Комплекс средств высокоуровневого синтеза «Тесей» для многокристальных реконфигурируемых вычислительных систем

А.И. Дордопуло[1], И.И. Левин[1,2], В.А. Гудков[1, 2], А.А. Гуленок[1]

[1]Общество с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров», г. Таганрог, Россия
[2]Федеральное государственное автономное образовательное учреждение высшего образования Южный федеральный университет, г. Таганрог, Россия

В статье рассматривается программный комплекс «Тесей», преобразующий последовательную программу на языке высокого уровня С в конфигурационные файлы программируемых логических интегральных схем (ПЛИС). В отличие от известных средств высокоуровневого синтеза, создающих из последовательной программы автономное IP-ядро, «Тесей» создает параллельно-конвейерное решение задачи для заданного пользователем ресурса реконфигурируемой вычислительной системы (РВС), содержащей множество ПЛИС, соединенных пространственной коммутационной системой. Для этого компоненты комплекса строят информационный граф из последовательной программы, преобразуют его в кадровую структуру, которая формальными методами в автоматическом режиме без участия пользователя адаптируется к заданной конфигурации РВС. Отличительной особенностью подхода является существенно меньшее по сравнению с распараллеливающими компиляторами число анализируемых вариантов, что позволяет сократить время портации последовательных программ. Описана новая программа «Ариадна» для синтеза стратегий преобразования программ разных предметных областей и разработанные решения для распараллеливание косвенной адресации, часто встречающейся в последовательных программах.

Ключевые слова: высокоуровневый синтез, HLS, трансляция программ, язык С, редукция производительности, реконфигурируемая вычислительная система, программирование многопроцессорных вычислительных систем.

1. Введение

Средства высокоуровневого синтеза (High Level Synthesis) или HLS-компиляторы [1, 2] предназначены для преобразования программ на одном из языков высокого уровня в конфигурационные файлы специализированных аппаратных средств на языках описания аппаратуры HDL (Hardware description language). Использование HLS-компиляторов позволяет снизить требования к квалификации разработчиков для вычислительных систем с реконфигурируемой элементной базой, например, построенных на основе программируемых логических интегральных схем (ПЛИС). HLS-компиляторы предназначены для быстрой разработки проектов, где важна скорость создания конфигурационных файлов, а не эффективность кода и/или сокращение аппаратных затрат. Большинство средств высокоуровневого синтеза преобразуют вычислительно трудоемкий фрагмент программы на языке С в сложный функциональный блок (IP-ядро), представляющий собой цифровой автомат или специализированный процессор, выполняющий вычисления быстрее за счет прямой реализации информационных зависимостей и свойств архитектуры ПЛИС даже при существенной (по сравнению с микропроцессорами) разнице в тактовой частоте работы.

Автоматизация создания IP-ядра HLS-компилятором облегчает портацию вычислений в архитектуру ПЛИС, но не гарантирует их эффективную реализацию, т.к. организация потоков данных возлагается на программиста, а средства масштабирования и синхронизации решения (хотя бы в пределах одного кристалла) отсутствуют. Для реконфигурируемых вычислительных систем

(РВС) [3], содержащих множество ПЛИС, связанных пространственной коммутационной системой, настройка синхронизации и согласования одновременной работы множества IP-ядер является сложным и трудоемким процессом, сравнимым по сложности с разработкой нового схемотехнического проекта.

Для преобразования последовательных программ на языке С в стандарте ISO/IEC 9899:1999 в решения многокристальных РВС с автоматической синхронизацией информационных и управляющих сигналов для всех используемых ПЛИС предназначен описываемый в настоящей статье комплекс средств высокоуровневого синтеза «Тесей» [4, 5].

В статье представлены результаты работ по созданию программного комплекса высокоуровневого синтеза [4, 5] с автоматическим преобразованием последовательных программ к доступному аппаратному ресурсу РВС: описана новая программа «Ариадна» для синтеза стратегий преобразования программ разных предметных областей и разработанные решения для распараллеливания косвенной адресации, часто встречающейся в последовательных программах. Статья организована следующим образом. Во втором разделе представлен краткий обзор известных HLS-компиляторов и их особенностей. Описание новой структуры комплекса и взаимосвязей между его компонентами, возможностей и ограничений представлено в третьем разделе. В четвертом разделе описана новая программа синтеза стратегий «Ариадна». В пятом разделе представлено описание разработанных преобразований для распараллеливания косвенной адресации. В заключении обобщаются полученные результаты и формулируются направления дальнейших исследований.

