ИПМ им.М.В.Келдыша РАН

Абрау-2017 • Труды конференции

А.П. Баглий, Д.В. Дубров,
Б.Я. Штейнберг, Р.Б. Штейнберг

**Повторное использование ресурсов
при конвейерных вычислениях**

Рекомендуемая форма библиографической ссылки

Баглий А.П., Дубров Д.В., Штейнберг Б.Я., Штейнберг Р.Б. Повторное использование ресурсов при конвейерных вычислениях // Научный сервис в сети Интернет: труды XIX Всероссийской научной конференции (18-23 сентября 2017 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2017. — С. 43-47. — URL: <http://keldysh.ru/abrau/2017/53.pdf> doi:[10.20948/abrau-2017-53](https://doi.org/10.20948/abrau-2017-53)

Повторное использование ресурсов при конвейерных вычислениях

А.П. Баглий¹, Д.В. Дубров¹, Б.Я. Штейнберг¹, Р.Б. Штейнберг¹

1 Южный федеральный университет

Аннотация. Данная работа посвящена автоматической генерации электронных схем вычислительного конвейера по программному циклу языка высокого уровня. При таком высокоуровневом синтезе не всегда можно заранее предсказать, достаточно ли будет ресурсов ПЛИС для реализации сгенерированного HDL описания электронной схемы. В данной работе рассматриваются высокоуровневые преобразования входного текста, которые могут привести к экономии ресурсов при генерации схемы. Реализация таких преобразований предполагается на основе ОРС – оптимизирующей распараллеливающей системы, в разработке которой участвуют авторы.

Ключевые слова: оптимизирующий компилятор, ПЛИС, преобразования программ, межпроцедурный анализ.

1. Введение

Данная работа относится к высокоуровневому синтезу и может использоваться в конвертере с высокоуровневого языка в язык описания электронных схем (см., например, [4], [5]). Разработка такого конвертора связана с проблемой прогноза достаточности ресурсов ПЛИС для размещения схемы.

Можно ли схему разместить на ПЛИС становится ясно после компиляции HDL-описания. Если размещение схемы невозможно, то ее следует перепроектировать, что требует больших затрат времени специалистов высокой квалификации. Если HDL-описание схемы получается в результате работы конвертора с языка высокого уровня, некоторое перепроектирование можно выполнить в исходном коде высокоуровневого языка программирования.

В представленной работе предполагается генерация HDL-описания конвейерного вычислителя. Предлагается преобразование исходной высокоуровневой программы, которое позволяет минимизировать ресурсы ПЛИС после конвертирования в HDL-описание электронной схемы. Суть преобразования в поиске подобных выражений, которые можно вычислять одной функцией. Предлагаемое преобразование является обобщением известного поиска общих подвыражений, которое используется в оптимизирующих компиляторах и рассматривалось применительно к оптимизации генерации HDL-описаний [7].

В некоторых случаях повторное использование конвейера может привести к увеличению интервала инициализации итераций (скважности). Но это лучше, чем совсем не создать схему из-за нехватки ресурсов.

2. О высокой производительности вычислительных систем на ПЛИС и о конверторах типа C2HDL

Традиционные вычислительные архитектуры, такие как процессоры и графические ускорители, имеют в своем составе фиксированное количество вычислительных модулей (порты микроопераций, ядра GPU и т. д.), каждый из которых может выполнять фиксированный набор операций с данными фиксированного набора разрядностей. Во время работы таких процессоров и ускорителей многие входящие в их состав устройства много времени простаивают. Особенно это относится к большей части логики устройств управления универсальных процессоров. В отличие от традиционных архитектур, логические элементы ПЛИС можно настроить на выполнение конкретной задачи наиболее эффективным образом: сгенерировать схемы вычислителей для выполнения только необходимых операций с данными требуемой разрядности, использовать свободные ресурсы для параллельного выполнения разных итераций цикла, выделить часть ресурсов под память нужного объема и т. д. На практике применение ПЛИС может ускорить решение задачи в десятки раз по сравнению с использованием других архитектур [1].

