УДК 004.472.43 DOI 10.18522/2311-3103-2020-7-107-121

С.А. Дудко

ЭКВИВАЛЕНТНЫЕ ПРЕОБРАЗОВАНИЯ НЕКОТОРЫХ ВИДОВ РЕКУРСИВНЫХ НЕЛИНЕЙНЫХ ВЫЧИСЛИТЕЛЬНЫХ СТРУКТУР ДЛЯ ЭФФЕКТИВНОЙ РЕАЛИЗАЦИИ НА РЕКОНФИГУРИРУЕМЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

Рассматриваются методы информационно-эквивалентных преобразований некоторых видов нелинейных вычислительных структур с обратными связями: квадратичных, дробных и условных. Наличие обратных связей в конвейерной вычислительной структуре, решаемых на реконфигурируемых вычислительных системах прикладных задач, приводит к замедлению скорости формирования данных, так как для вычисления очередного значения требуется дождаться результата по обратной связи. При этом замедление происходит не только на участке с обратной связью, но и во всей вычислительной структуре, что приводит к увеличению времени, за которое данная задача может быть решена. Предыдущие фрагменты вынуждены задерживать свои данные перед подачей в обратную связь, а последующие вынуждены простаивать, ожидая данные на выходе обратной связи. На сегодняшний день не существует средств автоматического проектирования прикладных задач для реконфигурируемых вычислительных систем, которые оптимизировали бы такие вычислительные структуры в автоматическом режиме. Поэтому пользователь вынужден самостоятельно изучать текст исходной программы и искать в нем выражения, содержащие обратные связи, а затем оптимизировать их. Это приводит к увеличению времени, требующегося для создания эффективных прикладных программ. Предложенные методы преобразований позволяют сократить интервал обработки данных (в лучшем случае до единицы) при решении прикладных задач на реконфигурируемых вычислительных системах. Для реализации информационно-эквивалентных преобразований необходимо, чтобы в вычислительной системе имелся дополнительный аппаратный ресурс. Реализация данных преобразований в оптимизирующем синтезаторе схемотехнических решений позволяет проводить оптимизацию вычислительной структуры с обратными связями автоматически. Это позволяет сократить время разработки эффективных прикладных программ, содержащих обратные связи, с нескольких дней до нескольких минут.

Информационно-эквивалентные преобразования; оптимизирующий синтезатор; реконфигурируемые вычислительные системы; нелинейные вычислительные структуры.

S.A. Dudko

EQUIVALENT TRANSFORMATIONS FOR SOME KINDS OF RECURSIVE NON-LINEAR COMPUTING STRUCTURES FOR EFFICIENT IMPLEMENTATION ON RECONFIGURABLE COMPUTER SYSTEMS

In the paper, we consider data-equivalent transformations of some kinds of non-linear computing structures, such as quadratic, fractional and conditional. All computing structures contain feedbacks. If a pipeline computing structure of a task, implemented on a reconfigurable computer system, contains feedbacks, the data processing rate slows down, because it is necessary to wait for feedback results to calculate the next value. The processing rate slows down not only in the chain with feedback, but in the whole computing structure. As a result, the task solution time increases. Previous fragments have to delay their data to supply it into a chain with feedback, and subsequent ones have to remain idle waiting for the feedback result data. At present, there are no software development tools for reconfigurable computer systems with automatic optimization of such computing structures. So, the user has to analyze the source code to find expressions with feedbacks, and to optimize them. As a result, the development time of efficient applications considerably increases. We suggest methods decreasing the data processing time interval (down to unity in the best case) for applied tasks solved on reconfigurable com-

puter systems. Besides, the task solution time also decreases. Owing to the suggested methods, implemented in the optimizing synthesizer of circuit solutions, transformations are performed automatically. As a result, the development time for efficient applied tasks with feedbacks decreases from several days to several minutes.

Data-equivalent transformation; optimizing synthesizer; reconfigurable computer system; non-linear computing structure.

Введение. В настоящее время одной из проблем, снижающих производительность реконфигурируемых вычислительных систем (РВС) [1, 2], построенных на основе программируемых логических интегральных схем (ПЛИС) [3, 4], является большой интервал обработки данных в вычислительных структурах [5]. Чаще всего такая проблема возникает, когда в структуре решаемой задачи образуются обратные связи, обусловленные рекурсивными структурами самой задачи или методами организации вычислений.