2. Краткий обзор проблемной области

Среди наиболее распространенных средств высокоуровневого синтеза можно выделить академические (DWARV [6], BAMBU [7], LEGUP [8]) и коммерческие (CatapultC, Vivado HLS [9], Vivado Vitis [10]) HLS-компиляторы, подробные обзоры возможностей, ограничений и отличительных характеристик которых представлены в [1, 2, 11–13]. Несмотря на различия в методах преобразования и форматах входных и выходных данных, большинство перечисленных компиляторов обладает следующими общими чертами:

– аппаратный ресурс для синтезируемого проекта ограничен одним кристаллом ПЛИС;

– для получения эффективного решения необходима выполняемая разработчиком предварительная разметка кода (например, директивами #pragma в Vivado HLS и Vitis), что требует детальных знаний о структуре и семантике транслируемой программы и синтезируемой реализации;

– использование и управление синтезированным IP-ядром осуществляется программистом самостоятельно, без поддержки со стороны среды разработки.

Синтез HLS-компиляторами однокристальных решений связан с широким распространением специализированных или отладочных плат, содержащих одну микросхему ПЛИС. В отличие от них комплекс «Тесей» разрабатывался для многокристальных РВС, что изначально предполагало создание не единичного IP-ядра, а поддержки различных форм организации вычислений для большого количества вычислительных устройств, требующих синхронизации для совместной согласованной работы. Синтез таких решений существенно сложнее, поскольку требует анализа и выбора рациональных вариантов из множества возможных, поэтому по сложности сравним с автоматическим распараллеливанием.

Комплекс средств высокоуровневого синтеза «Тесей», при схожей с Xilinx Vivado HLS и Vitis функциональности, обладает следующими основными отличиями:

– преобразование осуществляется на основе анализа информационных зависимостей входной программы, а не директив ручной разметки кода, аналогичных *#pragma* в Vivado HLS;

– для высокоуровневого синтеза и масштабирования его результатов параметры параллелизма кадровой структуры сокращаются, а не распараллеливаются, как для последовательной программы;

– произвольный доступ к памяти в последовательной программе преобразуется к регулярным операциям чтения-записи для потоков данных;

– эффективность синтезируемого решения может быть повышена различными способами организации вычислений (макроконвейер, конвейер в конвейере);

– автоматическая адаптация прикладной задачи к архитектуре и конфигурации заданной пользователем РВС;

– автоматическая синхронизация информационных и управляющих сигналов для всех используемых кристаллов ПЛИС;

– поддержка задач разных прикладных областей.

Входная программа на языке С для текущей версии комплекса «Тесей» должна соответствовать определенным требованиям:

– программа должна быть написана без использования операторов безусловного перехода goto, конструкций union и struct, функций strlen и функций работы с памятью (malloc/calloc, memcpy и др), а также некоторых директив условной компиляции для кроссплатформенных программ вида #if (Def_WinOrLin == 1);

– для определения входных и выходных переменных в тексте программы должны **обязательно** присутствовать процедуры загрузки и сохранения данных с фиксированными именами (load_data/save_data, load_data32/save_data32, load_data64/save_data64), соответствующие заданному шаблону;

– желательно избегать использования операторов break и return для немедленного выхода из циклов, а также использования переменных с косвенной адресацией, которую нельзя рассчитать на этапе трансляции, для счетчиков циклов и индексации массивов. Информационная эквивалентность для входных программ с такими операторами не гарантируется;

– желательно, чтобы текст входной программы был написан в процедурном стиле с наибольшей наглядностью, без применения возможных в языке С оптимизаций для реализуемого алгоритма.

3. Структура программного комплекса «Тесей»

Для эффективной реализации задач различных проблемных областей комплекс средств высокоуровневого синтеза «Тесей», в отличие от своих предыдущих версий [4, 5], дополнен компонентом синтеза стратегий преобразования параллельно-конвейерной программы «Ариадна». Структура новой версии комплекса приведена на рис. 1.

