

Программные средства высокоуровневого синтеза для многокристальных реконфигурируемых вычислительных систем

© 2022 А.И. Дордопуло¹, И.И. Левин^{1,2}, В.А. Гудков^{1,2}, А.А. Гуленок¹

¹Общество с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров»
(347922 Таганрог, пер. Итальянский, д. 106)

²Федеральное государственное автономное образовательное учреждение высшего
образования Южный федеральный университет
(347928 Таганрог, пер. Некрасовский, д. 44),

E-mail: dordopulo@superevm.ru, ilevin@sfsu.ru, gudkov@superevm.ru, gulenok@superevm.ru

Поступила в редакцию: ДД.ММ.ГГГГ

В статье описывается оригинальный комплекс высокоуровневого синтеза, преобразующий последовательные программы в схемотехническую конфигурацию специализированных аппаратных средств для реконфигурируемых вычислительных систем. Из исходной последовательной программы строится абсолютно-параллельная форма – информационный граф. Далее, граф преобразуется в ресурснезависимую параллельно-конвейерную форму – кадровую структуру, которую можно адаптировать к различному аппаратному ресурсу. Преобразование кадровой структуры в информационно-эквивалентную, но занимающую меньший аппаратный ресурс, структуру выполняется с помощью формализованных методов редукции производительности, что позволяет автоматически получить рациональное решение для заданной многокристальной реконфигурируемой вычислительной системы. В отличие от известных средств высокоуровневого синтеза результатом преобразования является не IP-ядро вычислительно-трудоемкого фрагмента, а автоматически синхронизированное решение прикладной задачи для всех кристаллов ПЛИС реконфигурируемой вычислительной системы. По сравнению с распараллеливающими компиляторами, число анализируемых вариантов синтеза рационального решения существенно меньше, что является отличительной особенностью описываемого комплекса. Применение программных средств высокоуровневого синтеза рассматривается на примере задачи решения системы линейных алгебраических уравнений методом Гаусса, содержащей информационно-взаимозависимые вычислительные фрагменты с существенно разной степенью параллелизма.

Ключевые слова: высокоуровневый синтез, трансляция программ, язык C, редукция производительности, реконфигурируемые вычислительные системы, программирование многопроцессорных вычислительных систем.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

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. X, № Y. С. Z1–Z2. DOI: 10.14529/cmse220Y0Z.

Введение

Основной прагматичной целью высокопроизводительных вычислений является сокращение времени решения задачи. Существенное сокращение возможно путем повышения быстродействия элементной базы многопроцессорной вычислительной системы (МВС), применения специальных форм организации вычислений или максимального распараллеливания операций задачи [VoevodinV02]. Возможности микроэлектроники по минимизации размеров транзистора практически достигли предела, поэтому перспективы дальнейшего роста производительности с помощью повышения скорости работы ядер и увеличения их

числа существенно ограничены. Распараллеливание вычислительных операций на десятки миллионов вычислительных узлов современных суперкомпьютеров, таких как Summit, Fugaku, TaihuLight [AntonovAV21], приводит к обратному эффекту, если информационные зависимости в структуре решаемой задачи не учитываются: при увеличении числа узлов время решения задачи не сокращается, а увеличивается. Поэтому применение специальных форм организации вычислений с учетом информационных зависимостей между вычислительными фрагментами является наиболее перспективным направлением поиска способов сокращения времени решения задачи.

Учет информационных зависимостей – это одно из положений концепции структурно-процедурной организации вычислений [Guzik16] для реконфигурируемых вычислительных систем (РВС) с программируемыми логическими интегральными схемами (ПЛИС). РВС, содержащие множество объединенных коммутационной системой кристаллов ПЛИС [LevinDFK16], значительно превосходят МВС кластерной архитектуры по энергоэффективности и реальной производительности, но сильно уступают в удобстве программирования и отладки. Одним из многообещающих направлений упрощения программирования РВС является поиск новых решений в области средств трансляции последовательных программ в конфигурационные файлы ПЛИС с помощью программных средств высокоуровневого синтеза [Nane16].

В настоящей работе рассматриваются методы преобразования входной программы на языке C в кадровую структуру и методы автоматического масштабирования полученных решений для доступного аппаратного ресурса заданной РВС с помощью механизмов редукции производительности. Во втором разделе рассматриваются известные средства высокоуровневого синтеза и особенности их работы, формулируются ключевые отличия описываемого комплекса от существующих. В третьем разделе приведены теоретические основы преобразований, используемых программными средствами высокоуровневого синтеза для автоматической адаптации прикладной задачи к доступному аппаратному ресурсу РВС. Четвертый раздел описывает основные этапы выполняемых программными средствами преобразований. В пятом разделе представлены результаты, полученные при трансляции задачи решения системы линейных алгебраических уравнений (СЛАУ) методом Гаусса. В заключении обобщаются предложенные принципы и методы.

1. Обзор существующих средств высокоуровневого синтеза

Для программирования ускорителей на ПЛИС, наряду с традиционными системами схемотехнического проектирования, такими как Xilinx Vivado или Intel Quartus Prime, все чаще используются средства высокоуровневого синтеза (High Level Synthesis, HLS). HLS-компиляторы – это трансляторы [Nane16, NumanPPF20] с некоторого языка программирования в конфигурационные файлы ПЛИС, преобразующие программы в схемотехническую конфигурацию специализированных аппаратных средств на языках HDL-группы. В зависимости от языка входной программы средства HLS [Nane16] можно отнести к одной из двух категорий (рис. 1): трансляторы проблемно-ориентированных языков, т.е. адаптированных к определенной проблемной области версий языков программирования, и трансляторы языков общего назначения, т.е. диалектов языков с некоторыми особенностями и ограничениями. Из рис. 1 видно, что наибольшее число HLS-компиляторов относится к группе процедурных языков общего назначения на основе диалектов языка программирования C. В этой группе представлены как академические (DWARV [NaneSOMYB12],

BAMBU [PilatoF13] и LEGUP [CanisCAZKABC11]), так и активно развивающиеся коммерческие (CatapultC, Vivado HLS [Make21], Vivado Vitis [Vitis21]) HLS-компиляторы, сравнительный анализ возможностей которых представлен в [Nane16, NumanPPF20].

