Раздел III. Реконфигурируемые вычислительные системы

УДК 004.442.2 DOI 10.18522/2311-3103-2020-7-94-106

А.И. Дордопуло, И.И. Левин, В.А. Гудков, А.А. Гуленок, А.В. Бовкун, С.А. Дудко

КОМПЛЕКС СРЕДСТВ ТРАНСЛЯЦИИ ПРОГРАММ НА ЯЗЫКЕ С В ПРОГРАММЫ НА ЯЗЫКЕ ПОТОКА ДАННЫХ COLAMO

Рассматриваются программные средства трансляции последовательных программ на языке С в масштабируемые параллельно-конвейерные программы на языке программирования реконфигурируемых вычислительных систем COLAMO. В отличие от существующих средств высокоуровневого синтеза, результатом трансляции является не IP-ядро фрагмента задачи, а комплексное решение задачи для многокристальных реконфигурируемых вычислительных систем с автоматической синхронизацией информационных и управляющих сигналов. Рассмотрены основные этапы трансляции последовательной программы на языке С: преобразование в информационный граф, анализ информационных зависимостей и выделение функциональных подграфов, преобразование в масштабируемую ресурсонезависимую параллельно-конвейерную форму и масштабирование программы на языке COLAMO для заданной многокристальной реконфигурируемой вычислительной системы. Масштабирование программы осуществляется с помощью методов редукции производительности абсолютно-параллельной формы задачи – информационного графа, который адаптируется под архитектуру реконфигурируемой вычислительной системы. Разработан ряд правил, позволяющих существенно сократить число шагов преобразований при масштабировании задачи и обеспечить плотный поток обработки данных в функциональных подграфах задачи. Созданный комплекс средств трансляции программ на языке С в конфигурационные файлы ПЛИС позволяет существенно сократить время синтеза вычислительной структуры задачи для многокристальных РВС и обеспечить сокращение общего времени решения задачи.

Информационные графы; компилятор; трансляция программ; язык С; редукция производительности; реконфигурируемые вычислительные системы; программирование многопроцессорных вычислительных систем.

A.I. Dordopulo, I.I. Levin, V.A. Gudkov, A.A. Gulenok, A.V. Bovkun, S.A. Dudko

HIGH-LEVEL TOOLS FOR TRANSLATION OF C-APPLICATIONS INTO APPLICATIONS IN DATAFLOW LANGUAGE COLAMO

In the paper we review software tools for translation of sequential C-programs into scalable parallel-pipeline programs written in the COLAMO language, used for programming of reconfigurable computer systems. In contrast to existing tools of high-level synthesis, the translation result is not an IP-core of a task fragment, but a complex task solution for multichip reconfigurable computer systems with automatic synchronization of data and control signals. We analysed the main translation steps of a sequential C-program such as transformation into an information graph, analysis of data dependencies and selection of functional subgraphs, transformation into a scalable resource-independent parallel-pipeline form, and scaling a COLAMO-program for a specified multichip reconfigurable computer system. A program is scaled with the help of performance reduction methods, applied to a completely parallel form of a task (an information graph), adapted to the architecture of a reconfigurable computer system. We developed several rules,

significantly reducing the number of transformation steps of task scaling, and providing a continuous flow of data processing in the functional subgraphs of the task. The developed software tools for translation of C-programs into FPGA configuration files significantly decrease the synthesis time of a task computing structure for multichip RCSs and the total task solution time.

Information graph; compiler; translation of programs; C language; performance reduction; reconfigurable computer system; programming of multiprocessor computer systems.

Введение. Сокращение времени решения задачи является основной прагматичной целью высокопроизводительных вычислений. Ускорение вычислений достигается повышением быстродействия вычислительных узлов многопроцессорной вычислительной системы (МВС), максимальным распараллеливанием вычислительных операций или сочетанием обоих подходов [1]. На практике максимальное распараллеливание, как правило, недостижимо, потому что возможности распараллеливания вычислительных операций задачи ограничены соотношением между размерностью задачи и доступным аппаратным ресурсом вычислительной системы. Поэтому при реализации прикладной задачи в МВС разработчик выполняет поиск наиболее рационального варианта ее реализации, обеспечивающего минимальное время решения с учетом характеристик доступного аппаратного ресурса.

Для МВС, построенных на основе процессорной архитектуры, задача представляется в парадигме *передачи управления* от одного процесса (или вычислительного устройства) к другому, на которых они реализуются последовательно с помощью команд процессора. Характерные для большинства задач информационные зависимости по данным при передаче управления могут быть нарушены, что не позволяет достичь ускорения вычислений, пропорционального числу используемых узлов. Поэтому на реальных вычислительных задачах производительность таких МВС существенно сокращается до 10–15% от пиковой.

Учет информационных зависимостей в структуре прикладной задачи обеспечивается в концепции структурно-процедурной организации вычислений (СПОВ), что позволяет обеспечить высокую реальную производительность вычислений на реконфигурируемых вычислительных системах (РВС) [2] с программируемыми логическими интегральными схемами (ПЛИС) [2], которые находят все больше применений при решении вычислительно трудоемких задач в различных областях науки и техники [3]. РВС обладают существенными преимуществами в реальной производительности и энергоэффективности по сравнению с МВС кластерной архитектуры, но их широкое применение во многом сдерживается высокой сложностью программирования. Сложность эффективного программирования РВС на основе ПЛИС, признанная многими исследователями [1–3], несмотря даже на наличие языков программирования высокого уровня (HandelC[4], SystemC[4], COLAMO[2]), побуждает искать решения в области разработки и создания средств автоматической трансляции последовательных программ (например, на языке С) в конфигурационные файлы ПЛИС.