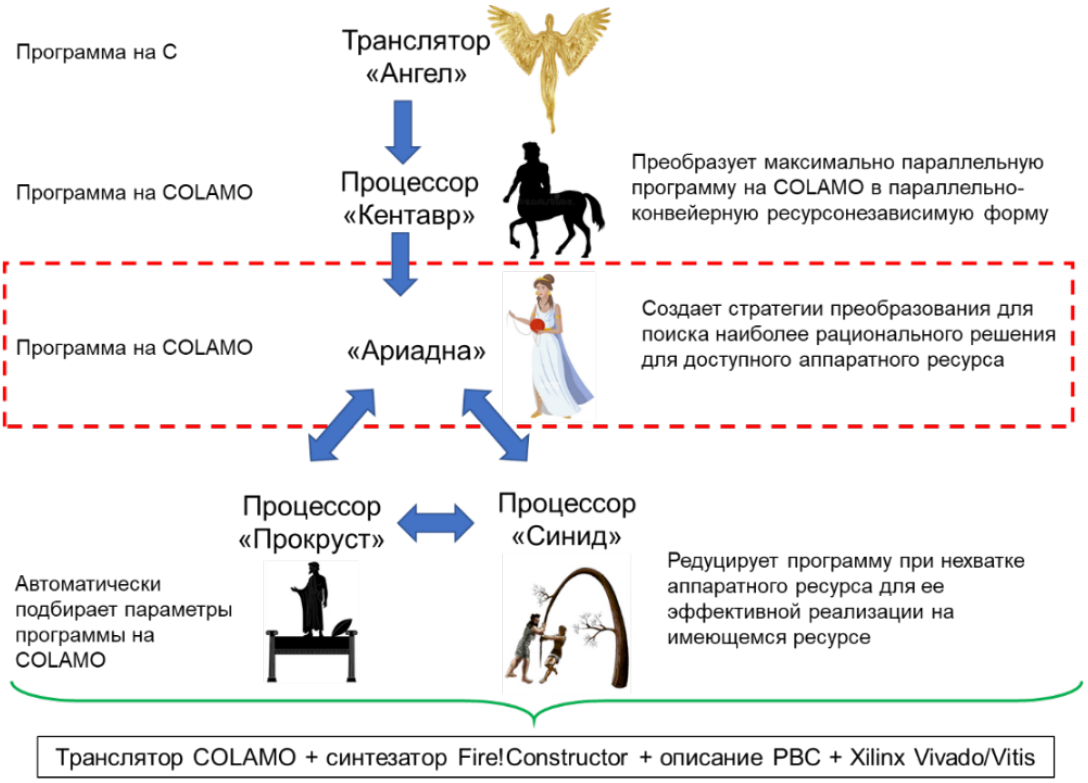

Рис. 1. Структура взаимодействия компонентов программного комплекса

«Ариадна» выполняет часть функций, ранее реализованных в «Прокрусте», и ряд новых преобразований для синтеза рациональных способов организации вычислений (макроконвейер, конвейер в конвейере). Первые этапы преобразования последовательной программы не претерпели изменений: как и в предыдущих версиях, «Ангел» преобразует входную программу на языке С в максимально параллельную кадровую структуру, описанную конструкциями языка программирования с неявным описанием параллелизма COLAMO, а «Кентавр» преобразует максимально параллельную кадровую структуру в ресурсонезависимую параллельно-конвейерную программу на языке COLAMO. Далее «Ариадна» определяет возможные стратегии преобразования ресурсонезависимой программы для поиска наиболее рационального решения на доступном аппаратном ресурсе РВС. Согласно стратегиям, синтезированным «Ариадной», «Прокруст» преобразует ресурсонезависимую программу и подбирает параметры каждой кадровой структуры для ее рациональной реализации в архитектуре заданной РВС. Далее, из всех синтезированных вариантов параллельно-конвейерной программы на COLAMO «Ариадной» выбирается наиболее рациональный по времени решения и занимаемому аппаратному ресурсу. «Синид», редуцирующий производительность кадровой структуры, выполняет вспомогательную роль и вызывается при необходимости либо «Прокрустом», либо «Ариадной», если его участие предусмотрено синтезированной стратегией.

Результатом работы «Ариадны» и «Прокруста» является программа на языке высокого уровня COLAMO с заданным (или наиболее близким к нему) уровнем реальной производительности на доступном вычислительном ресурсе. Транслятор языка программирования COLAMO транслирует программу в конфигурационные файлы ПЛИС многокристальных РВС, а синтезатор многокристальных решений Fire!Constructor выполняет автоматическую синхронизацию распределенных по разным кристаллам фрагментов решения. После этого синтезатор Xilinx Vivado создает загрузочные конфигурационные файлы (*.bit) для каждого кристалла.

4. Стратегии преобразования кадровых структур: программа «Ариадна»

Входными данными для «Ариадны» является полученная от «Кентавра» параллельная программа на языке COLAMO со структурной реализацией базового подграфа, у которой все константы масштабирования кадровой структуры равны 1. «Ариадна» оценивает аппаратные затраты на реализацию этой программы в РВС по всем компонентам ПЛИС (LUT, FlipFlop, DSP, BRAM и др.) и определяет критический аппаратный ресурс для решаемой задачи. Кроме этого, выполняется анализ интервала обработки данных, количества каналов распределенной памяти и латентности. Далее, в соответствии с методикой преобразования кадровых структур [4], после выбора критического ресурса «Ариадна» определяет последовательность (или стратегию) выполнения преобразований входной кадровой структуры с учетом особенностей ее информационных зависимостей и базового подграфа.

