Методы синтеза промежуточных представлений программ для высокоуровневого проектирования специализированных вычислителей

П.Н. Советов

***Аннотация*—В статье рассматриваются методы автоматизации построения инструментального программного обеспечения без явного описания архитектуры на ранних этапах проектирования специализированных аппаратных вычислителей. Представлена формальная модель архитектурно-независимого промежуточного представления программы, а также метод синтеза архитектурно-зависимого представления на основе анализа структурной избыточности и подобия графов целевых алгоритмов. Полученное архитектурно-зависимое представление в виде графа архитектуры может использоваться для автоматической генерации программного симулятора и компилятора. Разработан метод синтеза специализированных команд, включающий генерацию полей литералов и семантическую фильтрацию множества синтезируемых инструкций. Кроме того, изложен метод построения архитектур реконфигурируемых и программируемых вычислителей на основе задачи слияния структурно подобных графов алгоритмов с обобщением ресурсов. Данная задача сведена к модели программирования в ограничениях, что гарантирует отсутствие нежелательных комбинационных циклов в аппаратной реализации. Апробация предложенных методов на алгоритмах криптографии (AES, CRC32, Magma, SHA256) подтвердила возможность автоматического исследования пространства архитектурных решений для обеспечения необходимого компромисса между затратами аппаратных ресурсов и производительностью вычислителя.**

***Ключевые слова*—специализированные вычислители, промежуточное представление программ, синтез системы команд, слияние трактов данных.**

I. Введение

Развитие вычислительных систем в условиях замедления действия закона Мура и исчерпания потенциала масштабирования тактовых частот (завершение действия закона Деннарда) характеризуется переходом от универсальных решений к глубокой специализации. Архитектурная оптимизация под конкретные классы задач становится одним из важных источников повышения производительности и энергоэффективности. Д. Паттерсон и Дж. Хеннесси характеризуют этот этап как «новый золотой век компьютерной архитектуры», связанный с массовым внедрением предметно-ориентированных архитектур.

В отличие от универсальных процессоров, где значительная часть энергопотребления и времени тратится на выборку команд, декодирование, предсказание переходов и работу кэш-памяти, специализированные вычислители (СВ) сокращают накладные расходы на универсальность вычислений. Высокие показатели энергоэффективности и уменьшение занимаемой площади кристалла для СВ, по сравнению с универсальными решениями, достигается, в частности, за счет использования массового пространственного параллелизма, глубокой конвейеризации вычислений, специализации подсистемы памяти и применения специализированных типов данных нестандартной разрядности.

СВ широко применяются в задачах из области криптографии, сетевой маршрутизации и безопасности, цифровой обработки сигналов, машинного обучения и других. При этом для достижения технологического суверенитета необходимо сокращать сроки и улучшать результаты проектирования СВ. Это особенно актуально для СВ, реализуемых на ПЛИС, а также для СВ, представляющих собой набор специализированных команд для расширяемых процессорных архитектур, таких как RISC-V.

В условиях ограниченного доступа к передовым производственным технологиям возникает необходимость разработки методов синтеза, обеспечивающих адаптацию архитектурных моделей СВ к различным технологическим реализациям. Требуются инструменты, позволяющие адаптировать архитектуру как под доступные ПЛИС, так и под технологические нормы отечественных производителей микроэлектроники (СБИС), компенсируя технологическое отставание элементной базы за счет глубокой структурной оптимизации архитектуры СВ. В условиях дефицита узких специалистов, инженеров-проектировщиков СВ, необходимо сократить семантический разрыв между описанием целевых алгоритмов на языках высокого уровня, таких как Python, и низкоуровневым проектированием аппаратуры.

Статья получена 20 апреля 2026.
П. Н. Советов, МИРЭА – Российский технологический университет, Москва, Россия (e-mail: sovetov@mirea.ru).

Рис. 1. Индуктивный метод синтеза промежуточных представлений для СВ

Традиционно сначала проектируется СВ, а затем создается инструментальное программное обеспечение (ПО) для него, в том числе компилятор и симулятор. Такой подход требует длительного цикла разработки и не дает разработчику компилятора необходимую обратную связь с проектировщиком аппаратуры.

Перспективным является направление совместного проектирования программной и аппаратной части (HW/SW Co-design) СВ, но известные подходы на этой основе обладают рядом недостатков:

1. Высокоуровневый синтез (HLS) [1]. Как правило, генерирует вычислители с жестко заданной функциональностью и ориентирован на C/C++. Во многих случаях генерирует решения недостаточно высокого качества в силу универсальности подхода, а также использует характеристики макроячеек и микросхем конкретных зарубежных производителей.
2. Решения на основе языков описания архитектуры (ADL) [2] для программируемых СВ. Требуют ручного и детального описания микроархитектуры.