Vivado HLS [10] – это САПР Xilinx, предназначенная для создания цифровых устройств с применением языков высокого уровня. Vivado HLS содержит ряд средств оптимизации, которые характерны как для компиляторов, так и для систем разработки цифровых схем [Tarasov13]:

- конвейеризацию операций, которая планирует последовательность выполнения операций в пределах заданного числа тактов;
- анализ и оптимизацию разрядности для сокращения количества битов, передаваемых в информационных каналах;
- выделение внутренней памяти BRAM для быстрого доступа к памяти при низких аппаратных затратах;
- оптимизацию (конвейеризацию) циклов, которая использует параллелизм на уровне цикла для запуска очередной итерации цикла до завершения предшествующей с учетом информационной зависимости;
- пространственное распараллеливание для одновременной аппаратной реализации информационно-независимых команд;
- оптимизацию выполняемых операций, заменяющую выполняемую операцию информационно-эквивалентной, но требующей меньше затрат или ресурсов ПЛИС;
- прогнозирование и перемещение кода для его выноса из условных и управляющих конструкций;
- преобразования условий для планирования параллельных команд из непересекающихся путей выполнения.

Xilinx позиционирует Vivado HLS как инструмент разработки проектов, где важна скорость разработки, а не эффективность кода и/или сокращение аппаратных затрат при реализации в ПЛИС.

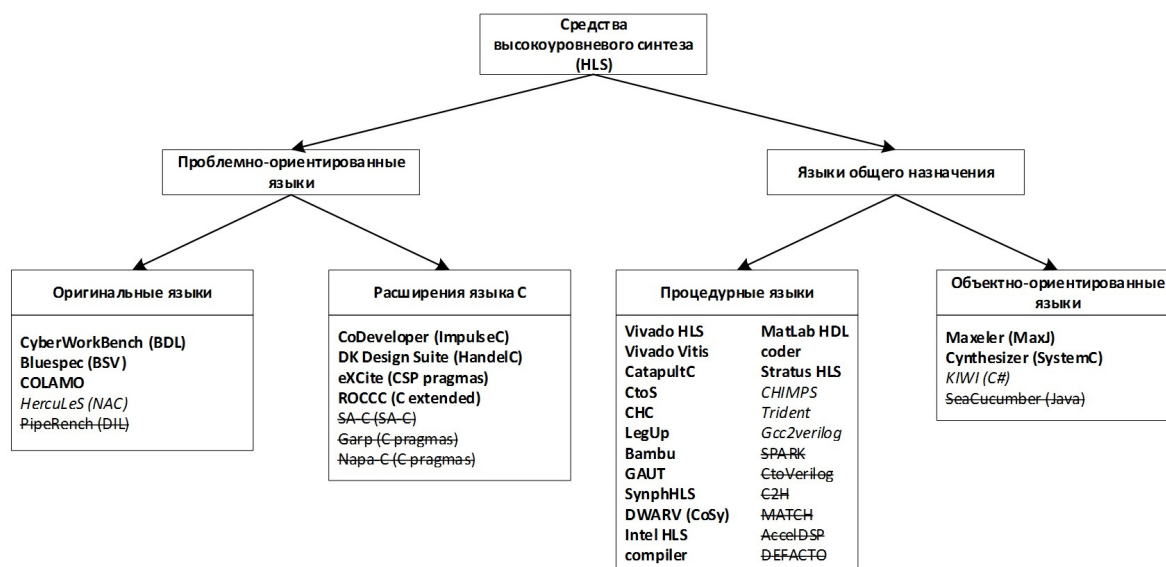


Рис. 1. Классификация средств высокоуровневого синтеза (используется, недоступен, завершен)

Xilinx Vitis [12] – это новая среда разработки, объединяющая графические инструменты, компиляторы, анализаторы и отладчики для ускорения фрагментов кода последовательных программ. Vitis содержит различные пакеты для решения отдельных задач, таких как Vitis Core для разработки ускорительных ядер широкого круга задач и Vitis AI для ускорения работы систем искусственного интеллекта на базе нейронных сетей. Vitis ориентирован на работу со встроенными платформами Xilinx – ускорителями на базе ПЛИС для серверных и облачных приложений и/или ускорителями Alveo для встраиваемых устройств. Целевая платформа Vitis определяет базовую аппаратную и программную архитектуру и контекст приложения для платформ Xilinx, включая интерфейсы внешней памяти, пользовательские интерфейсы ввода/вывода и библиотеки времени исполнения.

Несмотря на обилие реализованных преобразований и методов оптимизации, применение Vivado HLS и Vitis не гарантирует автоматического роста производительности, т.к. далеко не каждая программа на языке C может быть эффективно реализована в ПЛИС [Tarasov13]. Как и Vivado HLS/Vitis, подавляющее большинство приведенных в [Nane16, NumanPPF20] HLS-компиляторов, анализируют вычислительно трудоемкий фрагмент программы на языке C и преобразуют его в специализированный вычислитель (IP-ядро), синтезируемый на основе автоматной модели или процессорной парадигмы. Несмотря на существенный выигрыш полученного IP-ядра в скорости вычислений по сравнению с процессорной реализацией [Nane16], масштабирование решения даже в пределах одного кристалла и организация потоков данных возлагаются на пользователя. Поэтому, несмотря на автоматизацию трансляции фрагмента последовательной программы, поиск рационального решения всей задачи для доступного аппаратного ресурса является целиком и полностью задачей программиста. При использовании нескольких ПЛИС, связанных пространственной коммутационной системой, или многокристальных PBC [Guzik16, LevinDFK16] эта задача многократно усложняется и становится сродни масштабированию решений распараллеливающим компилятором. Распараллеливающий компилятор из последовательной программы восстанавливает параллелизм исходного алгоритма, выявляет участки и фрагменты, которые можно выполнить одновременно (например, итерации циклов), и добавляет инструкции для их параллельного исполнения на узлах МВС. Вычислительная сложность автоматического распараллеливания для МВС с распределенной памятью состоит в том, что для каждого варианта распараллеливания фрагментов последовательной программы компилятору нужно оценивать и ранжировать варианты размещения данных по процессорам с учетом характеристик коммуникационной сети, что многократно расширяет пространство перебора. Неоднократные попытки различных исследователей [Kolganov20] создать стабильно развивающийся автоматизированный распараллеливающий компилятор для современных кластерных МВС с распределенной памятью показали [Kolganov20], что для реальных, а не тестовых, задач это возможно только в крайне редких случаях из-за чрезвычайно большого числа вариантов и отсутствия удобного критерия для оценки эффективности синтезируемых версий параллельной программы.