Принципы и методы создания средств высокоуровневого синтеза Трансляторы программ на языке С в конфигурационные файлы ПЛИС [4–9] получили названия средств высокоуровневого синтеза (High Level Synthesis), которые по типу входного языка программирования могут быть разделены на две основные категории [4]: трансляторы проблемно-ориентированных языков (Domain Specific Languages – адаптированной к определенной проблемной области версии языка программирования С) и трансляторы языков общего назначения (General Purpose Languages – диалекты языка программирования С с некоторыми особенностями и ограничениями). В настоящее время активно развиваются как академические (DWARV [5], BAMBU [6] и LEGUP [7]), так и коммерческие (CatapultC, Vivado HLS [8], Vivado Vitis[9]) комплексы проектирования. Подробный и детальный обзор существующих средств высокоуровневого синтеза и сравнительный анализ их

возможностей при трансляции задач некоторых предметных областей приведен в [4]. Подавляющее большинство приведенных в [4] трансляторов программ на языке С преобразуют вычислительно трудоемкий фрагмент задачи в IP-ядро и синтезируют в ПЛИС специализированный вычислитель на основе автоматной модели или процессорной парадигмы. Несмотря на существенный выигрыш в скорости вычислений по сравнению с процессорной реализацией (по данным обзора [4]), используемые для решения задачи IP-ядра реализуют только лишь фрагмент задачи и для создания завершенного решения необходимо участие высококвалифицированного программиста. Масштабирование решения, даже в пределах одного кристалла ПЛИС, также полностью возлагается на программиста.

В отличие от рассмотренных в [4] трансляторов программ на языке С, разрабатываемый в НИЦ супер-ЭВМ и нейрокомпьютеров комплекс средств трансляции программ на языке С в программы на языке потока данных COLAMO исходно предназначен для синтеза комплексного решения прикладной задачи для РВС, содержащих вычислительное поле ПЛИС, связанных пространственной коммутационной системой. Комплекс отображает последовательную программу на языке С в абсолютно-параллельную форму, которую с помощью редукционных преобразований адаптирует под доступный аппаратный ресурс многокристальных РВС с учетом информационных связей и зависимостей данных задачи. При схожей функциональности от наиболее близких аналогов (Xilinx Vivado HLS [8] и Xilinx Vitis [9]) комплекс отличается не только поддержкой многокристальных решений, но и автоматической синхронизацией информационных и управляющих сигналов.

Прикладная задача на последовательной языке программирования С компонентами комплекса преобразуется в абсолютно параллельную форму – информационный граф [1, 2], отражающий естественный параллелизм и конвейеризацию операций, который может быть адаптирован к архитектуре и текущей конфигурации РВС с помощью методов масштабирования (индукции и редукции) производительности кадровой структуры [2]. Масштабирование задачи (рис. 1) представляется движением в трехмерном пространстве: слоев (информационно-независимые реализации базового подграфа), итераций (информационно-зависимые реализации базового подграфа) и команд (вычислительные устройства заданной разрядности, составляющие базовый подграф).

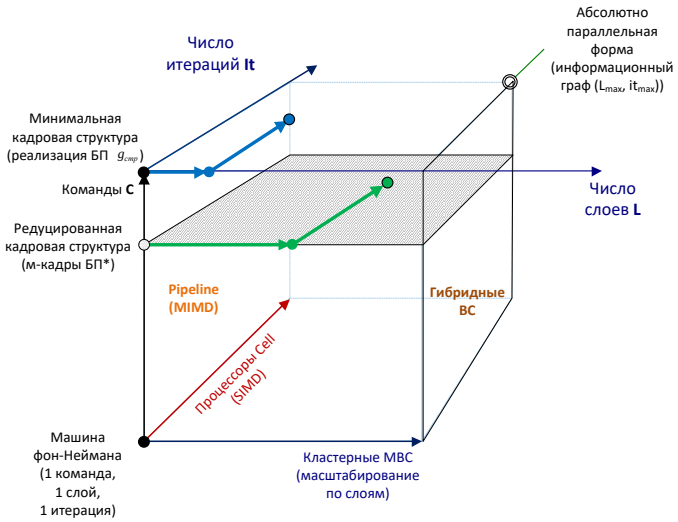

Рис. 1. Пространство реализаций вычислений для различных архитектур

В представленной на рис. 1 модели пространства возможных решений задачи реализации вычислений на процессоре соответствует точка «Машина фон-Неймана (1 команда, 1 слой, 1 итерация)», а характерное для кластерных МВС увеличение числа процессоров при реализации информационно-независимых фрагментов вычислений соответствует движению по оси «Число слоев L». Конвейеризация вычислительных операций при реализации информационно-зависимых фрагментов вычислений, характерная для вычислителей с архитектурой SIMD, примером которой являются процессоры Cell, соответствует движению от этой точки по оси «Число итераций It». Заданные осями «Команды C» и «Число итераций It» параллельные плоскости «Pipeline(MIMD)» и «Гибридные ВС» описывают реализацию вычислений для MIMD-вычислителей и гибридных вычислительных систем с графическими ускорителями.