Таким образом, имеется необходимость в разработке методов и алгоритмов автоматизации построения инструментального ПО без явного описания архитектуры, а также во время проектирования или даже до начала проектирования СВ различной степени специализации.

В связи с этим, целью данной статьи является представление методов и алгоритмов автоматического синтеза промежуточных представлений, использующихся компилятором, для построения инструментального ПО вычислителей различной степени специализации.

Статья организована следующим образом. Во втором разделе вводится формальная модель архитектурно-независимого промежуточного представления с явным параллелизмом, в качестве базиса для автоматически синтезируемых архитектурно-ориентированных промежуточных представлений. В третьем разделе описываются методы синтеза специализированных команд с точки зрения словарного графового сжатия для заданного множества целевых алгоритмов с учетом архитектурных ограничений. В четвертом разделен рассматривается метод индуктивного синтеза реконфигурируемых и программируемых ВС на основе устранения структурной избыточности. В пятом разделе представлены результаты апробации предложенных методов.

II. Метод синтеза промежуточных представлений

В рамках статьи рассматриваются следующие основные классы архитектур СВ:

1. Конвейер с фиксированной вычислительной структурой. Обеспечивает максимальную производительность для неизменяемых алгоритмов.
2. Реконфигурируемый конвейер. Позволяет в ограниченных пределах изменять схему потоков данных, адаптируясь под различные алгоритмы за счет коммутации функциональных узлов. В конвейерную структуру вводятся мультиплексоры. Конфигурация задается перед выполнением алгоритма.
3. Специализированный процессор. Обладает гибкостью программного управления при наличии специализированных тракта данных и системы команд. Имеется программа, регистровый файл и счетчик команд.

Метод синтеза промежуточных представлений в наиболее общем виде представлен в виде схемы на рис. 1.

Индуктивный синтезатор принимает на вход множество реализаций целевых алгоритмов, которые необходимо поддерживать в СВ. Эти реализации могут быть, в частности, написаны на предметно-ориентированном языке, встроенном в Python (Python/DSL). Кроме того, на вход подается множество

архитектурных ограничений, определяющих класс архитектур СВ и аппаратные особенности внутри этого класса.

Класс входных программ ограничен алгоритмами со статическим потоком управления и данных, представимыми в виде ациклических графов вычислений без динамического выделения ресурсов.

На первом этапе синтеза осуществляется трансляция реализаций целевых алгоритмов в архитектурно-независимое представление (Meta-IR). Далее индуктивный синтезатор формирует единое для целевых алгоритмов архитектурно-зависимое представление (Target-IR) с учетом заданных архитектурных ограничений.

В результате из полученного Target-IR формируются:

1. Программная модель (симулятор) СВ.
2. Генератор кода компилятора СВ [3].
3. Спецификация архитектуры СВ для проектирования уровня RTL.

Meta-IR — ациклический, иерархический граф зависимостей по данным и состоянию, с явным параллелизмом на основе вычислительной модели потоков данных (dataflow model):

$$M = \langle V, L, E_{\text{data}}, E_{state}, C, I, O \rangle,$$

где:

- V — множество вершин;
- L — отображение вершин на метки операций-примитивов;
- $E_{data} \subseteq (V \times \mathbb{N}) \times (V \times \mathbb{N})$. Дуга $e = ((u, p_{out}), (v, p_{in})) \in E_{data}$ означает, что данные с выходного порта p_{out} узла u поступают на входной порт p_{in} узла v;
- $E_{state} \subseteq V \times V$. Дуга $s = (u, v) \in E_{state}$ устанавливает порядок исполнения операций, зависящий от состояния памяти;
- Множество кластеров $C = \{c_1, c_2, \dots, c_k\}$, где $c_i \subseteq V$;
- Входы и выходы кластеров $I(c) = \langle (v_1, p_{out_1}), (v_2, p_{out_2}), \dots, (v_n, p_{out_n}) \rangle, O(c) = \langle (u_1, q_{in_1}), (u_2, q_{in_2}), \dots, (u_m, q_{in_m}) \rangle$.

На рис. 2 показан пример графа зависимостей Meta-IR. Красными дугами отмечены зависимости по состоянию памяти. Операция synch принимает несколько независимых дуг по состоянию, синхронизируя действия с памятью.

