ДИНАМИЧЕСКИЙ АНАЛИЗ ПРОГРАММНОГО КОДА КАК ИНСТРУМЕНТ ОПТИМИЗАЦИИ АРХИТЕКТУРЫ АППАРАТНЫХ УСКОРИТЕЛЕЙ

Трубач К.И., студент

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Иванюк А.А. – д-р техн. наук, профессор

В работе предложен метод оптимизации аппаратных ускорителей на основе динамического анализа кода. В отличие от статического подхода, он позволяет адаптировать систему команд под реальные рабочие нагрузки и исключить избыточные аппаратные блоки. Эффективность метода продемонстрирована на примере ускорителя для расчета характеристик ФНФ, что обеспечивает создание более компактных и энергоэффективных процессорных архитектур.

Современная микроэлектроника характеризуется стремлением к максимальной энергоэффективности и производительности при ограниченных ресурсах кристалла. Применение универсальных ядер для решения узкоспециализированных задач часто приводит к избыточности аппаратных ресурсов. Статический анализ [1] позволяет определить потенциально необходимые ресурсы, однако он не всегда учитывает реальные сценарии выполнения программы, где определенные ветви кода могут исполняться крайне редко или зависеть от специфических входных данных.

Целью настоящей работы является формализация процесса динамического анализа программного кода для уточнения требований к архитектуре аппаратного ускорителя. Динамический анализ позволяет перейти от теоретической оценки «худшего случая» к адаптивной конфигурации аппаратного обеспечения, которая оптимизирована под реальные рабочие нагрузки. Он предполагает выполнение скомпилированного машинного кода на эмуляторе или виртуальной платформе с целью сбора трассировочных данных. В отличие от статического анализа, который оперирует структурой исходного кода, динамический анализ фокусируется на потоке управления и потоке данных во время исполнения, и включает формирование наборов входных данных для максимально широкого покрытия входных векторов, включая граничные и случайные значения; сбор статистики по каждому выполненному переходу и каждой операции; выявления цепочек вычислений, определяющих критический путь выполнения алгоритма; построение профиля использования ресурсов. Выходными данными динамического анализа является набор данных о ветвлениях в алгоритме, о частоте использования инструкций, о цепочках связанных инструкций и прочие.

Одним из применений специализированный аппаратных ускорителей является управление аппаратными блоками физически неклонируемых функций (ФНФ) [2], а также расчёт их характеристик. На рисунке 1 представлен алгоритм, заданный на псевдоязыке, для расчёта характеристики уникальности (UNQ) для *i*-той и *j*-той ФНФ, а также аппаратура, синтезированная для этого алгоритма на основе статического анализа. Функция *STA_ch(CHk, E, i)* введена для сокращения псевдокода и рассчитывает стабильность ФНФ при многократном повторении (2^E раз) запроса к ФНФ.

Input: N, E, i, j
Output: UNQ_{ij}

1 $resp_{equal} \leftarrow 0$
2 **for** k := 1 to 2^N **do**
3 $resp_{accum} \leftarrow 0$
4 sta, $resp_{last} \leftarrow STA\_ch(CH_k, E, i)$
5 **if** (sta == 1) **then**
6 $resp_{accum} \leftarrow resp_{accum} + 1$
7 sta, $resp_{last} \leftarrow STA\_ch(CH_k, E, j)$
8 **if** (sta == 1) **then**
9 $resp_{accum} \leftarrow resp_{accum} + 1$
10 **if** ($resp_{accum}$ == 2) **then**
11 $resp_{equal} \leftarrow resp_{equal} + 1$
12 **end for**
13 $resp_{equal} \leftarrow 1 - (resp_{equal} >> N)$
14 **return** $resp_{equal}$

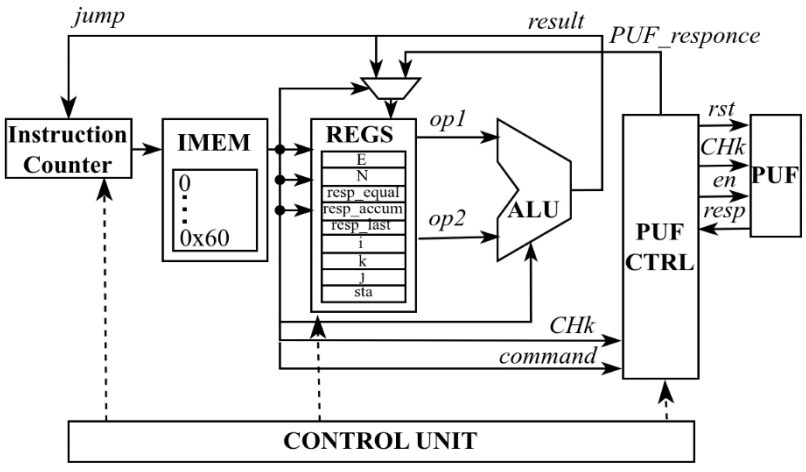

Рисунок 1 – Алгоритм расчёта уникальности и структура аппаратного ускорителя, реализующего его