Для выбора преобразований «Ариадна» анализирует динамику нелинейного изменения аппаратных затрат $A_{CS} = \left(a_1^{CS}, a_2^{CS}, \dots, a_n^{CS}\right)$ кадровой структуры при изменении основных параметров быстродействия кадровой структуры $P_{CS} = (p_1, p_2, \dots, p_n)$ с учетом информационных зависимостей внутри и между подграфами (фрагментами) задачи. В соответствии с методикой [4], «Ариадна» анализирует параметры быстродействия кадровой структуры P_{CS}: L_i^{it} – число информационно-независимых подграфов в слое F_{ij}, It_i^{it} – число итераций в функции Φ_i, Ch_i^{it} – число внешних каналов кадровой структуры, Op_i^{it} – число устройств кадровой структуры FG^{it}, ρ_i^{it} – разрядность обрабатываемых данных, способ аппаратной реализации базового подграфа g_i и интервал обработки данных I. Каждый из этих параметров, в зависимости от предметной области, структуры и размерности решаемой задачи, может являться определяющим для критического аппаратного ресурса преобразуемой программы. Выбор первого изменяемого параметра быстродействия, как правило, определяет остальные преобразования стратегии.

Для выбранного критического аппаратного ресурса «Ариадна» определяет границы и рациональный шаг изменения параметров быстродействия кадровой структуры

$P_{CS} = \{L_i^{it}, It_i^{it}, Ch_i^{it}, Op_i^{it}, \rho_i^{it}, g_i\}$, а также список возможных и/или необходимых оптимизационных преобразований (конвейер в конвейере, макроконвейер, автоподстановка) для синтеза потенциально рациональной кадровой структуры. Таким образом, «Ариадна» определяет последовательность выполнения и основные параметры преобразований. «Прокруст» выполняет все преобразования кадровой структуры в соответствии с переданной ему стратегией как единую транзакцию, чтобы сократить накладные временные расходы на многократный вызов программы. Результатом реализации стратегии является создаваемая «Прокрустом» кадровая структура.

При наличии нескольких созданных «Ариадной» стратегий результирующие кадровые структуры помещаются в список, для всех элементов которого анализируются эффективность и время решения задачи, после чего выбирается наиболее рациональная кадровая структура, которая и является решением.

Одной из основных задач при формировании стратегии являются исключение заведомо неэффективных направлений движения в пространстве кадровых структур [5] и сокращение количества синтезируемых вариантов, каждый из которых должен удовлетворять заданным критериям эффективности – производительность не должна быть ниже 60 % от пиковой.

Основная стратегия преобразования состоит в структурной реализации базового подграфа и последующем изменении влияющих на параллелизм и занимаемый ресурс параметров кадровой структуры. При масштабировании сначала изменяется количество слоев – до ограничений максимального числа каналов в системе, затем – итерации – до ограничений доступного аппаратного ресурса, потом выполняется сокращение интервала обработки данных с помощью автоподстановки и/или других оптимизационных преобразований. Эта стратегия не является непременно обязательной к выполнению, она может быть изменена в зависимости от типа и предметной области преобразуемой задачи. Выбор такого порядка выполнения преобразований обусловлен следующими причинами:

– масштабирование (распараллеливание) по слоям линейно повышает производительность;

– распараллеливание по итерациям повышает производительность близко к линейной, но зачастую требует буферизации, которая в ряде случаев может достигать половины длины обрабатываемого потока данных;

– типовые методы сокращения интервала обработки данных – автоподстановка и замена переменных – существенно, в три–четыре раза увеличивают аппаратные затраты, а производительность при этом растет нелинейно.

Если в результате выполнения всех преобразований стратегии полученное решение удовлетворяет требованиям занимаемого ресурса и получен заданный уровень производительности, то цель считается достигнутой, и дальнейшие преобразования прекращаются. В противном случае, если заданный уровень производительности не достигнут, происходит изменение приоритетов выполнения преобразований и переход к следующей стратегии.

Если базовый подграф кадровой структуры входной параллельной программы поместился в доступный аппаратный ресурс, но не достигнут заданный уровень производительности (обычно 60 %), необходимо снова определять стратегии. Порядок выполнения преобразований значительно зависит от типа и предметной области решаемой задачи.