Одной из основных проблем применения ПЛИС для ускорения вычислений является сложность их программирования, которое требует использование низкоуровневых средств проектирования электронных схем. В последнее время для решения данной проблемы разрабатываются разнообразные инструменты конвертирования программ на традиционном языке программирования в язык описания оборудования. К ним относятся конвертеры с традиционных языков: C (Xilinx Vivado Design Suite, Altera C2H, Bambu), Haskell (Clash [3]), а также со специально разработанных для этой цели (Handel C, Mitrion-C, HaSCoL).

3. Поиск одинаковых подвыражений

В данной работе представлен алгоритм поиска эквивалентных фрагментов кода, которые могут быть вычислены одним и тем же конвейерным вычислителем. От алгоритма требуется:

1. достаточно малое время работы на большом объеме кода.
2. возможность находить полностью эквивалентные фрагменты максимального размера.
3. нахождение полностью эквивалентных фрагментов.

Для достижения этих целей за основу взят алгоритм поиска полностью или частично повторяющихся фрагментов из [2].

Алгоритм из [2] использует хеширование и сравнение поддеревьев AST программы для быстрого поиска похожих фрагментов. На его выходе список фрагментов, метрика схожести которых выше заданного порога. Для поиска эквивалентных фрагментов этот список следует уточнить, проверив каждую пару похожих фрагментов на эквивалентность.

Использованный для первоначального поиска фрагментов алгоритм позволяет распознать некоторые простые выражения как совпадающие, например выражения $a \cdot \cos(x[i])$ и $\cos(y[i]) \cdot b$ отличаются только именами идентификаторов и порядком операндов в коммутативных операциях. Но в более сложных случаях, например, при замене операций эквивалентными, исходного алгоритма недостаточно, поэтому требуется дополнительный алгоритм проверки на эквивалентность, который основан на преобразовании «Линеаризация выражений», имеющееся в ОРС [6].

4. Линеаризация выражений

Предположим, дано выражение, состоящее из арифметических операций, переменных и констант. Линеаризацией выражения относительно набора переменных a_i называется преобразование произвольного выражения к виду: $e_1 \cdot a_1 + e_2 \cdot a_2 + \dots + e_n \cdot a_n + e_0$, где e_i – выражения, a_i – переменные, входящие в исходное выражение, и при этом ни одна из них не входит ни в одно из выражений e_i . Анализом линейности выражения называется анализ возможности осуществления линеаризации. Будем называть полученное представление линейной формой выражения (линейным разложением). Будем называть базой линеаризации набор переменных a_i , e_i – коэффициентами, и при этом e_0 – свободным членом линейного разложения.

В ОРС реализована функция, выполняющая линеаризацию выражений, располагая базовые переменные a_i в алфавитном порядке. Базовые переменные могут быть индексными. Линеаризация выражений, используя свойства коммутативности, приводит выражения к виду, в котором слагаемые или множители расположены в алфавитном порядке. Линеаризация выражений проводит преобразования, которые иногда называют «приведением подобных». Например,

$$\begin{aligned}x+x &\rightarrow 2 \cdot x \\ b+a &\rightarrow a+b\end{aligned}$$

Если два выражения эквивалентны как функции, т.е. на одних и тех же входных данных оба выражения вычисляют одинаковые результаты, то из этого не вытекает, что преобразование «линеаризация выражений» приведет их к одному виду.

Определение 1. Два фрагмента кода А и В будем называть подобными, если существует (существует теоретически, не обязательно в данной

программе) такой третий фрагмент кода D, который может быть получен как из A, так и из B с помощью преобразования «Линеаризация выражений».