Статический анализ показал, что аппаратура должна уметь выполнять следующие инструкции и операции: сдвиги влево на единицу и значение регистра, инструкция модуля (эта и перечисленные ранее операции специфичны для функции *STA_ch(CHk, E, i)*), сложение и вычитание над регистрами и константами, сдвиг вправо на значение регистра, инструкции декремента и инкремента, сравнение с единицей, сравнение с двойкой, вычитание регистра от единицы, а также ветвления. Некоторые из этих инструкций нестандартны для существующих архитектур, поскольку представляют собой операции, полученные из нескольких простых инструкций и специфичные именно данному алгоритму. Зачастую составные инструкции, если они вызываются малое количество раз по сравнению с другими инструкциями, можно развернуть в набор простых операций без значительной потери производительности, что упростит процесс декодирования в процессорном ядре и выполнение микроопераций АЛУ.

Ответить на вопрос, какие специальные инструкции можно упростить, помогает динамический анализ. Его целью является изменение аппаратуры для достижения оптимального её использования в типичном запуске. Для алгоритма, представленного выше, параметры *N* и *E* являются уточняющими (то есть от их роста зависит точность выходного значения $resp_{equal}$, а параметры *i*, *j*, индексирующие ФНФ, – определяющими крайние случаи, что связано с вероятностью получить стабильный ответ от конкретной ФНФ. Согласно алгоритму выше, от стабильности пар запрос-ответ двух ФНФ зависит выполнение или невыполнение инструкций в условных блоках if. Если для данного запроса обе ФНФ нестабильны, не будет выполнен ни один такой блок; если хотя бы одна из ФНФ нестабильна, будет выполнен только один условный блок; и наконец, если обе ФНФ стабильны, будут выполнены все условные блоки. Таким образом, проведение динамического анализа желательно на трёх случаях, для каждого из которых одним из результатов динамического анализа становится частота встречаемости инструкций (выражаемая в процентах) при динамическом исполнении микрокода в случае стабильных ФНФ (F_{fs}), в случае частично стабильных ФНФ (F_{ps}) и в случае нестабильных ФНФ (F_{ns}). В таблице 1 представлены избранные инструкции алгоритма, наиболее ярко демонстрирующие особенности изменения процента своего использования в алгоритме в зависимости от стабильности ФНФ.

Таблица 1 – Частота встречаемости избранных инструкций в зависимости от стабильности ФНФ

Инструкция	F_{ns}, %	F_{ps}, %	F_{fs}, %
Сложение двух регистров (add)	20,0466	20,1738	20,2016
Вычитание регистра от единицы (one_sub)	0,0002	0,0002	0,0001
Переход по адресу, если два сравниваемых регистра не равны (bne)	11,1302	11,1170	10,0038
Переход по адресу, если регистр не содержит единицу (bne_one)	0,0002	0,0432	9,9648
Переход по адресу, если регистр не содержит двойку (bne_two)	0	0,0216	0,0389
Взятие абсолютного значения регистра (abs)	11,0869	11,0737	9,9649
Сброс регистра в значение нуля (reset)	0,1269	0,1299	0,1196
Арифметический сдвиг регистра вправо (shr)	0,0002	0,0002	0,0001

Инструкция add является примером тех инструкций, которые составляют большую часть потока управления. Они используются во внутреннем цикле функции *STA_ch(CHk, E, i)*), и их замена на ряд простых инструкций будет крайне негативно сказываться на общей производительности алгоритма. Специализированная инструкция вычитания из единицы (one_sub) встречается незначительно редко вне зависимости от входных данных, и является хорошим кандидатом на удаление из архитектуры набора инструкций (instruction set architecture, ISA) путём замены на комбинацию инструкций сложения с константой (addi) и вычитания (sub). Такой же подход хорошо применим для сброса регистра в значение нуля (zero): её можно заменить использованием инструкции add в сочетании с аппаратным регистром-константой логического нуля. При этом несмотря на то, что инструкция сдвига вправо (shr) встречается так же редко, как и инструкция сброса регистра, она является незаменимой и удалению из ISA не подлежит. Инструкции ветвления при сравнении регистров (bne), сравнении регистра с единицей (bne_one) и сравнения регистра с двойкой (bne_two) хорошо демонстрируют описанную ранее связь между условными конструкциями и стабильностью ФНФ. Из таблицы 1 видно, что инструкция bne_one участвует редко, но она может дать существенный прирост производительности в случае стабильных ФНФ, и убирать её из ISA нежелательно.

Динамический анализ позволяет оптимизировать архитектуру ускорителей и адаптировать систему команд под реальные нагрузки и исключить избыточность. Это влечёт к пересмотру и изменению существующей архитектуры, но обеспечивает создание более компактных и энергоэффективных ядер.

Список использованных источников:

1. Kaiky M., IVANIUK A., ZALIVAKA S. Methodology for constructing statically reconfigurable soft-processor cores for hardware acceleration of algorithms : пат. 12204882 США. – 2025.

2. Иванюк А.А. Исследование физически неклонируемой функции конфигурируемого кольцевого осциллятора. Информатика. 2025, т. 22, № 1, с. 73–89. DOI: https://doi.org/10.37661/1816-0301-2025-22-1-73-89.