Например, при реализации вложенных конвейеров сокращение интервала обработки данных становится даже более эффективным, чем распараллеливание по слоям. Причиной этого является линейный рост производительности решения практически без дополнительных аппаратных затрат (добавляется лишь цепочка регистров в цепи обратной связи). Поэтому для задач, содержащих такие фрагменты, первым преобразованием стратегии целесообразно выполнять оптимизацию интервала обработки данных.

При решении задач цифровой обработки сигналов распараллеливание по слоям при увеличении аппаратного ресурса, как ни странно, приводит к нелинейному (сублинейному) росту производительности. Это связано с тем, что увеличение числа каналов обработки требует динамической коммутации, что усложняет не только коммутацию, но и адресацию памяти, что, в свою очередь, способствует разрывам информационных потоков. Поэтому порядок выполнения преобразований для таких задач целесообразно изменить по сравнению с базовой стратегией и первым преобразованием выполнить редукцию по числу каналов связи.

В ряде сложных задач полная реализация базового подграфа со всеми операционными вершинами требует огромного числа информационных каналов, что не позволяет применять распараллеливание по слоям. В этом случае даже при достаточности аппаратного ресурса для реализации базового подграфа его необходимо редуцировать, чтобы сократить количество каналов, которые являются критическим ресурсом.

Стратегия преобразования кадровой структуры входной параллельной программы не фиксирована и в значительной степени определяется информационной структурой задачи и типом информационных зависимостей. Так, для целого ряда задач символьной обработки характерно отсутствие итераций, но число слоев крайне велико, поэтому преобразования итерационной структуры для таких задач лишены смысла. При этом в задачах математической физики рациональная реализация итерационной составляющей кадровой структуры является залогом эффективности решения всей задачи. При решении задач цифровой обработки сигналов зачастую первым преобразованием является редукция количества входных информационных каналов с переходом к м-кадрам и последующим масштабированием полученного решения для эффективного использования аппаратного ресурса.

В ряде случаев аппаратного ресурса РВС может быть недостаточно для структурной реализации базового подграфа. Тогда к базовому подграфу применяются редукционные преобразования [5], что позволяет впоследствии использовать методы масштабирования редуцированной кадровой структуры для сокращения времени решения задачи. Если необходима редукция функционального подграфа задачи, «Ариадна» рассчитывает требуемый коэффициент редукции и вызывает вспомогательный компонент – программу преобразования фрагментов кадровой структуры на языке COLAMO в микро-кадры «Синид». В текущей версии программы «Ариадна» реализованы не все указанные в [5] преобразования – в частности, не поддерживается редукция по разрядности обрабатываемых операндов, которая дает ощутимый для быстродействия эффект только для достаточно узкого класса задач.

Таким образом, в качестве первого преобразования кадровой структуры в стратегии может использоваться изменение любого из пяти параметров параллелизма (L, It, Ch, Op, I), что и определяет различие стратегий. Оценкой сверху для общего количества возможных стратегий тогда является 5! = 120, т.е. их число достаточно невелико даже для полного перебора современными средствами вычислительной техники. На практике, в силу ортогональности преобразований кадровой структуры [5], количество возможных стратегий существенно меньше и едва ли превышает 20. Используемый в «Ариадне» алгоритм синтеза стратегий реализован в соответствии с методикой преобразования (портации) кадровой структуры [4], причем выбор последовательности выполняемых преобразований зависит от информационных зависимостей в структуре задачи, параметров быстродействия кадровой структуры и доступного аппаратного ресурса. Для оптимизации алгоритма синтеза стратегий также использовался многолетний опыт разработки схемотехнических решений специалистов НИЦ СЭ и НК.

Сравнительно небольшое общее число возможных стратегий преобразования кадровых структур позволяет существенно сократить время анализа кадровых структур и время синтеза **гарантированно** рационального решения прикладной задачи: если при полном переборе всех стратегий не был достигнут заданный уровень производительности, будет выбрано лучшее из проанализированных решений.

Сравнение результатов высокоуровневого синтеза шести задач символьной обработки и четырех задач линейной алгебры без использования «Ариадны» и с ее применением показало, что количество синтезируемых конвейерных реализаций не сокращается, а время синтеза для ряда задач сокращается на 15–50 %.