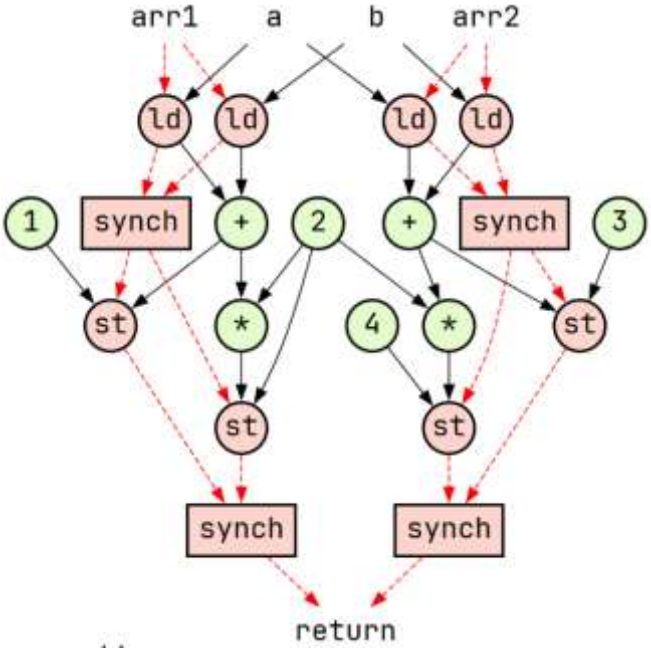

Рис. 2. Пример графа зависимостей Meta-IR

На рис. 3 показаны детали работы индуктивного синтезатора представления Target-IR. Целевые реализации алгоритмов анализируются на предмет структурной избыточности и уровня структурного подобия. Устранение структурной избыточности осуществляется на основе графового сжатия повторяющихся подграфов в кластеры. Синтез графа архитектуры реализуется путем слияния структурно подобных графов с обобщением ресурсов, а также с помощью вставки узлов-мультиплексоров и конвейеризующих регистров.

Синтез Target-IR для архитектуры непрограммируемого конвейера описан в [4].

III. Метод синтеза специализированных команд

На вход синтезатора специализированных команд

Рис. 3. Схема индуктивного синтеза представления Target-IR

поступает множество графов зависимостей целевых алгоритмов G в представлении Meta-IR. Результатом синтеза является множество специализированных команд, извлеченных из G.

Синтез специализированных команд состоит из следующих этапов:

1. Перечисление индуцированных подграфов из G.
2. Удаление структурных (изоморфных) дублей.
3. Синтез полей литералов.
4. Выбор команд.
5. Удаление семантических дублей.

Перечисление индуцированных подграфов $V(S_i) \subseteq V(G)$ при заданных ограничениях архитектурных ограничениях:

$$|S_i| \le N_{max}, |\mathrm{Ins}(S_i)| \le I_{max}, |\mathrm{Outs}(S_i)| \le O_{max}, \mathrm{depth}(S_i) \le D_{max},$$

и с учетом ограничения выпуклости:

$$\forall u, v \in S_i: (\exists \text{ путь } u \dashrightarrow w \dashrightarrow v \text{ в } G) \Longrightarrow w \in S_i.$$

Перечисление индуцированных подграфов может быть осуществлено одним из известных алгоритмов [5, 6], выбираемых в зависимости типа специализированных команд (число выходов, являются ли команды векторными и так далее), а также от допустимой вычислительной сложности выбранного алгоритма.

Удаление изоморфных дублей происходит с учетом свойства коммутативности команд. Используется канонический код $canon$ графа (например, с помощью известных алгоритмов BLISS или McKay) как ключа в хеш-таблице H для группировки эквивалентных кандидатов:

$$H: \mathrm{canon}(S_i) \to \{S_1, \dots, S_m\}.$$

В процессе синтеза полей литералов обобщаются подграфы команд, отличающиеся только значением узла-константы. Создаются обобщенные узлы-диапазоны с указанием разрядности для поддержки множества обобщенных констант.

Этап выбора команд осуществляется для удаления тех подграфов команд, которые в реальности не используются соответствующей фазой компилятора. Выбор команд осуществляется в два этапа: 1) минимизация числа вершин покрытого и свернутого графа, 2) минимизация числа выбранных шаблонов команд.

Семантическая фильтрация множества синтезированных команд представляет собой удаление команд, которые являются частными вычислительными случаями других команд (поглощение функций), как описано в [5]. Полученные соответствия между общими и частными случаями позволяет сформировать набор правил машинно-зависимой оптимизации.

Таким образом метод синтеза специализированных команд можно рассматривать как специальный случай словарного графового сжатия для заданного множества целевых алгоритмов и с учетом архитектурных ограничений.

IV. Метод синтеза графа архитектуры

Слияние структурно подобных графов алгоритмов происходит с обобщением ресурсов. Это слияние осуществляется на основе вставки в общий синтезированный граф узлов-мультиплексоров, что позволяет настроить этот граф на выполнение каждого из алгоритмов.

Метод синтеза графа архитектуры основан на решении задачи программирования в ограничениях. При этом, в отличие от известных подходов, гарантируется отсутствие комбинационных циклов в графе архитектуры.