В парадигме СПОВ [2] масштабирование задачи осуществляется в плоскости слоев и итераций движением от минимальной кадровой структуры, реализующей базовый подграф (БП) задачи $g_{стр}$, к абсолютно-параллельной форме задачи – кадровой структуре $(L_{\max}, It_{\max})$, соответствующей информационному графу задачи. Масштабирование выполняется индуктивным тиражированием минимальной кадровой структуры, реализующей БП $g_{стр}$. Минимальная кадровая структура, как правило, является структурной конвейерной реализацией БП $g_{стр}$, синтезированной инженером-схемотехником вручную для заданного кристалла ПЛИС, оптимальной как по числу каналов, так и по интервалу обработки данных. Поэтому автоматическое масштабирование решения задачи для структурно-процедурной организации вычислений затруднено необходимостью разработки методов и средств не только автоматического выделения БП $g_{стр}$, но и его оптимальной конвейерной реализации.

В рассматриваемом комплексе средств трансляции программ на языке С в программы на языке потока данных COLAMO используется другой подход. Масштабирование начинается не от минимальной кадровой структуры, а от абсолютно-параллельной кадровой формы задачи $(L_{\max}, It_{\max})$, соответствующей информационному графу задачи. Движение осуществляется с помощью методов редукции производительности и аппаратных затрат [10], что не требует ручного выделения БП $g_{стр}$ и позволяет находить решение для случаев, когда доступного аппаратного ресурса недостаточно для реализации даже минимальной кадровой структуры $g_{стр}$. В этом случае она редуцируется до м-кадров (микро-кадров, заштрихованная плоскость на рис. 1), последовательно выполняющих часть команд БП $g_{стр}$ на меньшем аппаратном ресурсе. М-кадры, как и минимальную кадровую структуру, можно индуктивно масштабировать в плоскости слоев и итераций для получения рационального решения.

Принципы масштабирования программы на языке С под доступный аппаратный ресурс при трансляции в COLAMO. Методику автоматического масштабирования задачи в пространстве возможных кадровых структур для многокристальных РВС при трансляции программы на языке С в программы на языке потока данных COLAMO можно представить следующими последовательными преобразованиями.

1. Построение информационного графа задачи – преобразование входной программы на языке С в абсолютно-параллельную форму, описанную синтаксическими конструкциями COLAMO.

2. Анализ структуры информационного графа задачи: выделение подзадач, анализ информационных зависимостей в структуре каждой подзадачи и между ними, определение числа слоев и итераций для всех фрагментов всех подзадач, расщепление скалярных переменных и растягивание массивов по итерациям для устранения нарушений правил однократного присваивания и единственной подстановки.

3. Преобразование задачи в масштабируемую параллельно-конвейерную форму: добавление потоковой составляющей ко всем векторным измерениям массивов и циклам обработки в программе на COLAMO, расчет возможных значений параметров масштабирования по аппаратному ресурсу и числу каналов памяти.

4. Масштабирование задачи: движение от абсолютно-параллельной формы по слоям, итерациям и командам в соответствии с рассчитанными параметрами с помощью методов редукции производительности, расчет и подбор значений параметров масштабирования для рационального использования доступного ресурса РВС.

5. Оптимизация полученного решения задачи по критерию минимума времени: сокращение интервала обработки данных с помощью преобразования к конвейеру конвейеров или макроконвейеру [2].

Выполнение первых трех пунктов приведенной методики позволяет получить масштабируемую ресурсонезависимую параллельную прикладную программу на языке COLAMO, четвертый пункт методики адаптирует полученную программу под доступный вычислительный ресурс, а пятый пункт синтезирует рациональное (по времени выполнения) решение задачи. Далее синтезированная программа на языке программирования COLAMO транслируется в схемотехническую конфигурацию с помощью разработанных транслятора COLAMO и синтезатора Fire!Constructor [10] с автоматическими распределением по кристаллам ПЛИС и синхронизацией вычислений между ПЛИС.

Каждый из пунктов методики является функционально-завершенным преобразованием, которое выполняется отдельным программным компонентом комплекса средств трансляции программ на языке С в программы на языке потока данных COLAMO.

Структура комплекса средств трансляции. Разработанная с учетом описанной методики структурная схема комплекса средств трансляции представлена на рис. 2. Трансляцию из С в COLAMO выполняют следующие программные компоненты:

- транслятор «Ангел», преобразующий программы на языке программирования «С» в стандарте ISO/IEC 9899:1999 в абсолютно-параллельную программу на языке Colamo;
- процессор «Русалка», преобразующий абсолютно-параллельную программу на языке Colamo в ресурсонезависимую программу на языке Colamo;
- процессор «Прокруст», масштабирующий ресурсонезависимую программу на языке Colamo (адаптирующий параметры программы под архитектуру РВС);
- процессор «Щелкунчик», осуществляющий редукцию производительности программы при нехватке аппаратного ресурса.

Ранее разработанные транслятор языка программирования COLAMO и синтезатор многокристальных решений Fire!Constructor транслируют полученную программу на COLAMO и создают многокристальное схемотехническое решение – VHDL-файлы для всех используемых при реализации задачи кристаллов ПЛИС РВС.

Рис. 2. Структурная схема комплекса средств трансляции последовательных программ на языке С в конфигурационные файлы ПЛИС

Синтез загрузочных конфигурационных файлов (*.bit) осуществляется синтезатором системы автоматизированного проектирования Xilinx Vivado для каждого кристалла в отдельности.

Основные этапы трансляции последовательной программы на языке С в параллельную программу на COLAMO. Рассмотрим основные этапы преобразования программы компонентами комплекса.