В отличие от рассмотренных HLS-средств и распараллеливающих компиляторов авторами разработан оригинальный метод адаптации вычислений прикладной задачи к доступному вычислительному ресурсу многокристальных PBC. Согласно классификации (рис. 1), разработанный комплекс занимает промежуточное положение: исходной формой представления задачи является процедурный язык общего назначения C, который транслируется в программу на проблемно-ориентированном языке высокого уровня COLAMO [Guzik16], а

далее – в конфигурационные файлы ПЛИС. При схожей функциональности от наиболее близких коммерческих аналогов, таких как Xilinx Vivado HLS и Vitis, комплекс отличается автоматическим преобразованием входной программы без указаний пользователя в виде директив `#pragma` или других способов ручной разметки кода, поддержкой многокристальных решений и автоматической синхронизацией информационных и управляющих сигналов.

2. Теоретические основы адаптации вычислений прикладной задачи к доступному вычислительному ресурсу реконфигурируемых вычислительных систем

В отличие от рассмотренных HLS-компиляторов, последовательная программа в комплексе представляется в абсолютно параллельной форме в виде информационного графа [LevinDFK16] – ориентированного графа, вершины которого соответствуют операциям над данными, а дуги отражают информационную зависимость между ними. Каждая операция над элементами данных представляется операционной вершиной, поэтому общее число операционных вершин соответствует числу операций над данными задачи, а число входных вершин соответствует размерности всех обрабатываемых данных. Операционные вершины информационного графа задачи (ИГЗ) распределены по слоям и итерациям: слои, как и ярусы графа алгоритма [1], содержат информационно-независимые вершины, поэтому связи между вершинами в слое отсутствуют, а итерации описывают информационную зависимость вершин разных слоев между собой. ИГЗ является ациклическим, т.е. не содержит обратных связей. Операционные вершины в слоях и итерациях ИГЗ обычно представляют подграфы (например, соответствующие телу цикла в последовательных программах) из нескольких простых арифметических операций, связанными информационными зависимостями, среди которых могут быть условные или коммутационные операции. Информационный граф может быть достаточно просто построен из последовательной программы с помощью развертки циклов: на рис. 2 представлен фрагмент последовательной программы реализации прямого хода решения системы линейных алгебраических уравнений (СЛАУ) методом Гаусса на языке C.

```
for (i = 0; i < N; i++) {
    for (j = i+1 ; j < N ; j++) {
        d = (m[j][i]/m[i][i]);
        for (k = i; k < N+1 ; k++)
            m[j][k] = m[j][k] - (m[i][k]*d);
    }
}
```

Рис. 2. Фрагмент программной реализации прямого хода решения СЛАУ методом Гаусса на языке C

В слоях информационного графа решения СЛАУ методом Гаусса (рис. 3) расположены информационно-независимые подграфы g_{jk}^i , каждый из которых содержит 3 операционные вершины: деление, умножение и вычитание, что соответствует операциям в теле цикла по переменным j и k (рис. 2). Информационная зависимость между итерациями задается связями между подграфами: выход $m^1[1,1]$ подграфа $g_{1,1}^0$ нулевой итерации является входом подграфов $g_{2..N,1..N}^1$ первой, выход $m^2[3,2]$ подграфа $g_{3,2}^1$ является входом $g_{3..N,2..N}^2$ второй и

т.д.). Согласно коду программы (рис. 2) каждый следующий слой ИГЗ содержит меньшее число подграфов g_{jk}^i .

Наиболее важным свойством ИГЗ является инвариантность его формы для различных способов описания прикладной задачи. ИГЗ – это абсолютно-параллельная форма представления вычислений, отражающая информационные зависимости между функционально-завершенными фрагментами программы, поэтому, с точностью до топологической перестановки подграфов в слое, ИГЗ будет неизменен для разных последовательных программ, описывающих одну и ту же прикладную задачу, при одинаковом базисе используемых вычислительных операций. Так, при изменении кода программы (рис. 2), например, при разбиении внешнего цикла на два, ИГЗ не изменится, а для распараллеливающего компилятора измененная программа не будет тождественна исходной, поэтому при автоматическом распараллеливании этих семантически одинаковых программ будут получены разные варианты распараллеливания.

Инвариантность представления параллельных вычислений ИГЗ позволяет объединять топологически различные, но информационно-эквивалентные, варианты описания вычислений, что существенно сокращает число потенциальных анализируемых вариантов организации параллельных вычислений уже на уровне информационной модели.

Аппаратная реализация ИГЗ в виде вычислительной структуры имеет минимальную латентность, наименьшее время решения и максимальную производительность, но требует множество вычислительных устройств и каналов памяти для одновременной подачи всех заданных параметрами задачи входных данных и выполнения всех операций, что обычно недостижимо для реальных вычислительных систем. Поэтому для реализации в вычислительной системе ИГЗ преобразуется в кадровую структуру [Guzik16], которая учитывает информационные зависимости между его фрагментами и основные параметры вычислительных устройств. Переход от информационного графа к кадровой структуре выполняется заменой операционных вершин графа на соответствующие операциям вычислительные устройства, а дуг – на связи коммутационной системы. Полученная в результате абсолютно параллельная кадровая структура (АПКС) отличается от ИГЗ учетом характеристик реализации на заданной вычислительной системе (частоты работы и быстродействия устройств, интервала обработки данных и др.). Например, ИГЗ прямого хода решения СЛАУ методом Гаусса (рис. 3) будет соответствовать абсолютно параллельной кадровой структуре при замене операций подграфов вычислительными устройствами при неизменной общей вычислительной структуре.

АПКС может быть преобразована в другие кадровые структуры, более экономичные по занимаемому аппаратному ресурсу и обеспечивающие информационную эквивалентность результатов вычислений. Если сократить ресурс, занимаемый АПКС (рис. 3), путем упорядочивания подграфов по слоям, получится приведенная на рис. 4-а структура, содержащая все итерации исходной АПКС с одним структурно-реализованным подграфом в каждом слое. При дальнейшем сокращении занимаемого аппаратного ресурса будет получена минимальная кадровая структура прямого хода решения СЛАУ методом Гаусса (рис. 4-б), которая содержит только базовый подграф, изоморфный остальным подграфам задачи, расположенным в слоях и итерациях. Каждая кадровая структура (рис. 3, 4) обладает различными характеристиками быстродействия и занимаемого ресурса, поэтому будем считать, что преобразование кадровых структур параметризуется аппаратным ресурсом, влияющим на время решения задачи.