5. Структурная реализация косвенной адресации

Структурная реализация косвенной адресации при потоковой обработке является одной из самых «неудобных» задач, эффективная реализация которой затруднена, поскольку произвольный порядок следования адресов памяти нарушает необходимую для конвейерной организации вычислений регулярность работы контроллеров распределенной памяти. В наиболее общем случае структурная реализация косвенной адресации для получения элемента массива X по косвен-

ному адресу Y[i] представляет собой каскад последовательно соединенных мультиплексоров, количество которых определяется длиной массива. Схема структурной реализации косвенной адресации приведена на рис. 2. Получение элемента X[Y[i]] из массива X сводится к последовательному сравнению поступившего адреса Y[i] с адресами всех элементов массива X, в результате которого сработает только один из мультиплексоров в цепочке. Как видно из рис. 2, структурная реализация косвенной адресации неизбежно приводит к нерациональному использованию аппаратного ресурса и существенному снижению производительности РВС.

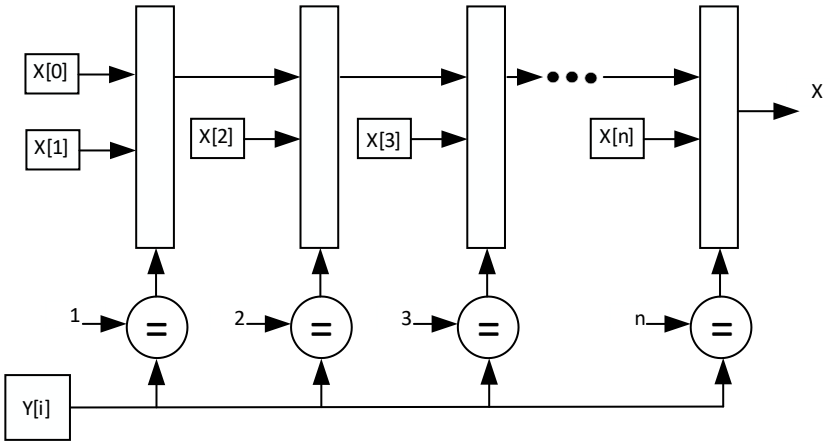

Рис. 2. Структурная реализация косвенной адресации для получения элемента из массива X по косвенному адресу Y[i]

Вместе с тем, на практике нередко встречаются частные случаи, когда для выражений, содержащих косвенную адресацию, возможно сократить время обращения к элементу массива и даже распараллелить. Наиболее частым примером является суммирование с накоплением, когда выполняется доступ к одному и тому же элементу массиву с совпадающими адресами чтения и записи: X[**Y[i]**] := X[**Y[i]**] + const. Выражения такого вида в ряде случаев могут быть распараллелены по адресам, по данным или по адресам и данным одновременно.

Распараллеливание косвенной адресации по адресам состоит в одновременной обработке двух и более адресов **Y[i]** для доступа к нескольким «копиям» исходного массива X (блокам X0 и X1 на рис. 3). При структурной реализации этого метода цепочка мультиплексоров заменяется внутренней памятью, количество блоков которой определяется степенью распараллеливания адресного массива Y. Необходимым условием применения метода распараллеливания по адресам является существование нейтрального элемента для операции в выражении с косвенной адресацией (нейтральный элемент оставляет любой другой элемент неизменным при применении к ним бинарной операции). Для операции арифметического сложения нейтральным элементом является 0, а для операции умножения – единица. Существование нейтрального элемента для выполняемой операции необходимо для предварительного заполнения его значением блоков внутренней памяти X0 и X1 на рис. 3.

При одновременной подаче адресов Y0[i] и Y1[i] выполняется чтение текущих значений из блоков памяти X0 и X1, их обработка (в рассматриваемом примере – сложение с константой K) и запись результата по тому же адресу в блоки внутренней памяти X0 и X1. Адреса в каналах Y0 и Y1 могут совпадать, совпадающие адреса будут обработаны информационно-независимо, но для получения итогового корректного результата, по завершению обработки всех адресов Y0[i] и Y1[i] необходимо объединить накопленные в блоках X0 и X1 частичные результаты – на рис. 3 это выполняется с помощью сумматора во втором ярусе. Учитывая возможную начальную инициализацию массива X, далее необходимо объединить его начальные значения с объединенными частичными результатами, что выполняет сумматор в последнем ярусе.

Для реализации предложенного метода распараллеливания косвенной адресации необходимы как дополнительные каналы распределенной памяти для одновременной подачи адресов Y, так и дополнительные аппаратные затраты в виде блоков внутренней памяти по размеру исходного массива.