1. Построение информационного графа задачи. Исходная программа на языке С транслятором «Ангел» преобразуется в абсолютно-параллельную форму – информационный граф – совокупность множеств входных, выходных и операционных вершин задачи, соединенных между собой информационными связями (дугами). Он не содержит циклов, все операционные вершины задачи представлены заданное размерностью задачи число раз.

Информационный граф задачи (ИГЗ) описывается конструкциями языка программирования COLAMO – все массивы исходной последовательной программы становятся мемориальными массивами с параллельным (векторным) типом доступа [2] по всем измерениям. Циклы исходной программы задают слои и итерации для подграфов, являющихся телом цикла. В слоях расположены изоморфные функционально-завершенные информационно независимые подграфы, а итерации характеризуют информационную зависимость данных во времени. Подграфы, расположенные в одном слое, информационно независимы (между ними отсутствуют дуги), а подграфы, расположенные на разных итерациях, зависимы по обрабатываемым данным во времени. На основе структуры исходной последовательной программы (циклов, функций и процедур) ИГЗ представляется в виде объединения подзадач, в каждой из которых выделяется базовый подграф. В базовом подграфе каждой подзадачи выделяются один или несколько функциональных подграфов (ФП) – циклов исходной программы – фрагментов вычислений с заданными функциями масштабирования по слоям и итерациям. Так, для последовательной программы решения систем линейных алгебраических уравнений (СЛАУ) методом Гаусса для обусловленной матрицы можно выделить две подзадачи со своими базовыми подграфами: прямой и обратный ход. Прямой ход, как правило, представляется тройным циклом, а обратный ход – одним циклом по числу строк матрицы. Для БП прямого хода функциональными подграфами будут операторы расчета нормировочного коэффициента строки и вычитание произведения нормировочного множителя и элемента ведущей строки из элемента столбца матрицы. Число итераций циклов по каждой переменной соответствует числу слоев и итераций.

2. Анализ структуры информационного графа задачи (информационных зависимостей исходной программы). На этом этапе транслятор «Ангел» преобразует последовательную программу с произвольным обращением к памяти к параллельной программе, оперирующей потоками данных. Разделение по слоям и итерациям выполняется на основе анализа циклов [12–17] с учетом информационных зависимостей переменных и массивов данных. Для этого в выделенных функциональных подграфах анализируются информационные зависимости и проверяется соблюдение правил единственной подстановки [18] и однократного присваивания, нарушения которых устраняются с помощью расщепление скалярных переменных и растягивание массивов по итерациям [19]. После выполнения всех преобразований создается синтаксически корректная программа на языке COLAMO в абсолютно-параллельной форме с выделенными разметкой **#FuncGraph** функциональными подграфами вычислений.

3. Преобразование задачи в масштабируемую параллельно-конвейерную форму. Масштабируемая параллельно-конвейерная форма, преобразование к которой выполняется процессором «Русалка», позволяет с помощью одной константы (степени параллелизма – числа одновременно реализованных подграфов) автоматически пересчитывать длину обрабатываемых потоков данных для каждого из реализованных подграфов. Для этого каждое измерение всех массивов программы должно быть представлено двумя взаимосвязанными составляющими – векторной и потоковой с параллельным и последовательным типами доступа соответственно. Поэтому все созданные транслятором «Ангел» векторные измерения массивов дополняются связанной с ними потоковой составляющей таким образом, чтобы произведение векторной и потоковой размерностей было в точности равно длине исходного массива в последовательной программе. Также в тексте программы на COLAMO добавляются циклы для обработки потоковой составляющей, границы которых связываются с векторными циклами аналогичным образом. В результате

этого преобразования абсолютно-параллельная программа на языке COLAMO становится параллельно-конвейерной, в которой можно управлять параллелизмом одним параметром с автоматическим пересчетом длины потоковой составляющей и соблюдением синтаксической корректности.

4. Масштабирование задачи. Расчет параметров масштабирования по аппаратному ресурсу и числу каналов памяти, редукция производительности функциональных подграфов задачи и согласование интенсивности потоков данных выполняются процессором «Прокруст» (и, при необходимости, процессором «Щелкунчик», которому отводится вспомогательная роль) и являются, пожалуй, самой сложной частью рассматриваемой методики.

Процессор «Прокруст» рассчитывает и подбирает параметры рациональной реализации всех функциональных подграфов задачи, после чего преобразует масштабируемую параллельно-конвейерную форму с помощью методов редукции производительности. Теоретические положения редукции производительности подробно рассмотрены в [11], а алгоритм синтеза рациональной вычислительной структуры ИГЗ для заданной РВС – в [10].

Для сбалансированного масштабирования и редукции производительности задачи необходимо обеспечить сокращение производительности всех подзадач в одинаковое число раз, заданное коэффициентом редукции производительности. Наиболее трудоемким этапом масштабирования является согласование темпов обработки данных во всех функциональных подграфах ИГЗ и выбор для каждого функционального подграфа наиболее рациональной формы организации вычислений с учетом вычислительной структуры других подграфов и задачи в целом. Для решения этой задачи разработан ряд правил, позволяющих сократить число анализируемых вариантов и общее время решения задачи.