Дано множество вычислительных графов, исполняемых взаимно исключительно:

$$G = \{G_1, \dots, G_N\}, G_g = (V_g, E_g), E_g \subseteq V_g \times V_g \times \mathbb{N}.$$

Ребро $(v, w, p) \in E_g$ означает, что вершина v использует результат вершины w по входному порту p.

Каждому графу соответствует функция разметки

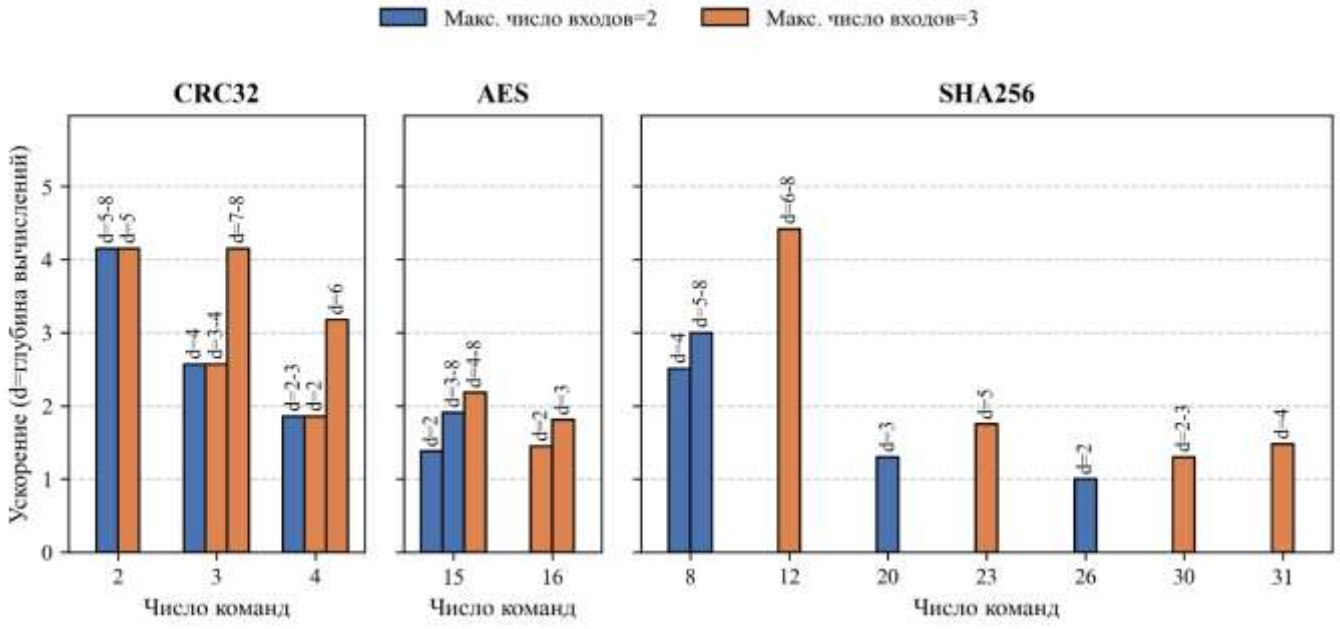

Рис. 4. Синтезированные наборы команд для заданного пространства конфигураций архитектурных параметров