Наличие рекурсивных вычислительных структур негативно сказывается на времени решения прикладной задачи, поскольку для начала очередной итерации вычислений необходимо дождаться формирования выходного сигнала в обратной связи. Это приводит к тому, что часть вычислительных ресурсов вынуждена простаивать в ожидании необходимых данных.

Пример фрагмента рекурсивной вычислительной структуры показан на рис. 1. Блок α является рекурсивным и требует для вычислений данные из блока G, а также предыдущее данное, сформированное блоком α. Блок F принимает выходные данные, полученные блоком α, и продолжает необходимые вычисления. Если блок α выполняет вычисления за S тактов, то это приводит к тому, что блок G вынужден подавать свои выходные данные на вход блока α раз в S тактов, соответственно и блок F будет получать данные для обработки один раз в S тактов. Если S>1, то это приведет к увеличению времени, требуемого на обработку всего потока данных и, соответственно, к увеличению времени решения задачи.

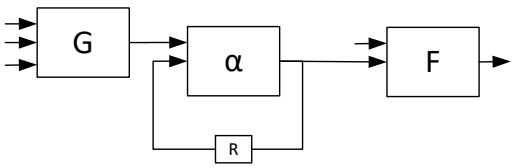


Рис. 1. Пример вычислительной структуры с обратной связью

В самом общем случае время, необходимое для решения задачи на конвейерной вычислительной структуре, при заданном аппаратном ресурсе может быть рассчитано по следующей формуле:

$$T_R \approx NS\tau,$$

где T_R – время решения задачи при доступном аппаратном ресурсе R, N – длина потока данных, S – интервал обработки данных, τ – длительность такта.

Соответственно, чтобы сократить время решения задачи, необходимо уменьшить значение одного из сомножителей данной формулы. Уменьшить количество обрабатываемых данных или длительность такта не представляется возможным, поэтому, единственным способом сокращения времени решения задачи является сокращение интервала обработки данных.

Существующие средства разработки программ для РВС [6–8], в лучшем случае могут подсказать пользователю, что в вычислительной структуре решаемой задачи обнаружены обратные связи, но провести оптимизацию подобных структур

они не способны. Пользователь вынужден самостоятельно искать образовавшиеся обратные связи в тексте исходной программы и оптимизировать их. При этом задача поиска обратных связей в исходном тексте является достаточно трудоемкой, так как структуры, образующие обратную связь, могут располагаться как в различных частях одного файла с текстом исходной программы, так и в различных связанных файлах.

За счет этого существенно возрастает время создания эффективных прикладных программ на РВС, так как даже опытному пользователю будет сложно отыскать и эффективно оптимизировать все обратные связи (особенно при их большом количестве).

Для того чтобы сократить время разработки эффективных прикладных программ, необходимо автоматизировать процедуру оптимизации обратных связей в вычислительной структуре задачи. Под оптимизацией обратных связей будем понимать процесс уменьшения интервала обработки данных (в наилучшем случае до единицы, S=1) по сравнению с интервалом исходной задачи.

Поиск и оптимизацию рекурсивных структур будем приводить не в тексте исходной программы, а в его промежуточном представлении в виде плоской вычислительной структуры, что позволит быстрее находить и обрабатывать обратные связи, так как все вершины, образующие обратную связь, располагаются последовательно друг за другом.

В работе [9] были рассмотрены методы преобразования линейных вычислительных структур с обратными связями. Настоящая статья содержит методы преобразования некоторых видов нелинейных вычислительных структур, таких как квадратичные, дробные и условные.

Преобразования квадратичных вычислительных структур. Одним из примеров нелинейных вычислительных структур [10, 11] являются квадратичные структуры, образующиеся при вычислении выражений, аналогичных квадратным уравнениям. Пример квадратного уравнения может быть представлен в следующем виде:

$$y_i = y_{i-1}^2 * \mathrm{a}_i + b_i .$$

Обозначим операцию «умножения» как β, а операцию «сложения» как φ. Тогда соответствующая данному уравнению вычислительная структура будет представлена на рис. 2.

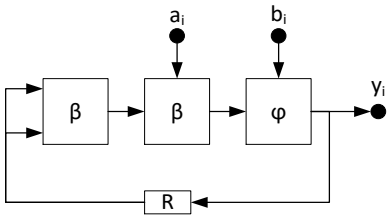


Рис. 2. Квадратичная вычислительная структура

Следует отметить, что операционные вершины β и φ могут представлять собой любые алгебраические операции, выполняемые над любыми множествами, образующими кольцо относительно данных операций [12, 13].