Масштабирование и редукция производительности информационного графа задачи начинается с «флагмана» – ФП, который занимает наибольший аппаратный ресурс (произведение занимаемого телом цикла ресурса и числа итераций цикла). Остальные ФП будем считать «катерами» – сравнительно небольшими по аппаратным затратам ФП подзадачи/задачи, занимающими существенно меньший по сравнению с «флагманом» ресурс. Как правило, «флагманом» является цикл с наибольшим числом итераций и/или с самым «тяжеловесным» телом: аппаратные затраты на реализацию «флагмана» существенно (не менее, чем на один десятичный порядок) превышают аппаратные затраты на реализацию «катера». Рациональная аппаратная реализация «флагмана» крайне важна, т.к. именно «флагман» вносит наибольший вклад в общее время решения задачи, поэтому чем ближе его реализация к оптимальной, тем меньше время решения задачи в целом. Все остальные ФП («катера») при масштабировании согласовываются с ним по числу каналов памяти, аппаратным затратам и интенсивности потоков данных.

Прикладные задачи (рис. 3) содержат либо «флагман», либо «катер», а чаще всего – обе подзадачи, связанные различными типами информационной зависимости. При анализе аппаратных ресурсов ФП и базового подграфа целесообразно сравнивать их с минимальным ресурсом, который заведомо реализуется в одном кристалле ПЛИС, чтобы не масштабировать ФП методами редукции по числу устройств и разрядам. Если ФП помещается в заданный минимальный ресурс, то его можно реализовать структурно без масштабирования. В самом ФП, если он содержит несколько выражений (операторов), используется аналогичная методика – в первую очередь масштабируется (редуцируется) самое большое по занимаемому аппаратному ресурсу («тяжеловесное») выражение.

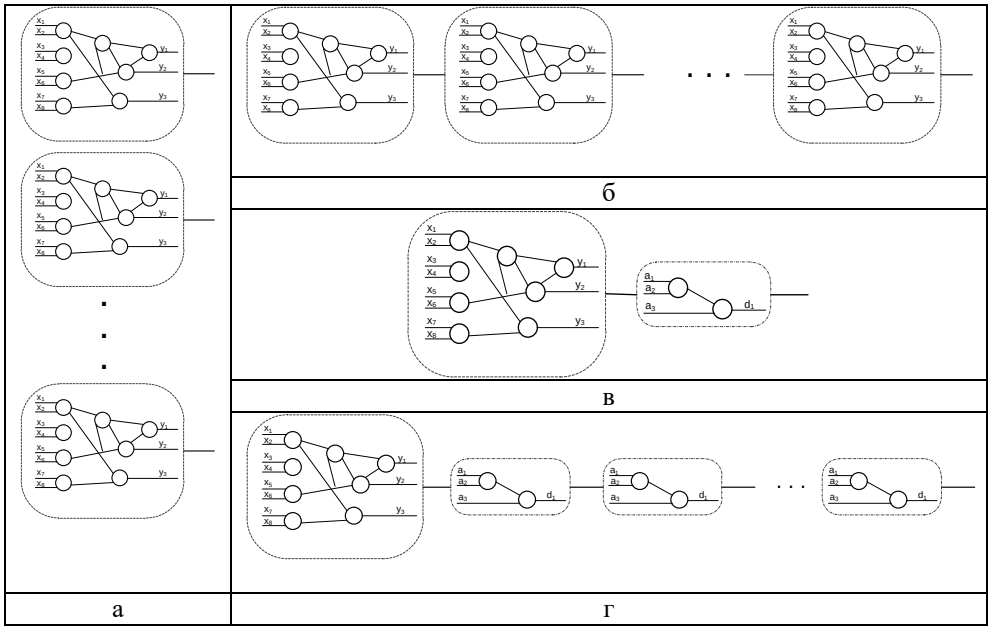

Рис. 3. Структуры информационных графов прикладных задач: а – информационно-независимые «флагманы»; б – информационно-зависимые «флагманы»; в – «флагман»-«катер»; г – «флагман»-множество «катеров»

Один из основных приемов при редукции «флагмана» – сохранить неизменным (нередуцируемым) итерационный цикл, поскольку это сокращает время решения задачи, не увеличивает число каналов памяти и задействует аппаратный ресурс, которого достаточно много. Если это невозможно, то реализуются несколько итерационных ступеней, связанных обратной связью, что приводит к существенному увеличению интервала обработки данных, который можно сократить с помощью оптимизационных преобразований (п.5). Основным нежелательным свойством роста интервала обработки данных является его влияние не только на один вычислительный фрагмент, а распространение на всю задачу, чтобы обеспечить единую интенсивность обработки данных.

«Катер» с маленьким весом (не превышающим 0,05) целесообразно реализовывать структурно, без применения методов редукции, сокращающих аппаратные затраты (по числу устройств и разрядности), поскольку такие ФП не вносят существенного вклада в превышение аппаратного ресурса задачи. Для достижения заданного коэффициента редукции производительности ФП с маленьким весом предпочтительнее использовать интервал обработки данных. При наличии информационной зависимости между слоями (в случае функционально-нерегулярного графа) необходимо редуцировать ФП до процедурной реализации.

Для того чтобы задача масштабировалась (то есть пропорционально увеличивалась или уменьшалась по всем подзадачам), необходимо обеспечить не только одинаковую величину редукции для всех подзадач, но и пропорциональное изменение интенсивности потоков данных между подзадачами. Применительно к редукции это означает, что (если это возможно) для сохранения интенсивности потоков данных целесообразно использовать одинаковые виды редукции с одинаковыми коэффициентами. При использовании разных видов редукции в разных подзадачах в общем случае требуется согласовать интенсивность потоков данных,

что, как правило, приводит к дополнительным аппаратным затратам (структурная реализация блоков согласования – элементы задержки с мультиплексорами/демультиплексорами, буфера, внутренняя двухпортовая память (BRAM)) и увеличивает время решения задачи.