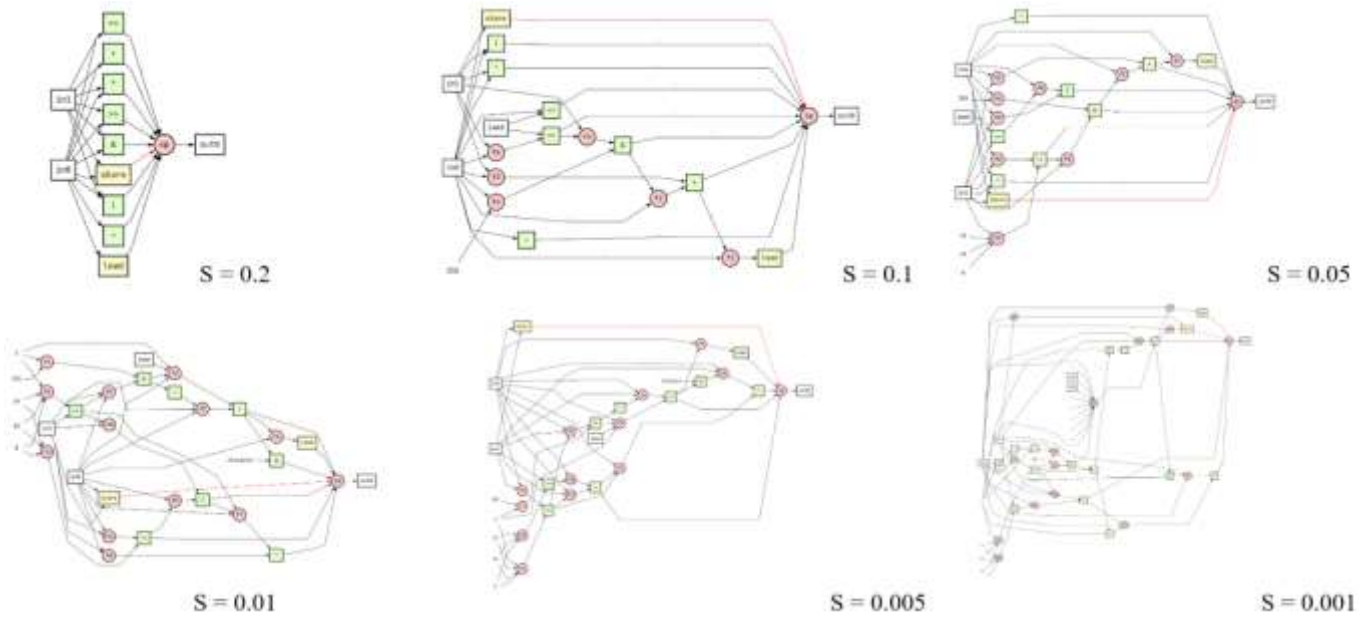

Рис. 5. Примеры синтезированных графов архитектуры (AES, CRC32, Magma) для различных значений поддержки S

$$\mathcal{L} = \{L_1, \dots, L_N\}, L_g: V_g \to O,$$

где $O = \{o_1, o_2, \dots, o_m\}$ — конечное множество типов операций.

Для каждого типа операции $o \in O$ выделяется минимально необходимое число аппаратных ресурсов:

$$\text{size}(o) = \max_{1 \le g \le N} |\{v \in V_g : L_g(v) = o\}|.$$

Формируется плоское адресное пространство ресурсов. Пусть

$$S_o = \sum_{o' < o} \text{size}(o'),$$

тогда множество идентификаторов ресурсов типа o определяется как

$$R_o = \{S_o, S_o + 1, \dots, S_o + \text{size}(o) - 1\}.$$

Задано глобальное множество ресурсов

$$R = \bigcup_{o \in O} R_o.$$

Допустимое множество ресурсов для вершины $v \in V_g$:

$$M_{g,v} = R_{L_g(v)}.$$

Заданы переменные оптимизации. Назначение вершины $v \in V_g$ на ресурс:

$$x_{g,v} \in M_{g,v}.$$

Наличие физической связи $w \xrightarrow{p} u$:

$$y_{u,w,p} \in \{0,1\}.$$

Топологический уровень ресурса u:

$$z_u \in \{0, \dots, |R| - 1\}.$$

Заданы ограничения. В пределах одного графа две различные вершины не могут быть отображены на один и тот же ресурс. Поэтому задано взаимоисключающее назначение ресурсов (свойство инъективности):

$$\forall g \in \{1, \dots, N\}: \text{AllDifferent}(\{x_{g,v}\}_{v \in V_g}).$$

Задано ограничение, сохраняющее порядок зависимостей и ацикличность графа архитектуры:

$$\forall g, \forall (v, w, p) \in E_g, \forall u \in M_{g,v}, \forall u' \in M_{g,w}, u \neq u':$$

$$(x_{g,v} = u \wedge x_{g,w} = u') \Rightarrow (y_{u,u',p} = 1 \wedge z_u > z_{u'}).$$

Сохранение ацикличности графа необходимо для гарантии отсутствия комбинационных циклов, то есть запрещенных настроек мультиплексоров, порождающих в графе цикл. Комбинационные циклы не поддерживаются многими САПР проектирования вычислительных систем.

Задана целевая функция для минимизации суммарного числа физических связей и, что то же самое, минимизации количества мультиплексоров во входах функциональных блоков:

$$min \sum_{u,w \in R, u \neq w, p \in \mathbf{N}} y_{u,w,p}.$$

Для формирования программной модели СВ необходимо топологически отсортировать узлы графа архитектуры с учетом зависимостей и транслировать получившийся список узлов в код программы-интерпретатора. При этом код команды определяется вектором значений мультиплексоров, настраивающих граф архитектуры на выполнение конкретной команды.

V. Апробация предложенных методов

На рис. 4 показаны результаты поиска в пространстве синтезированных наборов команд алгоритмов CRC32, AES и SHA256. Для каждого из этих алгоритмов команды синтезируются на основе подхода из [6], а пространство решений задано ограничениями по максимальному числу входов (числу аргументов) синтезированных команд $m \in \{2,3\}$ и глубине вычислений $d \in \{2, \dots, 8\}$.