При наличии обратных связей интервал обработки данных может быть рассчитан по следующей формуле:

$$S = \frac{\sum L}{R}, \quad (1)$$

где L – латентность отдельных операционных вершин в пути обратной связи, R – количество регистров в обратной связи. Путь – последовательность вершин, образующая обратную связь.

Будем считать, что латентность операционных вершин β и φ равна единице. Тогда интервал обработки данных, в соответствии с формулой (1), будет равен 3. Отметим, что данная обратная связь состоит из двух аналогичных путей (так как левая вершина β имеет два входа, соответствующих выходу обратной связи), что усложняет применение информационно-эквивалентных преобразований.

Для того чтобы уменьшить интервал обработки данных в подобных структурах, воспользуемся методом автоподстановки [9]. Для этого развернем исходную обратную связь на 1 шаг (рис. 3,а), что приведет к росту необходимого для реализации аппаратного ресурса, но при этом позволит установить в обратную связь дополнительный регистр. Опираясь на эквивалентные преобразования дистрибутивных и ассоциативных операционных вершин [9], преобразуем вычислительную структуру, показанную на рис. 3,а.

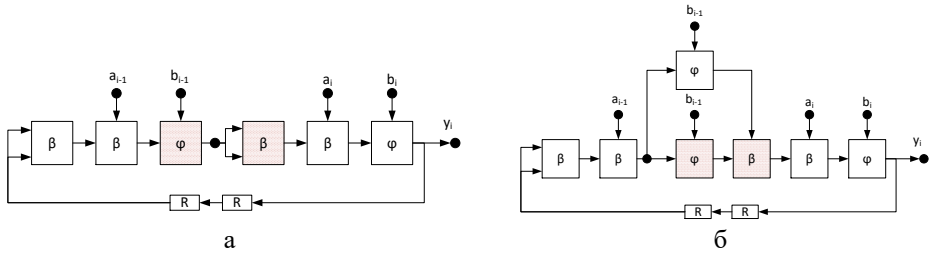


Рис. 3. Эквивалентные преобразования: а – преобразование развертки обратной связи на 1 шаг; б – дублирование операционной вершины для удаления ветвления

Как можно заметить, в обратной связи имеется операционная вершина, выходной сигнал которой ветвится на два (сигнал между заштрихованными вершинами на рис. 3,а). Данное ветвление не позволяет произвести преобразование дистрибутивных операционных вершин, поэтому необходимо избавиться от данного ветвления с помощью метода дублирования вычислений (рис. 3,б). Применение данного метода приводит к тому, что для реализации вычислительной структуры потребуется дополнительный аппаратный ресурс, необходимый на реализацию продублированной вершины. Далее, для заштрихованных на рис. 3,б вершин воспользуемся преобразованием дистрибутивных операционных вершин, после чего получим вычислительную структуру, показанную на рис. 4,а. Как можно заметить, в результате преобразования дистрибутивных операционных вершин появилась новая операционная вершина, требующая дополнительного аппаратного ресурса. После данных преобразований интервал обработки данных не уменьшился (7/2 в соответствии с формулой 1), а количество занимаемого аппаратного ресурса возросло, поэтому необходимо продолжить дальнейшие преобразования.

Повторно избавимся от образовавшихся ветвлений между заштрихованными вершинами, показанными на рис. 4,а. Полученная после дублирования операционных вершин вычислительная структура показана на рис. 4,б.

Далее продолжим выполнять преобразования дистрибутивных и ассоциативных операционных вершин до тех пор, пока это возможно.