Для согласования интенсивности потоков данных используются методы оптимизации, сокращающие увеличившийся в результате редукции интервал обработки данных (скважность) до сбалансированного по всем подграфам наименьшего значения (в наилучшем случае – до минимального значения, равного 1).

5. Оптимизация полученного решения. Сокращение интервала обработки данных задачи по критерию минимума времени решения выполняется с помощью преобразования к конвейеру конвейеров или макроконвейеру. Необходимость в сокращении интервала обработки данных возникает в ряде случаев, например, когда число реализованных итерационных ступеней меньше, чем требуется для реализации итерации целиком – в этом случае неизбежно возникает обратная связь и возрастает интервал обработки данных.

Интервал обработки данных при обратной связи равен отношению латентности реализованных итерационных ступеней к числу регистров в обратной связи. Поэтому, если невозможно увеличить число реализованных итерационных ступеней, для сокращения интервала обработки и создания плотного потока данных необходимо увеличить число регистров в обратной связи до значения латентности – в этом и состоит преобразование к конвейеру конвейеров (вложенному конвейеру, конвейеру в конвейере). В этом случае можно сократить интервал обработки данных до минимального значения, равного 1, что особенно важно при масштабировании «флагмана», вносящего наибольший вклад в общее время решения задачи. Преобразование к макроконвейеру аналогично по своей цели – сокращение интервала обработки данных, но применяется для процедурно реализованных фрагментов и состоит в увеличении их числа до значения, численно равного числу тактов работы одного процедурного устройства – в этом случае каждый такт будет освобождаться как минимум один блок для обработки очередной порции данных, которые подаются плотным потоком каждый такт, с единичным интервалом обработки данных.

Представленная система ограничений и оптимизаций позволяет синтезировать рациональную организацию вычислений (конвейер в конвейере, макроконвейер) для наиболее вычислительно-трудоемкого функционального подграфа «флагмана», что обеспечивает сокращение общего времени решения задачи, поскольку «катера» будут согласовываться именно с ним.

Результаты экспериментальных исследований разработанного комплекса. С помощью созданной экспериментальной версии комплекса средств трансляции успешно транслированы и реализованы на РВС «Терциус» [20, 21] описанные на языке С следующие прикладные программы линейной алгебры: решение систем линейных алгебраических уравнений (СЛАУ) методом Гаусса для матриц размером 8000×8000, решение СЛАУ методом Якоби для матриц размером 8000×8000 (табл. 1), решения СЛАУ с помощью верхней треугольной и нижней треугольной матриц (LU-разложение) для матриц размером 8000×8000.

Таблица 1

Число итерационных ступеней решения СЛАУ методом Якоби	Время решения		
	ПК	РВС	Ускорение
20 ступеней	0.00288	0.00268	1
200 ступеней	0.0292	0.00345	8,4
950 ступеней	0.2895	0.0056	51

Также с высокой эффективностью синтезированы конфигурационные файлы ПЛИС для ряда прикладных задач символьной обработки, использовавшихся в качестве тестов CHStone [4]: удельная производительность полученных решений составляет не менее 85 % от созданных прикладными программистами на языке COLAMO и не менее 70 % от прикладных решений, разработанных специалистами-схемотехниками. Время трансляции последовательных программ на языке С в программы на языке программирования высокого уровня COLAMO и синтеза конфигурационных файлов ПЛИС (без синтеза загрузочных конфигурационных bit-файлов) не превышает 30 минут.

Заключение. Разработанный комплекс средств трансляции реализует оригинальную методику преобразования последовательных программ на языке С в конфигурационные файлы ПЛИС на основе масштабирования информационного графа, описанного в синтаксисе языка COLAMO, для заданной многокристальной РВС. Масштабирование программы выполняется процессором «Прокруст» с помощью методов редукции производительности и аппаратных затрат, что позволяет синтезировать рациональную организацию вычислений и обеспечить сокращение общего времени решения задачи. Полученные при трансляции ряда прикладных задач экспериментальные результаты позволяют сделать вывод о том, что комплекс средств трансляции программ на языке С в программы на языке потока данных COLAMO позволяет существенно сократить время синтеза решения задачи для многокристальных РВС.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 608 с. – ISBN 5-94157-160-7.
2. *Гузик В.Ф., Каляев И.А., Левин И.И.* Реконфигурируемые вычислительные системы. – Таганрог: Изд-во ЮФУ, 2016. – 472 с.
3. *Trimberger S.M.* Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology // in Proceedings of the IEEE. – March 2015. – Vol. 103, No. 3. – P. 318-331. – Doi: 10.1109/JPROC.2015.2392104.
4. *Nane R. et al.* A Survey and Evaluation of FPGA High-Level Synthesis Tools // in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – Oct. 2016. – Vol. 35, No. 10. – P. 1591-1604. – Doi: 10.1109/TCAD.2015.2513673.
5. *Nane R., Sima V.-M., Olivier B., Meeuws R., Yankova Y., Bertels K.* DWARV 2.0: A CoSy-based C-to-VHDL Hardware Compiler // In FPL. – 2012. – P. 619-622.
6. *Pilato C. and Ferrandi F.* Bambu: A Modular Framework for the High Level Synthesis of Memory-intensive Applications // In FPL. 2013. – P. 1-4.
7. *Canis A., Choi J., Aldham M., Zhang V., Kammoona A., Anderson J.H., Brown S., Czajkowski T.* LegUp: High-Level Synthesis for FPGA-based Processor/Accelerator Systems // In ACM FPGA. – 2011. – P. 33-36.
8. Make Slow Software Run Fast with Vivado HLS. – https://www.xilinx.com/publications/xcellonline/run-fast-with-Vivado-HLS.pdf.
9. Vitis Unified Software Platform Documentation. Application Acceleration Development. – https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug1393-vitis-application-acceleration.pdf (дата обращения: 10.11.2020).
10. *Levin Ilya, Dordopulo Alexey, Gudkov Vyacheslav, Gulenok Andrey, Bovkun Alexander, Yevstafiyev Georgyi, Alekseev Kirill.* Software Development Tools for FPGA-Based Reconfigurable Systems Programming // In: Communications in Computer and Information Science, Vol. 1129, Chapter Parallel Computing Technologies. – 2019. – P. 1-16.
11. *Левин И.И., Дордопуло А.И.* К вопросу об автоматическом создании параллельных прикладных программ для реконфигурируемых вычислительных систем // Вычислительные технологии. – 2020. – Т. 25, № 1. – С. 66-81.
12. *Morvan A., Derrien S. and Quinton P.* Efficient nested loop pipelining in high level synthesis using polyhedral bubble insertion // 2011 International Conference on Field-Programmable Technology, New Delhi, 2011. – P. 1-10. – Doi: 10.1109/FPT.2011.6132715.