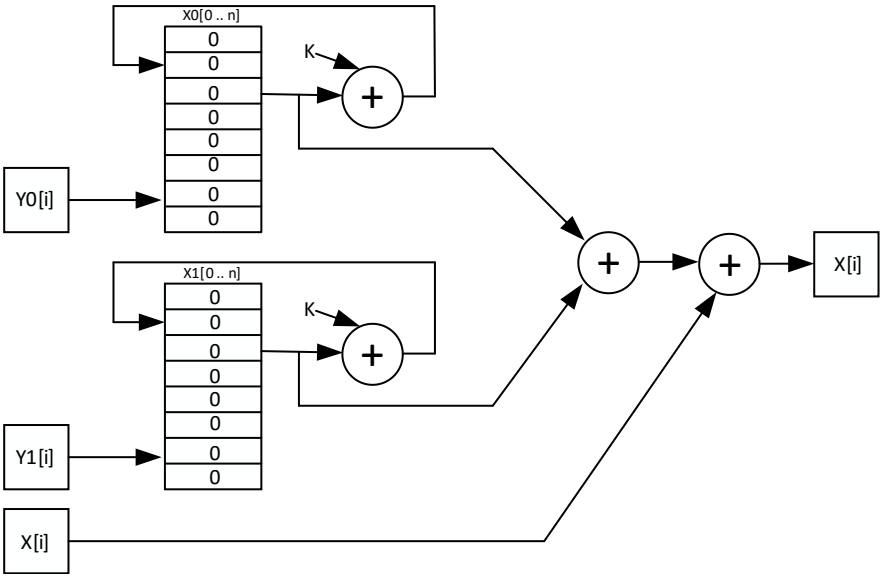

Рис. 3. Структурная реализация косвенной адресации при распараллеливании по адресам

Альтернативой рассмотренному методу является распараллеливание по данным, схема структурной реализации которого приведена на рис. 4. В отличие от рассмотренного на рис. 3 случая, исходный массив X делится на несколько непересекающихся частей, каждая из которых размещается в отдельном блоке внутренней памяти. Количество блоков внутренней памяти соответствует степени распараллеливания и совпадает с количеством входных адресных каналов Y. В отличие от предыдущей схемы подаваемые адреса Y0[i] и Y1[i] подаются не напрямую, а буферизируются в зависимости от своего значения с помощью коммутатора. Буферизация необходима для разрешения коллизии одновременного доступа к памяти, когда одновременно пришедшие адреса обращаются к одному и тому же блоку внутренней памяти.

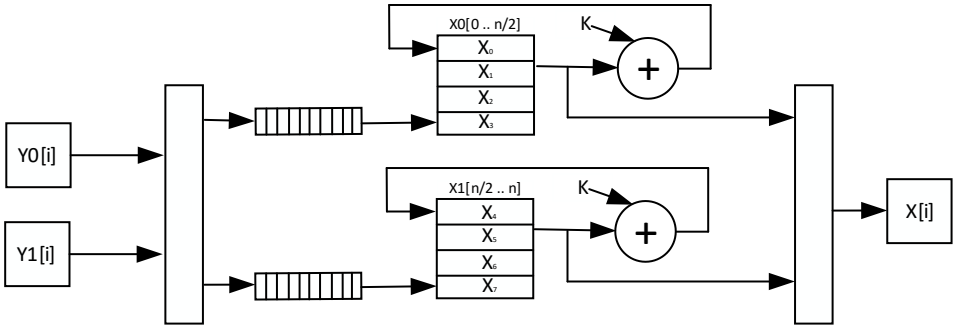

Рис. 4. Структурная реализация косвенной адресации при распараллеливании по данным

Представленная на рис.4 схема требует дополнительных аппаратных расходов на буферизацию, причем рекомендуемый размер каждого буфера в общем случае равен M/2, где M – размерность массива Y. Если распределение адресов в массиве Y заранее неизвестно или существенно отличается от нормального, то использование буферов меньшей длины теоретически может привести к их переполнению. Поэтому при выборе рационального размера буфера, с одной стороны, необходимо минимизировать размер буфера для сокращения аппаратных затрат, а с другой – обеспечить корректность работы, исключив возможность их переполнения. Как правило, размер буфера обычно подбирается эмпирически для каждой прикладной программы с учетом характеристик распределения адресов в массиве Y. В комплексе «Тесей» в настоящий момент реализован вариант с буфером размером M/2 как в наибольшей степени обеспечивающий корректность обработки данных.

Распараллеливание косвенной адресации для вычислительных операций с накоплением результата (суммирование, умножение) является одним из инструментов «Ариадны» для оптимизации организации вычислений и синтеза рациональной кадровой структуры.