Представленные результаты показывают, что выбор предпочтительного синтезированного набора команд зависит от допустимых архитектурных ограничений. В частности, для CRC32 максимальное ускорение по сравнению с набором команд, состоящим из примитивных узлов, не объединенных в кластеры, достигается для $d = 5$ и $m = 2$. Тем не менее, полученная глубина вычислений может оказаться неприемлемой с точки зрения достижения желаемого тактового периода в аппаратной реализации, что

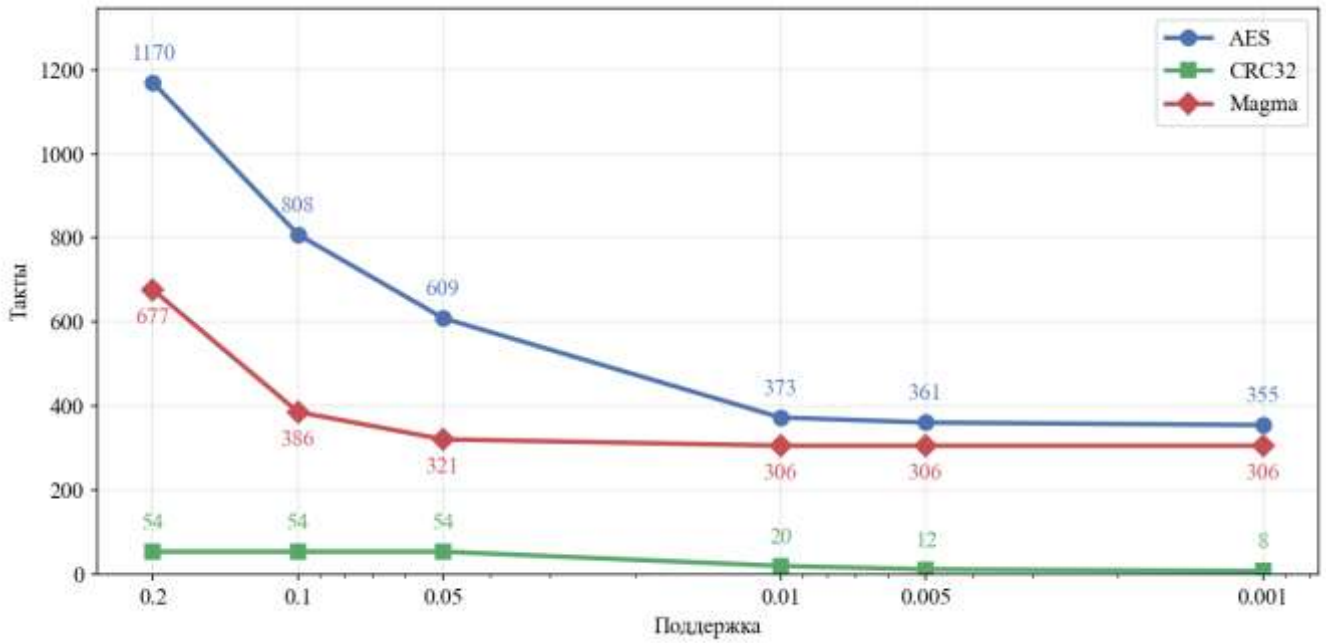

Рис. 6. Влияние поддержки на ускорение выполнения алгоритмов AES, CRC32, Magma с помощью синтезированных графов архитектуры

потребует перейти к рассмотрению иных конфигураций наборов команд.

При этом показатель числа синтезированных команд следует оценивать в контексте далее рассматриваемой процедуры слияния трактов данных, то есть большое число синтезированных команд не приводит однозначно к получению общего графа архитектуры, требующего большого числа аппаратных ресурсов.

На рис. 5 показаны синтезированные графы архитектуры, которая позволяет выполнить 3 алгоритма: AES, CRC32, Magma. Синтез показан для разных значений параметра поддержки (support) — нормализованной частоты вхождения подграфа команды во множество рассматриваемых графов алгоритмов. Поддержка используется в процедуре графового словарного сжатия и при синтезе набора команд, который далее объединяется в общий граф на основе структурного подобия. При этом можно заметить, что уменьшение поддержки приводит к графам более высокой сложности, что, с одной стороны, приводит и к использованию большего числа аппаратных ресурсов, а с другой — дает более специализированный тракт данных, позволяющий выполнить поддерживаемые алгоритмы за меньшее число тактов, чем в случае увеличенных значений поддержки.

На рис. 6 для тех же синтезированных графов архитектуры показано, как влияет значение поддержки на получаемое ускорение вычислений в тактах. Здесь компромиссным выглядит значение $S = 0.01$, ниже которого сокращение тактов является несущественным, но продолжает увеличиваться сложность аппаратной реализации (см. рис. 5).