13. *Jensen, Nicklas, Karlsson, Sven*. Improving Loop Dependence Analysis // ACM Transactions on Architecture and Code Optimization. – 2017. – Vol. 14 (3). – P. 1-24. – Doi: 10.1145/3095754.
14. *Solihin Yan*. Fundamentals of parallel computer architecture*:* multichip and multicore systems. – Chapman and Hall/CRC, 2016. – ISBN 978-1-4822-1118-4.
15. *Cooper, Keith D., Torczon, Linda*. Engineering a Compiler. – Morgan Kaufmann, 2005. – ISBN 1-55860-698-X.
16. *Kennedy Ken, Allen Randy*. Optimizing Compilers for Modern Architectures. A Dependence-based Approach. – Morgan Kaufmann, 2001. – ISBN 1-55860-286-0.
17. *Muchnick Steven S.* Advanced Compiler Design and Implementation. – Morgan Kaufmann, 1997. – ISBN 1-55860-320-4.
18. Векторизация программ // Векторизация программ: теория, методы, реализация: Сб. переводов статей. – М.: Мир, 1991. – С. 246-267.
19. Системы параллельной обработки: пер. с англ. / под ред. Д. Ивенса. – М.: Мир, 1985. – 416 с.
20. *Levin Ilya I. et al.* Reconfigurable computer systems: from the first FPGAs towards liquid cooling systems // Supercomputing Frontiers and Innovations. – 2016. – Vol. 3 (1). – P. 22-40. – Doi: 10.14529/jsfi160102.
21. *Kalyaev I.A., Levin I.I., Dordopulo A.I., Slasten L.M.* FPGA – based Reconfigurable Computer Systems // Proc. of 2013 Science and Information Conference SAI-2013- Oct 9, 2013, London, UK. – P. 148-155.

REFERENCES

1. *Voevodin V.V., Voevodin Vl.V.* Parallel'nye vychisleniya [Parallel computing]. Saint Petersburg: BKhV-Peterburg, 2002, 608 p. ISBN 5-94157-160-7.
2. *Guzik V.F., Kalyaev I.A., Levin I.I.* Rekonfiguriruemye vychislitel'nye sistemy [Reconfigurable computing systems]. Taganrog: Izd-vo YuFU, 2016, 472 p.
3. *Trimberger S.M*. Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology, *in Proceedings of the IEEE*, March 2015, Vol. 103, No. 3, pp. 318-331. Doi: 10.1109/JPROC.2015.2392104.
4. *Nane R. et al.* A Survey and Evaluation of FPGA High-Level Synthesis Tools, *in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Oct. 2016, Vol. 35, No. 10, pp. 1591-1604. Doi: 10.1109/TCAD.2015.2513673.
5. *Nane R., Sima V.-M., Olivier B., Meeuws R., Yankova Y., Bertels K.* DWARV 2.0: A CoSy-based C-to-VHDL Hardware Compiler, *In FPL*, 2012, pp. 619-622.
6. *Pilato C. and Ferrandi F*. Bambu: A Modular Framework for the High Level Synthesis of Memory-intensive Applications, *In FPL*, 2013, pp. 1-4.
7. *Canis A., Choi J., Aldham M., Zhang V., Kammoona A., Anderson J.H., Brown S., Czajkowski T.* LegUp: High-Level Synthesis for FPGA-based Processor/Accelerator Systems, *In ACM FPGA*, 2011, pp. 33-36.
8. Make Slow Software Run Fast with Vivado HLS. Available at: https://www.xilinx.com/ publications/xcellonline/run-fast-with-Vivado-HLS.pdf.
9. Vitis Unified Software Platform Documentation. Application Acceleration Development. Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ ug1393 -vitis-application-acceleration.pdf (accessed 10 November 2020).
10. *Levin Ilya, Dordopulo Alexey, Gudkov Vyacheslav, Gulenok Andrey, Bovkun Alexander, Yevstafiyev Georgyi, Alekseev Kirill.* Software Development Tools for FPGA-Based Reconfigurable Systems Programming, *In: Communications in Computer and Information Science, Vol. 1129, Chapter Parallel Computing Technologies*, 2019, pp. 1-16.
11. *Levin I.I., Dordopulo A.I.* K voprosu ob avtomaticheskom sozdanii parallel'nykh prikladnykh programm dlya rekonfiguriruemykh vychislitel'nykh sistem [On the issue of automatic creation of parallel application programs for reconfigurable computing systems], *Vychislitel'nye tekhnologii* [Computing technologies], 2020, Vol. 25, No. 1, pp. 66-81.
12. *Morvan A., Derrien S. and Quinton P*. Efficient nested loop pipelining in high level synthesis using polyhedral bubble insertion, *2011 International Conference on Field-Programmable Technology, New Delhi, 2011,* pp. 1-10. Doi: 10.1109/FPT.2011.6132715.
13. *Jensen, Nicklas, Karlsson, Sven*. Improving Loop Dependence Analysis, *ACM Transactions on Architecture and Code Optimization,* 2017, Vol. 14 (3), pp. 1-24. Doi: 10.1145/3095754.