6. Заключение

В комплексе средств высокоуровневого синтеза «Тесей», в отличие от известных HLS-компиляторов, входная программа на языке С преобразуется автоматически, без ручной разметки кода или иных указаний пользователя, а результатом является рациональное решение задачи для многокристальной РВС с автоматической синхронизацией информационных и управляющих сигналов.

В настоящей статье описана новая программа «Ариадна», выполняющая синтез стратегий преобразования кадровой структуры для поиска рационального решения исходной прикладной задачи. В отличие от предыдущих версий комплекса [4, 5] «Ариадна» выбирает преобразования кадровой структуры адаптивно, учитывая особенности ее информационных зависимостей и базового подграфа, что позволяет синтезировать рациональные кадровые структуры для различных предметных областей: символьной обработки, линейной алгебры, цифровой обработки сигналов.

Направления дальнейших исследований связаны с расширением классов и диапазонов решаемых задач и сокращением ряда требований к исходному тексту программ на С, в частности, поддержки косвенной адресации во входной программе и операторов динамического выделения памяти для массивов.

Литература

1. Nane R., Sima V.-M., Pilato C. et al. A Survey and Evaluation of FPGA High-Level Synthesis Tools // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2016. Vol. 35, no. 10. P. 1591–1604. DOI: 10.1109/TCAD.2015.2513673.
2. Numan M.W., Phillips B.J., Puddy G.S., Falkner K. Towards Automatic High-Level Code Deployment on Reconfigurable Platforms: A Survey of High-Level Synthesis Tools and Toolchains // IEEE Access. 2020. Vol. 8. P. 174692–174722. DOI: 10.1109/ACCESS.2020.3024098.
3. Каляев И.А., Левин И.И. Реконфигурируемые вычислительные системы на основе ПЛИС: монография. Ростов-на-Дону: Издательство ЮНЦ РАН, 2022. 474 с. ISBN 978-5-4358-0232-0
4. Дордопуло А.И., Левин И.И., Гудков В.А., Гуленок А.А. Программный комплекс высокоуровневого синтеза конфигурационных файлов для многокристальных реконфигурируемых вычислительных систем // Параллельные вычислительные технологии (ПаВТ'2023): Короткие статьи и описания плакатов. Материалы XVII всероссийской научной конференции с международным участием, Санкт-Петербург, 28-30 марта 2023 года. Челябинск: Издательский центр ЮУрГУ, 2023. С. 133–142.
5. Дордопуло А.И., Левин И.И., Гудков В.А., Гуленок А.А. Программные средства высокоуровневого синтеза для многокристальных реконфигурируемых вычислительных систем // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2022. Т. 11, № 3. С. 5–21. DOI: 10.14529/cmse220301.
6. Nane R., Sima V.-M., Olivier B. et al. DWARV 2.0: A CoSy-based C-to-VHDL Hardware Compiler // FPL. 2012. P. 619–622. DOI: 10.1109/FPL.2012.6339221.
7. Pilato C., Ferrandi F. Bambu: A Modular Framework for the High Level Synthesis of Memory-intensive Applications // FPL. 2013. P. 1–4. DOI: 10.1109/FPL.2013.6645550.
8. Canis A., Choi J., Aldham M. et al. LegUp: High-Level Synthesis for FPGA-based Processor/Accelerator Systems // ACM FPGA. 2011. P. 33–36. DOI: 10.1145/1950413.1950423.
9. Make Slow Software Run Fast with Vivado HLS. URL: https://www.xilinx.com/publications/xcellonline/run-fast-with-Vivado-HLS.pdf (дата обращения: 23.03.2023)
10. Vitis Unified Software Platform Documentation. Application Acceleration Development. URL: https://www.xilinx.com/support/documentation/sw_manuals/ xilinx2019_2/ug1393-vitis-application-acceleration.pdf (дата обращения: 23.03.2023).

11. Kamkin A.S., Chupilko M.M., Lebedev M.S. et al. Comparison of High-Level Synthesis and Hardware Construction Tools // Trudy ISP RAN/Proc. ISP RAS. 2022. Vol. 34, no. 5. P. 7–22. DOI: 10.15514/ISPRAS-2022-34(5)-1.
12. Liang Y., Rupnow K., Li Y. et al. High-Level Synthesis: Productivity, Performance, and Software Constraints // Journal of Electrical and Computer Engineering. 2012. Article 1. DOI: 10.1155/2012/649057.
13. Zhou Y., Gupta U., Dai S. et al. Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs // Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '18). Association for Computing Machinery, New York, NY, USA, 2018. P. 269–278. DOI: 10.1145/3174243.3174255.