Решение задач программирования в ограничениях в представленных примерах получено с помощью решателя CP-SAT из библиотеки OR-Tools компании Google.

VI. Сравнение с существующими решениями

Метод синтеза промежуточных представлений, описанный в этой статье, можно отнести к области индуктивного синтеза программ. Однако, в отличие от известных подходов [7], рассматриваемый в статье синтезатор принимает на вход не множество примеров (экземпляров входных данных и результатов), а множество программ, по которым необходимо построить поддерживающий их выполнение специализированный интерпретатор с учетом заданных архитектурных ограничений.

Известные работы из области синтеза специализированных процессорных команд [5,6,8] рассматривают, в основном, лишь частную задачу перечисления индуцированных графов. При этом, в отличие от представленного выше метода, игнорируются вопросы синтеза полей литералов, поддержки компилятором на этапе фазы выбора команд, а также вопросы семантической фильтрации полученного набора команд и синтеза правил локальной оптимизации.

Представленный в статье метод синтеза графа архитектуры имеет сходство с решением задачи слияния трактов данных [9], а также с объединением термов-деревьев [10]. В отличие от этих подходов изложенный выше метод впервые применен для синтеза интерпретатора, а также гарантирует отсутствие комбинационных циклов.

VII. Заключение

В статье рассмотрена задача автоматизации построения инструментального программного обеспечения на ранних этапах совместного проектирования программной и аппаратной части СВ. Представлен разработанный метод перехода от архитектурно-независимого промежуточного представления программ

к архитектурно-зависимому на основе анализа структурной избыточности и подобия графов целевых алгоритмов. Предложен метод синтеза специализированных команд, включающий синтез полей литералов, фазу выбора команд и семантическую фильтрацию множества генерируемых инструкций. Кроме того, разработан метод построения архитектур реконфигурируемых и программируемых СВ на основе слияния трактов данных, который за счет сведения к задаче программирования в ограничениях гарантирует отсутствие комбинационных циклов. Апробация на криптографических алгоритмах показала, что предложенные методы позволяют автоматически формировать обобщенные графы архитектуры с учетом ограничений на аппаратные ресурсы, а также позволяют автоматически получить генератор кода компилятора и программную модель СВ.

В качестве направлений дальнейших исследований планируется проведение оценки предложенных методов на более широком множестве целевых алгоритмов. Кроме того, предполагается развитие алгоритмов индуктивного синтеза для автоматической поддержки VLIW- и SIMD-архитектур, а также расширение графовых моделей промежуточных представлений для учета циклических конструкций.

Библиография

[1] H. Ye, H. Jun, H. Jeong, S. Neuendorffer, and D. Chen, “ScaleHLS,” *Proceedings of the 59th ACM/IEEE Design Automation Conference*, Jul. 2022, doi: https://doi.org/10.1145/3489517.3530631.

[2] F. Freitag *et al.*, “OpenVADL: An Open Source Implementation of the Vienna Architecture Description Language,” *Lecture Notes in Computer Science*, pp. 156–171, Oct. 2025, doi: https://doi.org/10.1007/978-3-032-03281-2_11.

[3] P. N. Sovetov, “Development of DSL Compilers for Specialized Processors,” *Programming and Computer Software*, vol. 47, no. 7, pp. 541–554, Dec. 2021, doi: https://doi.org/10.1134/s0361768821070082.

[4] P. N. Sovietov, “Trubol: Synthesis of Pipelined Circuits from Python-based DSL Specifications,” *2023 5th International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA)*, pp. 490–494, Nov. 2023, doi: https://doi.org/10.1109/summa60232.2023.10349644.

[5] C. Xiao, S. Wang, W. Liu, X. Wang, and E. Casseau, “An Optimal Algorithm for Enumerating Connected Convex Subgraphs in Acyclic Digraphs,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 1, pp. 261–265, Jan. 2021, doi: https://doi.org/10.1109/tcsii.2020.3000297.

[6] P. N. Sovietov, “Algorithms for Improving the Automatically Synthesized Instruction Set of an Extensible Processor,” *Programmnaya Ingeneria*, vol. 14, no. 5, pp. 225–231, May 2023, doi: https://doi.org/10.17587/prin.14.225-231.

[7] S. Jacindha, G. Abishek, and P. Vasuki, “Program Synthesis—A Survey,” *Lecture Notes in Electrical Engineering*, pp. 409–421, 2022, doi: https://doi.org/10.1007/978-981-16-8484-5_39.

[8] E. Rezunov, N. Zurstraßen, L. M. Reimann, and R. Leupers, “Automatic Microarchitecture-Aware Custom Instruction Design for RISC-V Processors,” *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, Oct. 2025, doi: https://doi.org/10.1109/iccad66269.2025.11240781.