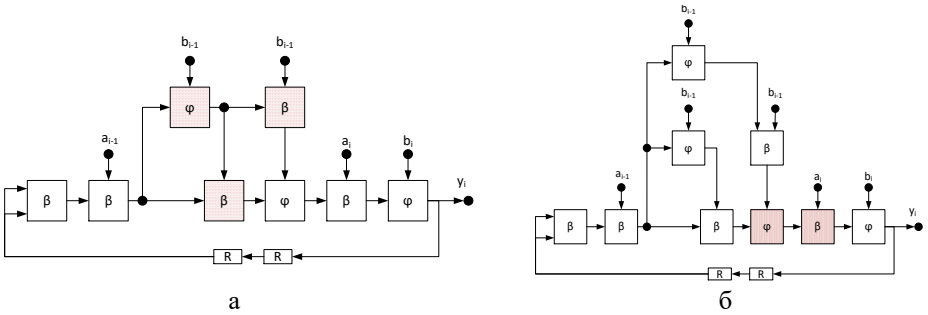


Рис. 4. Применение эквивалентных преобразований: а – преобразование дистрибутивных операционных вершин; б – дублирование операционной вершины для удаления ветвления

Полученная в итоге вычислительная структура (рис. 5), не может быть преобразована дальше с помощью дистрибутивных или ассоциативных преобразований. При этом интервал обработки данных (6/2) равен интервалу исходной вычислительной структуры, показанной на рис. 2. Поэтому для дальнейших оптимизаций можно использовать преобразование, которое будем называть обратным дистрибутивным преобразованием. Его схема показана на рис. 6.

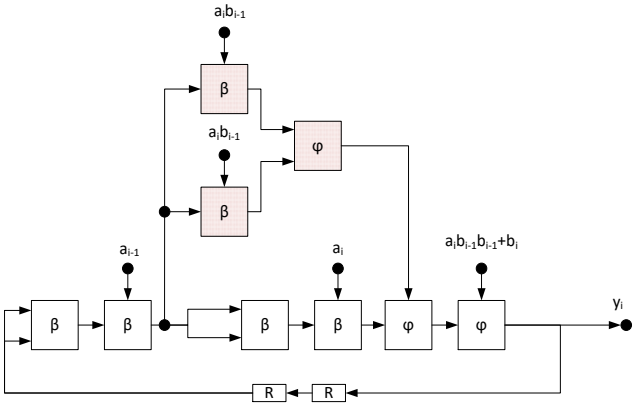


Рис. 5. Вычислительная структура после цепочки преобразований дистрибутивных и ассоциативных операционных вершин

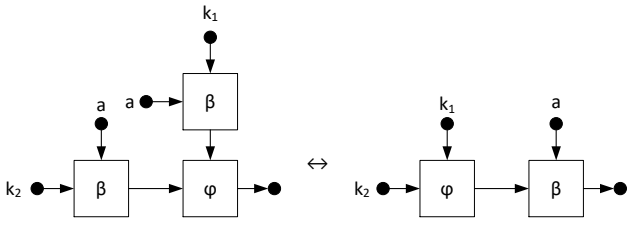


Рис. 6. Обратное преобразование дистрибутивных вершин

Если входные данные k_1 и k_2 равны между собой, то можно воспользоваться преобразованием ассоциативных вершин с общим входным операндом (рис. 7) и продолжить оптимизацию исходной вычислительной структуры.

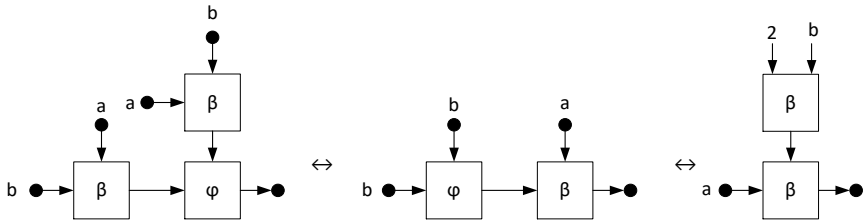
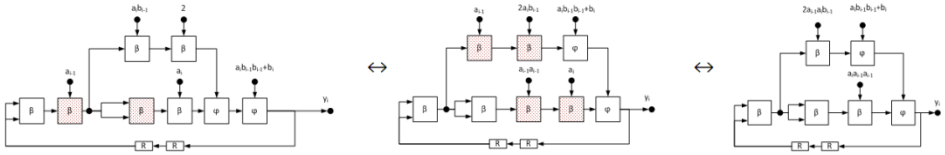


Рис. 7. Обратное преобразования дистрибутивных вершин и преобразование ассоциативных вершин с общим входным операндом

Воспользовавшись описанными выше преобразованиями, можем изменить вычислительную структуру, показанную на рис. 5, а затем применить преобразования дистрибутивных и ассоциативных операционных вершин, для того чтобы вынести часть вычислений за пределы обратной связи и, тем самым, уменьшив длину пути по обратной связи (рис. 8).