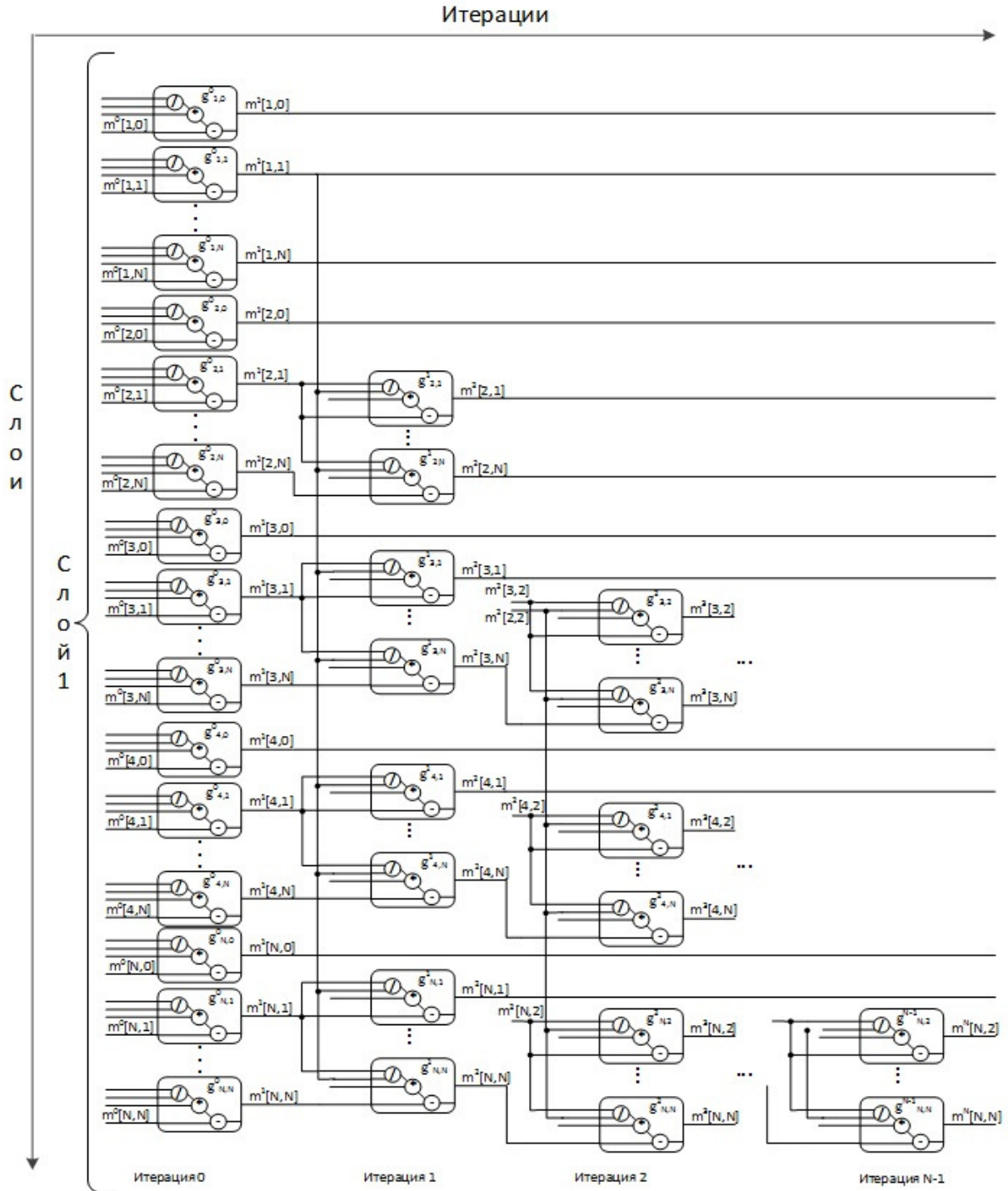


Рис. 3. Информационный граф прямого хода решения СЛАУ методом Гаусса

Параметризуемое аппаратным ресурсом преобразование абсолютно параллельной кадровой структуры существенно отличается не только от методов распараллеливания, но и от применявшейся для РВС технологии индуктивных программ [LevinDFK16], также зависевшей от доступного ресурса.

В зависимости от доступного вычислительного ресурса, информационно-эквивалентные преобразования кадровых структур можно представить движением в трехмерном пространстве, заданном осями основных характеристик кадровой структуры: «Число слоев», «Число

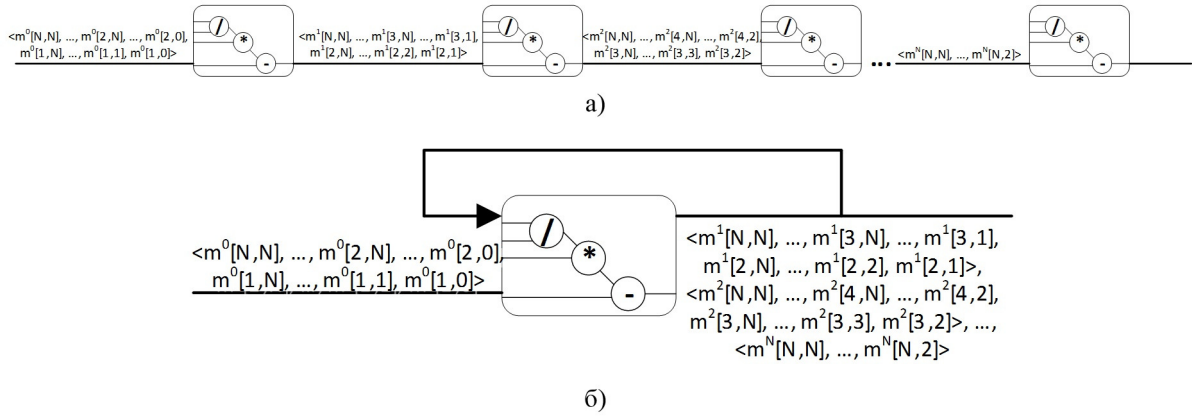


Рис. 4. Кадровые структуры разных вариантов реализации решения СЛАУ методом Гаусса: а) кадровая структура, полученная методом распараллеливания по итерациям; б) минимальная кадровая структура

итераций», «Команды», «Разрядность», «Время» и «Интервал» (рис. 5). Исходной точкой является абсолютно параллельная кадровая структура с координатами $(L_{max}, It_{max}, T_{min})$. Целью движения является достижение области доступного аппаратного ресурса, которая задана числом каналов распределенной памяти по слоям и аппаратным ресурсом по итерациям. На каждом отрезке движения, показанном на рис. 5 стрелками, сокращаются степень параллелизма, производительность и аппаратные затраты абсолютно параллельной кадровой структуры до информационно-эквивалентной структуры, которая находится в области доступного ресурса.