14. *Solihin Yan.* Fundamentals of parallel computer architecture*:* multichip and multicore systems. Chapman and Hall/CRC, 2016. ISBN 978-1-4822-1118-4.
15. *Cooper, Keith D., Torczon, Linda.* Engineering a Compiler. Morgan Kaufmann, 2005. ISBN 1-55860-698-X.
16. *Kennedy Ken, Allen Randy.* Optimizing Compilers for Modern Architectures. A Dependence-based Approach. Morgan Kaufmann, 2001. ISBN 1-55860-286-0.
17. *Muchnick Steven S.* Advanced Compiler Design and Implementation. Morgan Kaufmann, 1997. ISBN 1-55860-320-4.
18. Vektorizatsiya programm [Vectorization of programs], *Vektorizatsiya programm: teoriya, metody, realizatsiya: Sb. perevodov statey* [Vectorization of programs: theory, methods, implementation: Collection of translations of articles]. Moscow: Mir, 1991, pp. 246-267.
19. Sistemy parallel'noy obrabotki [Parallel processing systems]: trans. from engl, ed. by D. Ivensa. Moscow: Mir, 1985, 416 p.
20. *Levin Ilya I. et al.* Reconfigurable computer systems: from the first FPGAs towards liquid cooling systems, *Supercomputing Frontiers and Innovations,* 2016, Vol. 3 (1), pp. 22-40. Doi: 10.14529/jsfi160102.
21. *Kalyaev I.A., Levin I.I., Dordopulo A.I., Slasten L.M.* FPGA – based Reconfigurable Computer Systems, *Proc. of 2013 Science and Information Conference SAI-2013- Oct 9, 2013, London, UK,* pp. 148-155.

Статью рекомендовал к опубликованию д.т.н. Э.В. Мельник.

Дордопуло Алексей Игоревич – Общество с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров»; e-mail: dordopulo@superevm.ru; 347935, г. Таганрог, 9-й переулок, д. 44; тел.: 88634477407; начальник отдела математико-алгоритмического обеспечения; к.т.н.

Гуленок Андрей Александрович – e-mail: andrei_gulenok@mail.ru; 347910, г. Таганрог, ул. 1-я Котельная, 71, кв. 301; тел.: +79085083496; начальник сектора; к.т.н.

Бовкун Александр Викторович – e-mail: simans2002@mail.ru; 347917, г. Таганрог, ул. Северная, 67; тел.: 88634477407; н.с., к.т.н.

Левин Илья Израилевич – Южный федеральный университет; e-mail: iilevin@sfedu.ru; 347928, г. Таганрог, ул. Петровская, 15, кв. 143; тел.: 88634612111; и.о. зав. кафедрой интеллектуальных и многопроцессорных систем; д.т.н.; профессор.

Гудков Вячеслав Александрович – e-mail: Vgudkov@sfedu.ru; 347905, г. Таганрог, ул. Дзержинского, 110; тел.: 88634477407; к.т.н.

Дудко Сергей Анатольевич – e-mail: dudko@sfedu.ru; 347900, г. Таганрог, пер. Тургеневский, 44; тел.: +79034318173; кафедра интеллектуальных и многопроцессорных систем; аспирант.

Dordopulo Alexey Igorevich – “Supercomputers and Neurocomputers Research Center” Co. Ltd.; e-mail: dordopulo@superevm.ru; 44, 9th lane, Taganrog, 347935, Russia; phone: +78634477407; head of the division of mathematic and algorithmic support; cand. of eng. sc.

Gulenok Andrey Aleksandrovich – e-mail: andrei_gulenok@mail.ru; 1st Kotelnaya 71, app. 301, Taganrog, 347910, Russia; phone +79085083496; head of sector; cand. of eng. sc.

Bovkun Aleksandr Victorovich – e-mail: simans2002@mail.ru; Severnaya 67, Taganrog, 347917, Russia; phone +78634477407; researcher; cand. of eng. sc.

Levin Ilya Izrailevich – Southern Federal University; e-mail: iilevin@sfedu.ru; 15, Petrovskaya str., ap. 143, Taganrog, 347928, Russia; phone: +78634612111; head of the department of intellectual and multiprocessor systems;dr. of eng. sc.; professor.

Gudkov Vyacheslav Aleksandrovich – e-mail: Vgudkov@sfedu.ru; Dzerzhinskogo str., ap. 110, Taganrog, 347905, Russia; phone: +78634477407; cand. of eng. sc.

Dudko Sergei Anatolievich – e-mail: dudko@sfedu.ru; 44, Turgenevskii lane, Taganrog, 347900, Russia; phone: +79034318173; the department of intellectual and multiprocessor systems; graduate student.