[9] N. Moreano, E. Borin, Cid de Souza, and G. Araujo, “Efficient datapath merging for partially reconfigurable architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 7, pp. 969–980, Jul. 2005, doi: https://doi.org/10.1109/tcad.2005.850844.

[10] Y. Xiao, C. Yin, Y. Sun, Y. Zou, and Y. Liang, “Finding Reusable Instructions via E-Graph Anti-Unification,” *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 749–763, Mar. 2026, doi: https://doi.org/10.1145/3779212.3790162.

Methods for Synthesizing Program Intermediate Representations for the High-Level Design of Specialized Processors

P.N. Sovietov

***Abstract*— This paper discusses methods for automating the construction of software toolchains without explicit architecture descriptions at the early stages of designing specialized hardware accelerators. A formal model of an architecture-independent program intermediate representation is presented, along with a method for synthesizing an architecture-dependent representation based on the analysis of structural redundancy and similarity of target algorithm graphs. The resulting architecture-dependent representation, in the form of an architecture graph, can be used to automatically generate a software simulator and a compiler. A custom instruction synthesis method is developed, which includes the generation of literal fields and the semantic filtering of the set of synthesized instructions. Furthermore, the paper describes a method for constructing architectures of reconfigurable and programmable processors based on the problem of merging structurally similar algorithm graphs with resource generalization. This problem is reduced to a constraint programming model, which guarantees the absence of unwanted combinational loops in the hardware implementation. The evaluation of the proposed methods on cryptographic algorithms (AES, CRC32, Magma, SHA256) confirmed the feasibility of automatic architectural design space exploration to achieve the required trade-off between hardware resource utilization and processor performance.**

***Keywords*— specialized processors, program intermediate representation, instruction set synthesis, datapath merging.**

REFERENCES

[1] H. Ye, H. Jun, H. Jeong, S. Neuendorffer, and D. Chen, “ScaleHLS,” *Proceedings of the 59th ACM/IEEE Design Automation Conference*, Jul. 2022, doi: https://doi.org/10.1145/3489517.3530631.

[2] F. Freitag *et al.*, “OpenVADL: An Open Source Implementation of the Vienna Architecture Description Language,” *Lecture Notes in Computer Science*, pp. 156–171, Oct. 2025, doi: https://doi.org/10.1007/978-3-032-03281-2_11.

[3] P. N. Sovetov, “Development of DSL Compilers for Specialized Processors,” *Programming and Computer Software*, vol. 47, no. 7, pp. 541–554, Dec. 2021, doi: https://doi.org/10.1134/s0361768821070082.

[4] P. N. Sovietov, “Trubol: Synthesis of Pipelined Circuits from Python-based DSL Specifications,” *2023 5th International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA)*, pp. 490–494, Nov. 2023, doi: https://doi.org/10.1109/summa60232.2023.10349644.

[5] C. Xiao, S. Wang, W. Liu, X. Wang, and E. Casseau, “An Optimal Algorithm for Enumerating Connected Convex Subgraphs in Acyclic Digraphs,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 1, pp. 261–265, Jan. 2021, doi: https://doi.org/10.1109/tcsii.2020.3000297.

[6] P. N. Sovietov, “Algorithms for Improving the Automatically Synthesized Instruction Set of an Extensible Processor,” *Programmnaya Ingeneria*, vol. 14, no. 5, pp. 225–231, May 2023, doi: https://doi.org/10.17587/prin.14.225-231.

[7] S. Jacindha, G. Abishek, and P. Vasuki, “Program Synthesis—A Survey,” *Lecture Notes in Electrical Engineering*, pp. 409–421, 2022, doi: https://doi.org/10.1007/978-981-16-8484-5_39.

[8] E. Rezunov, N. Zurstraßen, L. M. Reimann, and R. Leupers, “Automatic Microarchitecture-Aware Custom Instruction Design for RISC-V Processors,” *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, Oct. 2025, doi: https://doi.org/10.1109/iccad66269.2025.11240781.

[9] N. Moreano, E. Borin, Cid de Souza, and G. Araujo, “Efficient datapath merging for partially reconfigurable architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 7, pp. 969–980, Jul. 2005, doi: https://doi.org/10.1109/tcad.2005.850844.

[10] Y. Xiao, C. Yin, Y. Sun, Y. Zou, and Y. Liang, “Finding Reusable Instructions via E-Graph Anti-Unification,” *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 749–763, Mar. 2026, doi: https://doi.org/10.1145/3779212.3790162.

P. N. Sovietov, MIREA – Russian Technological University, Moscow, Russia (e-mail: sovetov@mirea.ru).