Начальным направлением движения является сокращение числа каналов как типичного критического ресурса для большинства задач различных предметных областей, которое выполняется редукцией числа информационно-независимых слоевых подграфов ИГЗ, аппаратно реализуемых в кадровой структуре. Движение вдоль оси «Число слоев» завершается в некоторой точке с координатами (L_1, It_{max}, I_{min}) , где L_1 соответствует числу доступных каналов либо предельному значению $L_1 = 1$. Дальнейшее движение происходит по оси «Число итераций» для сокращения числа информационно-зависимых подграфов (итераций) и завершается в некоторой точке (L_1, It_1, I_1) , которая соответствует аппаратному ресурсу для размещения итерационных ступеней базовых подграфов g . Значение It_1 соответствует доступному ресурсу либо предельному значению $It_1 = 1$, а изменение I_{min} на I_1 отражает возможный при этом преобразовании рост интервала. Если область доступного аппаратного ресурса достигнута, кадровую структуру можно реализовать в архитектуре реконфигурируемой вычислительной системы. Для улучшения характеристик реализации и поиска наиболее рациональной кадровой структуры можно использовать оптимизационные преобразования, минимизирующие интервал обработки данных движением по оси «Интервал I».

Если даже при минимальных значениях слоев и итераций аппаратного ресурса вычислительной системы недостаточно для структурной реализации минимальной кадровой структуры из одного базового подграфа ИГЗ, выполняется переход в нижний октант пространства кадровых структур. Дальнейшее сокращение параллелизма кадровой структуры происходит с помощью редукции числа одновременно работающих устройств (команд) и разрядности одновременно обрабатываемых данных.

движении в пространстве возможных реализаций: число слоев, числу итераций, числу команд, разрядности и интервалу обработки данных. Общее число этапов масштабирования для синтеза рациональной кадровой структуры определяется суммой числа этапов для достижения доступного ресурса РВС и числа оптимизационных преобразований. Согласно учитываемым характеристикам, общее число выполняемых преобразований не превысит шести, разумеется, при рациональном и корректном определении коэффициента редукции и нового значения сокращаемой характеристики кадровой структуры. Полученная оценка значительно меньше числа вариантов параллельной программы, анализируемых распараллеливающим компилятором, и инвариантна для разных версий исходной программы и ИГЗ, так и для разных прикладных задач за счет неизменного числа характеристик кадровой структуры. Это позволяет существенно сократить время портации АПКС при изменении архитектуры целевой вычислительной системы.

3. Компоненты комплекса средств высокоуровневого синтеза

Описанные принципы преобразования кадровой структуры прикладной задачи реализованы в комплексе средств высокоуровневого синтеза [LevinDGGBYA19] для многокристальных реконфигурируемых вычислительных систем. Входная последовательная программа на языке C в стандарте ISO/IEC 9899:1999 преобразуется компонентами комплекса в ресурсонезависимую форму и масштабируется для доступного аппаратного ресурса РВС, а результатом трансляции является программа на языке высокого уровня COLAMO. Сокращение слоев, итераций, числа устройств, разрядности и интервала обработки данных при движении в пространстве возможных реализаций кадровых структур выполняется методами редукции производительности и аппаратных затрат, описанными в [DordopuloL20]. Все преобразования кадровой структуры выполняются следующими компонентами комплекса (рис. 6):

- транслятором «Ангел», преобразующим входную программу на языке C в информационный граф и абсолютно-параллельную кадровую структуру;
- процессором «Русалка», преобразующим АПКС в ресурсонезависимую параллельно-конвейерную форму, параметризуемую аппаратным ресурсом;
- выделение внутренней памяти BRAM для быстрого доступа к памяти при низких аппаратных затратах;
- процессором «Прокруст», выполняющим с помощью редукции производительности и аппаратных затрат движение в верхнем октанте и расчет параметров кадровой структуры для рациональной ее реализации в архитектуре заданной РВС;
- процессором «Щелкунчик», выполняющим редукцию производительности в нижнем октанте при нехватке аппаратного ресурса для структурной реализации базового подграфа ИГЗ.

Трансляция программы на языке COLAMO в конфигурационные файлы ПЛИС многокристальных РВС осуществляется разработанными ранее транслятором языка программирования COLAMO [Guzik16] и синтезатором многокристальных решений Fire!Constructor [Guzik16]. Загрузочные конфигурационные файлы (*.bit) синтезируются для каждого кристалла синтезатором системы автоматизированного проектирования Xilinx Vivado.

Методы преобразования последовательной программы с произвольным обращением к памяти в информационный граф задачи и абсолютно-параллельную кадровую структуру,