а б в

Рис. 8. Вычислительная структура после применения преобразования ассоциативных вершин

На первом шаге (рис. 8,а) необходимо избавиться от ветвления, продублировав левую заштрихованную операционную вершину β, после чего применить ассоциативное преобразование для заштрихованных вершин. На втором шаге (рис. 8,б) применяются ассоциативные преобразования для двух групп вершин, после чего получаем итоговую вычислительную структуру (рис. 8,в).

После применения всех эквивалентных преобразований получим вычислительную структуру, показанную на рис. 8,в. Интервал обработки данных в соответствии с формулой (1) для данной вычислительной структуры будет равен S = 2 (4 вершины / 2 регистра), что меньше исходного интервала обработки данных в обратной связи (S = 3) в 1,5 раза. Для того чтобы добиться более плотного потока данных, и, тем самым, повысить скорость решения задачи, можно воспользоваться методом автоподстановки повторно.

Преобразования условных вычислительных структур. При решении прикладных задач часто порядок выполнения операций зависит от выполнения каких-либо условий [14]. Для управления порядком выполнения операций применяются условные операторы. В языках программирования высокого уровня таким оператором является IF…THEN…ELSE или его аналоги [15]. При этом условные операторы также могут образовывать рекурсивные выражения. В условных рекурсивных выражениях выходное значение сигнала зависит от выполнения некоторого условия. В схемотехнике для реализации условных операторов используются мультиплексоры [16, 17]. Соответствующий данной программе фрагмент вычислительной структуры будет выглядеть следующим образом (рис. 9), где сигнал CE – какое-либо условие, отвечающее за выбор одной из ветвей вычислений:

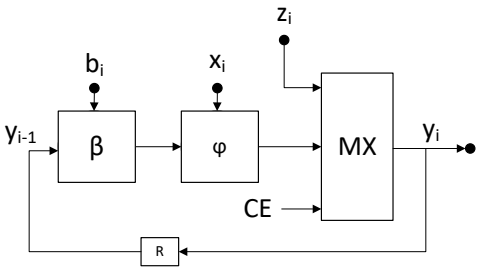


Рис. 9. Пример условной обратной связи

Мультиплексор отвечает за выбор той или иной ветви вычислений, результат которой образует обратную связь. Преобразования подобной конструкции не могут быть проведены с использованием метода автоподстановки, так как мультиплексор не обладает необходимыми свойствами дистрибутивности и ассоциативности. Поэтому для того чтобы применить к подобным структурам метод автоподстановки, необходимо заменить мультиплексор в обратной связи на набор других операционных вершин.

В общем случае для замены мультиплексора могут быть использованы логические операции «OR» и «AND» над битовым представлением данных [19, 18]. Но так как данные операции не удовлетворяют свойствам ассоциативности и дистрибутивности совместно с другими операциями в обратной связи, целесообразно заменить мультиплексор тем же набором операций, что уже входят в обратную связь (рис. 10). Данное преобразование основывается на свойствах теории групп [20].

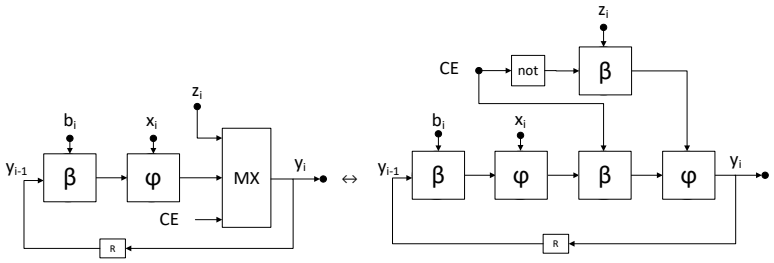


Рис. 10. Замена мультиплексора набором дистрибутивных и ассоциативных операционных вершин

Для корректности данного преобразования необходимо, чтобы множество K, над которым определены операции β и φ образовывало алгебраическое кольцо [12, 13].

По определению кольца в нем должны быть определены нейтральной «ноль» (n^0) для операции φ и нейтральная «единица» (n^1) для операции β, такие, что

$$a\ \varphi\ n^0 = n^0\ \varphi\ a = a; \tag{2}$$

$$x\ \beta\ n^1 = n^1\ \beta\ x = x; \tag{3}$$

$$a\ \beta\ n^0 = n^0\ \beta\ a = n^0. \tag{4}$$

Свойство (4) называется мультипликативным свойством нуля [20].