Пример 1. Поиск одинаковых поддеревьев AST-дерева не обнаружит, что выражения $a^2 + \cos(x[i])$ и $a + a + \cos(y[i]) * b$ подобны и можно выполнять одним и тем же конвейерным вычислителем. Использование преобразования «Линеаризация выражений» позволяет заметить, что оба эти выражения подобны, т.е. могут выполняться на одном конвейерном вычислителе.

Преобразование «Линеаризация выражений» применяется к индексным выражениям массивов для определения информационных зависимостей и используется для приведения циклов к виду, допускающему реализацию конвейерным или многоконвейерным вычислителем [8].

Определение 2. Два выражения будем называть родственными, если существует третье выражение, из которого исходные выражения получаются фиксацией значений некоторых переменных.

Пример 2. Следующие два выражения

$$(x*x+a*x)/(x+c)$$

$$(x*x+b)/(x+c)$$

являются частными случаями выражения (могут быть получены фиксацией переменных b и a соответственно)

$$(x*x+a*x+b)/(x+c)$$

и могут вычисляться одним конвейерным вычислителем.

Определение 3. Два выражения A и B будем называть родственно подобными, если существуют такие родственные выражения A1 и B1, что A1 подобно A и B1 подобно B.

Таким образом, родственно подобные выражения могут вычисляться одной и той же функцией. В поиске родственно подобных выражений может быть полезна линеаризация выражений.

5. Заключение

В представленной работе рассмотрена задача поиска нескольких фрагментов кода, которые можно выполнять одним конвейерным вычислителем. Эта задача является более общей по сравнению с известным оптимизирующим преобразованием программ «Поиск одинаковых

подвыражений». Решение такой задачи может привести к уменьшению вычислительных ресурсов, необходимых для реализации конвейерного вычислителя за счет повторного использования этих ресурсов. Это может быть ценно при создании конвейерных вычислителей и при создании компилятора на процессор с ПЛИС-ускорителем.

6. Литература

1. Румянцев Ю. Применение FPGA для расчета деполимеризации микротрубочки методом броуновской динамики. URL: <https://habrahabr.ru/post/314296/> (дата обр.: 18.07.2017).
2. Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, Lorraine Bier. Clone detection using abstract syntax trees // ICSM '98, march 16-19, 1998. Proceedings of the international conference on software maintenance, p. 368.
3. Boris Ya. Steinberg, Denis V. Dubrov, Yury V. Mikhailuts, Alexander S. Roshal, Roman B. Steinberg. Automatic High-Level Programs Mapping onto Programmable Architectures // Proceedings of the 13th International Conference on Parallel Computing Technologies, August 31 – September 4, 2015, Petrozavodsk, Russia. V 9251. p. 474-485.
4. Баглий А.П., Дубров Д.В., Михайлуц Ю.В., Штейнберг Б.Я., Штейнберг Р.Б. Developing a High-Level Language Compiler for a Computer with Programmable Architecture // Труды Международной конференции по программной инженерии CEE SECR-2016. CEE-SECR '16, October 28-29, 2016, Moscow, Russian Federation, 2016 ACM. ISBN 978-1-4503-4884-3/16/10. DOI: <http://dx.doi.org/10.1145/3022211.3022226>
5. Оптимизирующая распараллеливающая система www.ops.rsu.ru (дата обращения 08.02.2017).
6. J.J. Hammes, W. Bohm, C. Ross, M. Chawathe, B. Draper, R. Rinker, and W. Najjar. Loop Fusion and Temporal Common Subexpression Elimination in Window-based Loops // IPDPS 8th Reconfigurable Architectures Workshop, San Francisco, CA, April 27, 2001.
7. Wester R., Kuper J. Design space exploration of a particle filter using higher-order functions // Reconfigurable Computing: Architectures, Tools, and Applications. London: Springer. 2014. p. 219-226. (Lecture Notes in Computer Science; 8405). DOI: 10.1007/978-3-319-05960-0_21.
8. Штейнберг Р.Б. Отображение гнезд циклов на многоконвейерную архитектуру // Программирование. — 2010. — № 3. — С. 69–80.