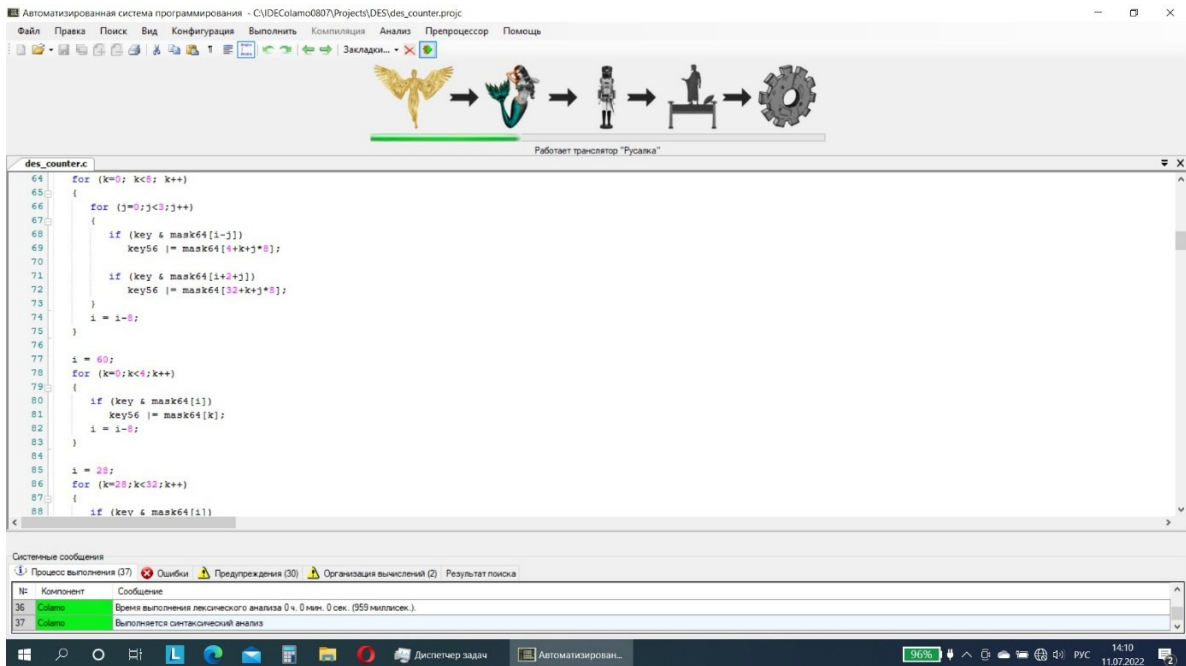


Рис. 6. Преобразование кадровой структуры компонентами комплекса средств высокоуровневого синтеза

выполняемое транслятором «Ангел», подробно рассмотрены в [LevinDG20]. Анализ структуры АПКС, выделение подзадач, определение числа слоев и итераций для каждого фрагмента, анализ информационных зависимостей в структуре каждой подзадачи и между ними, расщепление скалярных переменных и растягивание массивов по итерациям для устранения нарушений правил однократного присваивания и единственной подстановки рассмотрены в [LevinDG19]. Преобразование АПКС в масштабируемую параллельно-конвейерную форму выполняется процессором «Русалка» [LevinDG19], а расчет параметров параллелизма задачи для доступного аппаратного ресурса – процессором «Прокруст». Если задача содержит несколько вычислительных фрагментов или подзадач с разной степенью параллелизма, необходимо сократить производительность всех подзадач в одинаковое число раз, заданное коэффициентом редукции производительности. При этом необходимо найти сбалансированное по интенсивности потоков данных решение для фрагментов задачи с разной степенью параллелизма, учитывая их информационные взаимозависимости и рациональную реализацию этого решения на доступном аппаратном ресурсе. Реализованная в процессоре «Прокруст» методика расчета параметров и преобразования кадровой структуры применима в том числе для задач, содержащих несколько вычислительных фрагментов или подзадач с разной степенью параллелизма. Для таких задач считается, что наиболее трудоемкий фрагмент вносит наибольший вклад в общее время решения задачи, поэтому именно этот фрагмент должен быть реализован наиболее эффективно, в наилучшем случае – в виде (мульти)конвейерной структуры с минимальным интервалом обработки данных. Для этого используется предложенное в [LevinDG20] деление фрагментов задачи по величине занимаемого в АПКС аппаратного ресурса на «флагманские» фрагменты, занимающие наибольший ресурс, и занимающие существенно (как минимум, на один десятичный порядок) меньший ресурс «катера». Решение СЛАУ методом Гаусса, в котором можно выделить прямой и обратный ход, является одним из наиболее наглядных примеров такого разделения

подзадач. Прямой ход по числу итераций циклов является «флагманом» с трудоемкостью $O(N^3)$ для исходной матрицы размерности N , а обратный ход представляет собой «катер» с трудоемкостью не более $O(N^2)$. Алгоритмы работы процессора «Прокруст» обеспечивают сбалансированное преобразование таких задач, содержащих фрагменты с разной степенью параллелизма.

4. Результаты экспериментальных исследований разработанного комплекса

С помощью комплекса средств высокоуровневого синтеза для многокристалльных РВС реализован ряд прикладных программ линейной алгебры: решение систем линейных алгебраических уравнений методом Гаусса, методом Якоби для 3-диагональных матриц, методом Гаусса-Зейделя и разложением на верхнюю треугольную и нижнюю треугольную матрицы (LU-разложение). В таблице 1 представлены результаты экспериментальной проверки времени трансляции задачи решения СЛАУ методом Гаусса комплексом средств высокоуровневого синтеза. Размер обрабатываемых матриц составлял 8000×8001 элементов. Задача, содержащая прямой и обратный ход алгоритма, была реализована на трех РВС: «Тайгета» [LevinDFK16], «Терциус» и «Терциус-2». Для каждой из аппаратных платформ измерялись два показателя: время портации (преобразования) кадровой структуры к архитектуре РВС и время синтеза решения. Время портации задачи на конфигурацию РВС прикладными программистами было принято равным одному 8-часовому рабочему дню, выраженному в секундах. Время синтеза решения комплексом и прикладными программистами определялось как сумма времени портации и времени синтеза загрузочного конфигурационного файла ПЛИС, которое зависит от логической емкости ПЛИС и уровня заполнения кристалла. Заполнение кристалла рассматривалось на уровне 90%, а время синтеза загрузочного конфигурационного файла составляет 6-8 часов для РВС «Тайгета», 8 часов для РВС «Терциус» и «Терциус-2».

Коэффициент сокращения временных затрат (по времени портации и времени синтеза) определялся как отношение соответствующей временной характеристики, полученной прикладными программистами, ко времени работы средств высокоуровневого синтеза. Достигнутый комплексом уровень реальной производительности определялся как отношение числа подграфов в решении, полученном разработанными средствами, к числу подграфов, полученных прикладными программистами. Для каждого полученного решения проверялась его работоспособность и корректность получаемых результатов запуском на соответствующей аппаратной платформе, а его реальная производительность сопоставлялась с результатами, полученными при портации ресурсонезависимой параллельной программы прикладными программистами для определения эффективности и достижения заданного уровня реальной производительности, представленных в таблице 2.

Прямой ход, являющийся «флагманом», с помощью распараллеливания по итерациям реализован структурно в виде итерационных ступеней обработки, связанных информационной зависимостью. Удельная производительность полученного средствами высокоуровневого синтеза варианта задачи решения СЛАУ методом Гаусса составила 0,83-0,94 от созданной прикладными программистами на языке COLAMO программы. Время трансляции задачи указанной размерности всеми компонентами комплекса составило 22 мин. 45 сек. на персональном компьютере с процессором Intel i5 7300 и 8 Гб оперативной памяти.

