пример файла отображений

```
{
  "statistics->parameters": {
    "sim.StatisticValue": {
      "parameter1": {
        "values": [
          "first",
          "second",
          "third"
        ]
      },
      "parameter2": {
        "values": [
          1,
          2,
          3
        ]
      }
      // Остальные параметры
    },
    "sim.AnotherStatisticValue": {
      "parameter3": {
        "values": [
          [
            1,
            2,
            3
          ],
          [
            4,
            5,
            6
          ]
        ]
      }
    }
    // Остальные статистические данные
  },
  "statistics->generalized": {
    "sim.StatisticValue": "Generalized.StatisticValue",
    "sim.AnotherStatisticValue": "Generalized.AnotherStatisticValue"
    // Соотношения названий остальных статистик
  }
}
```

отображением на статистические данные (по ключу `"statistics->parameters"`), а также соотнесение названий статистических данных симулятора с их обобщёнными аналогами (по ключу с именем `"statistics->generalized"`). Кроме того, в листинге 1 представлены все поддержанные варианты задания значений для параметров: строка, целое число, либо список целых чисел.

В ходе работы алгоритма, статистические значения (в обобщённом виде) сравниваются с пороговыми значениями, полученными в результате инициализирующих запусков симулятора. Они определены в **файле пороговых значений** (англ. thresholds file) в формате JSON [2]. Пример такого файла представлен в листинге 2.

Листинг 2 Пример файла пороговых значений

```
{
  "Generalized.StatisticValue": 42
  // Пороговые значения для остальных статистик
}
```

Описанная выше инфраструктура позволяет алгоритму подбора оперировать исключительно обобщённой статистикой.

Глава 4

Результаты

В качестве симулятора производительности использовалась модель суперскалярного процессора с внеочередным исполнением команд архитектуры ARM [16], основанная на симуляторе gem5 [1]. В ходе работы в симулятор был добавлен подсчёт статистики для работы алгоритма подбора.

Запуски осуществлялись с помощью реализованной среды моделирования в режиме параллельного исполнения на наборе тестовых 3D – приложений.

Перед запуском была проведена предварительная обработка данных, с целью выделения групп (кластеров) приложений со схожими «горячими»[1] статистическими значениями. Для улучшения результатов кластеризации, была проведена нормализация данных. В результате предварительной обработки данных, было выделено два кластера, по которым в дальнейшем и производилось усреднение статистических значений.

Для оценки результата работы предложенного алгоритма, были проведены запуски в двух режимах:

- Обычный режим (поиск оптимальной конфигурации с помощью предложенного алгоритма)
- Режим полного перебора — обход всех точек пространства конфигураций с нахождением максимально возможного IPC

В результате вышеописанных запусков, был получен прирост производительности, в среднем отклоняющийся на 3.2% от оптимального прироста производительности (полученного методом полного перебора). Среднее улучшение производительности составило 2.22%, причём для достижения этого

[1]Значения, имеющие наибольшее относительное отклонение от своих пороговых значений

результата потребовалось в среднем обойти всего 14% от общего числа точек пространства конфигураций.

Полученный результат позволяет сделать вывод о том, что предложенный алгоритм способен успешно подбирать оптимальную конфигурацию процессора с производительностью, незначительно отличающейся от максимально возможной на заданном конфигурационном пространстве. Кроме того, алгоритму требуется значительно меньшее число точек конфигурационного пространства для обнаружения точки с оптимальной производительностью.

Глава 5

Заключение

В работе был предложен алгоритм оптимального подбора параметров, основанный на идее отображения статистических данных, полученных с симулятора, на параметры процессора.

Для использования и тестирования алгоритма была разработана среда моделирования с поддержкой обобщенного интерфейса работы с симуляторами. Данная среда моделирования также обладает возможностью запуска трасс исполнения в параллельном режиме.

С помощью тестовых запусков было выяснено, что предложенный алгоритм позволяет находить оптимальную конфигурацию микроархитектуры процессора через варьирование параметров низкого порядка.

В результате работы алгоритма на тестовом наборе приложений было получено среднее увеличение производительности на 2.22%, при этом отклонение этого значения от максимально возможного прироста производительности, полученного методом полного перебора, составило 3.2%.

Список литературы

[1] Nathan Binkert и др. «The gem5 simulator». *ACM SIGARCH computer architecture news* **39** 2 (2011), с. 1—7.

[2] Tim Bray. «The javascript object notation (json) data interchange format» (2014).

[3] Jason Brownlee. «Why one-hot encode data in machine learning». *Machine Learning Mastery* (2017), с. 1—46.

[4] Trevor E Carlson, Wim Heirman, Lieven Eeckhout. «Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation». *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 2011, с. 1—12.

[5] Tianshi Chen и др. «Archranker: A ranking approach to design space exploration». *ACM SIGARCH Computer Architecture News* **42** 3 (2014), с. 85—96.

[6] Derek Chiou и др. «Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators». *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE. 2007, с. 249—261.

[7] Christophe Dubach, Timothy M Jones, Michael FP O'Boyle. «An empirical architecture-centric approach to microarchitectural design space exploration». *IEEE Transactions on Computers* **60** 10 (2010), с. 1445—1458.

[8] Stijn Eyerman и др. «A performance counter architecture for computing accurate CPI components». *ACM SIGPLAN Notices* **41** 11 (2006), с. 175—184.

[9] Stijn Eyerman и др. «Multi-stage CPI stacks». *IEEE Computer Architecture Letters* **17** 1 (2017), с. 55—58.

[10] Nathan Gober и др. «The Championship Simulator: Architectural Simulation for Education and Competition». *arXiv preprint arXiv:2210.14324* (2022).

[11] Jiawei Han, Micheline Kamber, Jian Pei. «Data mining concepts and techniques third edition». *University of Illinois at Urbana-Champaign Micheline Kamber Jian Pei Simon Fraser University* (2012).

[12] Engin Ïpek и др. «Efficiently exploring architectural design spaces via predictive modeling». *ACM SIGOPS Operating Systems Review* **40** 5 (2006), с. 195—206.

[13] PJ Joseph, Kapil Vaswani, Matthew J Thazhuthaveetil. «Construction and use of linear regression models for processor performance analysis». *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.* IEEE. 2006, с. 99—108.

[14] Jaewon Lee, Hanhwi Jang, Jangwoo Kim. «Rpstacks: Fast and accurate processor design space exploration using representative stall-event stacks». *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture.* IEEE. 2014, с. 255—267.

[15] Jason E Miller и др. «Graphite: A distributed parallel simulator for multicores». *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture.* IEEE. 2010, с. 1—12.

[16] David Seal. *ARM architecture reference manual.* Pearson Education, 2001.

[17] Mervyn Stone. «Cross-validation: A review». *Statistics: A Journal of Theoretical and Applied Statistics* **9** 1 (1978), с. 127—139.

[18] Станислав Окулов. *Дискретная математика. Теория и практика решения задач по информатике.* Litres, 2014.