Таблица 1. Результаты экспериментальной проверки времени портации задачи решения СЛАУ методом Гаусса

Аппаратные платформы	РВС «Тайгета»	РВС «Терциус»	РВС «Терциус-2»
Время портации на конфигурацию РВС комплексом, с	2111,3	2748,2	2751
Время портации на конфигурацию РВС прикладными программистами, с	57600	57600	57600
Время синтеза решения комплексом, с	23711,3	31548,2	31548,2
Время синтеза решения прикладными программистами, с	79200	86400	86400
Коэффициент сокращения временных затрат по времени портации	27,3	21	21
Коэффициент сокращения временных затрат по времени синтеза	3,3	2,7	2,7

Таблица 2. Результаты экспериментальной проверки уровня производительности при портации задачи решения СЛАУ методом Гаусса

Аппаратные платформы	РВС «Тайгета»	РВС «Терциус»	РВС «Терциус-2»
Число подграфов в решении, полученном комплексом	784	1178	1178
Число подграфов в решении, полученном прикладными программистами	830	1286	1286
Уровень реальной производительности, достигнутый комплексом	0,94	0,91	0,91
Заданный уровень реальной производительности,	0,7		

При трансляции этой же задачи в Vivado HLS полученное решение содержит 1 итерационную ступень, с помощью ручной разметки кода директивами `#pragma` удалось получить решение для 2 итерационных ступеней алгоритма прямого хода. Сравнение результатов времени портации и числа подграфов в решении, полученных комплексом, Ю прикладными программистами и Vivado HLS позволяет сделать вывод о применимости предложенных теоретических основ для адаптации задачи решения СЛАУ методом Гаусса к различным РВС и более высокой эффективности для этой задачи по сравнению с HLS-компилятором Vivado HLS.

Заключение

С помощью представленной оригинальной методики преобразования последовательных программ комплекс средств высокоуровневого синтеза позволил автоматически получить конфигурационные файлы ПЛИС задачи решения СЛАУ методом Гаусса для многокристальных реконфигурируемых вычислительных систем. Применение методов редукции производительности и аппаратных затрат в новой версии комплекса для адаптации параллельных вычислений позволяет значительно сократить число анализируемых вариантов организации вычислений при синтезе вычислительной структуры для РВС. В отличие от известных HLS-компиляторов, входная программа на языке С преобразуется автоматически, без ручной разметки кода или иных указаний пользователя. Комплекс поддерживает синтез многокристальных решений с автоматической синхронизацией информационных и управляющих сигналов. Применение предложенного подхода к преобразованию кадровой структуры задачи при высокоуровневом синтезе позволяет находить рациональное решение (с эффективностью не ниже 50% от результатов, полученных инженерами-схемотехниками) за существенно меньшее (по сравнению с распараллеливающими компиляторами) число преобразований и повысить скорость разработки решения прикладной задачи.

Список литературы

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. БХВ-Петербург, 2002. 608 с.
2. Антонов А.С., Афанасьев И.В., Воеводин Вл.В. Высокопроизводительные вычислительные платформы: текущий статус и тенденции развития // Вычислительные методы и программирование. 2021. Т. 22. Вып. 2. С. 135–177. DOI: 10.26089/NumMet.v22r210.
3. Гузик В.Ф., Каляев И.А., Левин И.И. Реконфигурируемые вычислительные системы. Таганрог: Изд-во ЮФУ, 2016. 472 с.
4. Levin I., Dordopulo A., Fedorov A., Kalyaev I. Reconfigurable computer systems: from the first FPGAs towards liquid cooling systems. Supercomputing Frontiers and Innovations. 2016. Vol. 3, no. 1. P. 22–40. DOI: 10.14529/jsfi160102.
5. Nane R. 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.
6. 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.
7. Nane R., Sima V.-M., Olivier B., Meeuws R., Yankova Y., Bertels K. DWARV 2.0: A CoSy-based C-to-VHDL Hardware Compiler. FPL. 2012. P. 619–622. DOI: 10.1109/FPL.2012.6339221.
8. 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.

9. 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. ACM FPGA. 2011. P. 33–36. DOI:10.1145/1950413.1950423.
10. Make Slow Software Run Fast with Vivado HLS. URL: <https://www.xilinx.com/publications/xcellonline/run-fast-with-Vivado-HLS.pdf> (дата обращения: 10.3.2021).
11. 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: 10.3.2021).
12. Тарасов И. Проектирование для ПЛИС Xilinx с применением языков высокого уровня в среде Vivado HLS // Компоненты и технологии. 2013. № 12. С. 40–48.
13. Kolganov A.S. An experience of applying the parallelization regions for the step-by-step parallelization of software packages using the SAPFOR system. Numerical methods and programming. 2020. Vol. 21, no. 66. P. 388–404. DOI: 10.26089/NumMet.v21r432.
14. Levin I., Dordopulo A., Gudkov V., Gulenok A., Bovkun A., Yevstafiyev G., Alekseev K. Software Development Tools for FPGA-Based Reconfigurable Systems Programming // Russian Supercomputing Days. Vol. 1129 / ed. by Vl. Voevodin, S. Sobolev. Cham: Springer, 2019. P. 625–640. Communications in Computer and Information Science. DOI: 10.1007/978-3-030-36592-9_51.
15. Dordopulo A.I., Levin, I.I. Performance Reduction For Automatic Development of Parallel Applications For Reconfigurable Computer Systems. Supercomputing Frontiers and Innovations. 2020. Vol. 7, no. 2. P. 4–23. DOI: 10.14529/jsfi200201.
16. Левин И. И., Дордопуло А. И., Гудков В. А. [и др.] Комплекс средств программирования реконфигурируемых вычислительных систем на основе ПЛИС // Параллельные вычислительные технологии (ПаВТ'2020) : Короткие статьи и описания плакатов, Пермь, 31 марта – 02 апреля 2020 года. Пермь: Издательский центр ЮУрГУ, 2020. С. 163–173. DOI: 10.14529/cmse150202.
17. Левин И. И., Дордопуло А. И., Гудков В. А. [и др.]. Средства программирования реконфигурируемых и гибридных вычислительных систем на основе ПЛИС // Параллельные вычислительные технологии (ПаВТ'2019) : Короткие статьи и описания плакатов XIII Международной научной конференции, Калининград, 02–04 апреля 2019 года. Калининград: Издательский центр ЮУрГУ, 2019. С. 299–312.

Дордопуло Алексей Игоревич, к.т.н., начальник, отдел материального и алгоритмического обеспечения, Общество с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров», dordopulo@superevm.ru (Таганрог, Российская Федерация)

Левин Илья Израилевич, д.т.н., профессор, кафедра интеллектуальных и многопроцессорных систем, Южный федеральный университет, Общество с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров», iilevin@sfedu.ru (Таганрог, Российская Федерация)

Гудков Вячеслав Александрович, к.т.н., доцент, кафедра интеллектуальных и много-процессорных систем, Южный федеральный университет, Общество с ограниченной ответственностью «НИИ супер-ЭВМ и нейрокомпьютеров», gudkov@superevm.ru (Таганрог, Российская Федерация)

Гуленок Андрей Александрович, к.т.н., старший научный сотрудник, отдел материального и алгоритмического обеспечения, Общество с ограниченной ответственностью «НИИ супер-ЭВМ и нейрокомпьютеров», gulenok@superevm.ru (Таганрог, Российская Федерация)

DOI: 10.14529/cmse220Y0Z

HIGH-LEVEL SYNTHESIS SOFTWARE FOR MULTI-CHIP RECONFIGURABLE COMPUTING SYSTEMS

© 2022 A.I. Dordopulo¹, I.I. Levin^{1,2}, V.A. Gudkov^{1,2}, A.A. Gulenok²

¹South Ural State University (pr. Lenina 76, Chelyabinsk, 454080 Russia),

²Lomonosov Moscow State University (GSP-1, Leninskie Gory 1, Moscow, 119991 Russia)

E-mail: dordopulo@superevm.ru, iilevin@sfedu.ru, gudkov@superevm.ru, gulenok@superevm.ru

Received: ДД.ММ.ГГГГ

The article describes an original complex of high-level synthesis that converts sequential programs into a circuit configuration of specialized hardware for reconfigurable computing systems. An absolutely parallel form, an information graph, is constructed from the original sequential program. Further, the graph is transformed into a resource-independent parallel-pipeline form – a personnel structure that can be adapted to various hardware resources. The transformation of the personnel structure into an information-equivalent structure, but occupying a smaller hardware resource, is performed using formalized methods of performance reduction, which allows you to automatically obtain a rational solution for a given multi-chip reconfigurable computing system. Unlike the known means of high-level synthesis, the result of the transformation is not the IP core of a computationally time-consuming fragment, but an automatically synchronized solution of an applied problem for all FPGA crystals of a reconfigurable computing system. Compared with parallelizing compilers, the number of analyzed variants of the synthesis of a rational solution is significantly less, which is a distinctive feature of the described complex. The application of high-level synthesis software is considered by the example of the problem of solving a system of linear algebraic equations by the Gauss method containing information-interdependent computational fragments with significantly different degrees of parallelism.

Keywords: high-level synthesis, program translation, C language, performance reduction, reconfigurable computing systems, programming of multiprocessor computing systems.

FOR CITATION

Dordopulo A.I., Levin I.I., Gudkov V.A., Gulenok A.A. High-level synthesis software for multi-chip reconfigurable computing systems. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2022. Vol. X, no. Y. P. Z1–Z2. (in Russian) DOI: 10.14529/cmse220Y0Z.

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 4.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

Список литературы

1. Voevodin V.V., Voevodin V.I. Parallel computing. BHV-Petersburg, 2002. 608 p. (in Russian).

2. Antonov A.S., Afanasyev I.V., Voevodin V. V. High-performance computing platforms: current status and development trends // Num. Meth. Prog. 2021. Vol. 22, Issue 2. P. 135–177. DOI: 10.26089/NumMet.v22r210 (in Russian).
3. Guzik V.F., Kalyaev I.A., Levin I.I. Reconfigurable computer systems. Taganrog: SFEDU Publishing, 2016. 472 p. (in Russian).
4. Levin I., Dordopulo A., Fedorov A., Kalyaev I. Reconfigurable computer systems: from the first FPGAs towards liquid cooling systems. Supercomputing Frontiers and Innovations. 2016. Vol. 3, no. 1. P. 22–40. DOI: 10.14529/jsfi160102.
5. Nane R. 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.
6. 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.
7. Nane R., Sima V.-M., Olivier B., Meeuws R., Yankova Y., Bertels K. DWARV 2.0: A CoSy-based C-to-VHDL Hardware Compiler. FPL. 2012. P. 619–622. DOI: 10.1109/FPL.2012.6339221.
8. 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.
9. 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. ACM FPGA. 2011. P. 33–36. DOI:10.1145/1950413.1950423.
10. Make Slow Software Run Fast with Vivado HLS. URL: <https://www.xilinx.com/publications/xcellonline/run-fast-with-Vivado-HLS.pdf> (accessed: 10.3.2021).
11. Vitis Unified Software Platform Documentation. Application Acceleration Development. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug1393-vitis-application-acceleration.pdf (accessed: 10.3.2021).
12. Tarasov I. Designing for Xilinx FPGAs using high-level languages in Vivado HLS environment // Components and Technologies. 2013. № 12. P. 40–48 (in Russian).
13. Kolganov A.S. An experience of applying the parallelization regions for the step-by-step parallelization of software packages using the SAPFOR system. Numerical methods and programming. 2020. Vol. 21, no. 66. P. 388–404. DOI: 10.26089/NumMet.v21r432.
14. Levin I., Dordopulo A., Gudkov V., Gulenok A., Bovkun A., Yevstafiyev G., Alekseev K. Software Development Tools for FPGA-Based Reconfigurable Systems Programming // Russian Supercomputing Days. Vol. 1129 / ed. by V. Voevodin, S. Sobolev. Cham: Springer, 2019. P. 625–640. Communications in Computer and Information Science. DOI: 10.1007/978-3-030-36592-9_51.

15. Dordopulo A.I., Levin, I.I. Performance Reduction For Automatic Development of Parallel Applications For Reconfigurable Computer Systems. Supercomputing Frontiers and Innovations. 2020. Vol. 7, no. 2. P. 4–23. DOI: 10.14529/jsfi200201.
16. Levin I.I., Dordopulo A.I., Gudkov V.A. [et al.] A set of programming tools for reconfigurable computing systems based on FPGA // Parallel Computational Technologies (PaVT'2020): Short articles and posters, Perm, March 31 – April 02, 2020. Perm: SUSU Publishing Center, 2020. P. 163–173. DOI: 10.14529/cmse150202 (in Russian).
17. Levin I.I., Dordopulo A.I., Gudkov V.A. [et al.]. Programming tools for reconfigurable and hybrid computing systems based on FPGA // Parallel Computational Technologies (PaVT'2019): Short articles and posters, Kaliningrad, 02–04 April 2019. Kaliningrad: SUSU Publishing Center, 2019. P. 299–312 (in Russian).