Преобразование замены мультиплексора набором дистрибутивных вершин (рис. 10) возможно, так как при выполнении операции β с «единицей» (n^1) полученное данное будет пропущено далее без изменений в соответствии со свойством (3). А при выполнении операции β с «нулем» (n^0) данные будут сброшены к n^0 в

соответствии со свойством (4). В итоге при выполнении операции φ с каким-либо числом и n^0 данное число будет без изменений преобразовано в выходное значение в соответствии с (2).

После замены мультиплексора набором дистрибутивных операционных вершин необходимо воспользоваться эквивалентными преобразованиями дистрибутивных и ассоциативных операционных вершин для того, чтобы упростить полученную вычислительную структуру. Данные преобразования показаны на рис. 11, где вершина $\overline{CE}$ является результатом выполнения операции «NOT» над вершиной CE. На первом шаге к заштрихованным вершинам применяется преобразование дистрибутивности, а затем на втором шаге - преобразование ассоциативности.

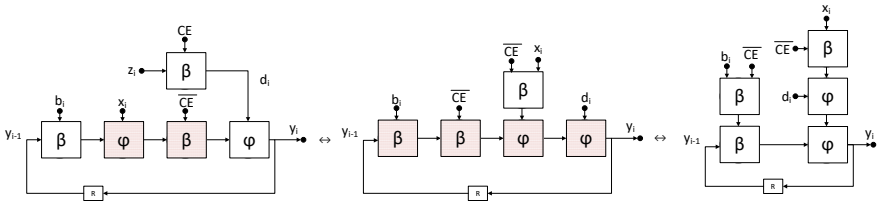


Рис. 11. Цепочка преобразований дистрибутивных и ассоциативных вершин

После этого к упрощенной вычислительной структуре для уменьшения интервала обработки данных может быть применено преобразование автоподстановки, которое было рассмотрено ранее.

Таким образом, замена мультиплексора набором дистрибутивных операционных вершин позволяет преобразовать исходную вычислительную структуру, к которой затем могут быть применены методы эквивалентных преобразований, направленные на снижение интервала обработки данных.

Эквивалентные преобразования над прямыми и обратными операционными вершинами. Перед тем как перейти к рассмотрению следующего типа нелинейных вычислительных структур, необходимо рассмотреть набор базовых эквивалентных преобразований над обратными операционными вершинами (вершинами, соответствующими обратным к β и φ операциям).

Пусть определено множество G, в котором выполняются операции β и φ. Для проведения рассматриваемых далее преобразований необходимо, чтобы данное множество образовывало алгебраическое поле [13, 20], т.е. должны выполняться следующие условия:

1) Для операций β и φ существуют нейтральные элементы n^1 и n^0 соответственно.

2) Для каждого элемента x из множества G существует обратный элемент $\bar{x}$.

3) Для операций β и φ определены обратные операции β^{-1} и φ^{-1} соответственно, такие что

$$\bar{x} = n^1\,\beta^{-1}\,x;$$
$$\bar{x} = n^0\,\varphi^{-1}\,x.$$

Если все условия выполняются, то к вычислительной структуре могут быть применены следующие эквивалентные преобразования.

Эквивалентное преобразование замены обратной операционной вершины β^{-1} прямой операционной вершиной β (рис. 12,а) осуществляется путем инвертирования входного оператора на одном из входов (в зависимости от типа дистрибутивности обратной операционной вершины). Блок «1/x» обозначает операцию получения обратного операнда.

При выполнении прямой операции с прямым и обратным операндами в результате образуется нейтральный элемент, который позволяет исключить операционную вершину из вычислительной структуры задачи (рис. 12,б).

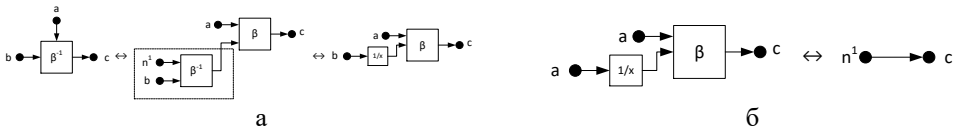


Рис. 12. Эквивалентные преобразования с обратными вершинами: а – замена обратной операционной вершины; б – сокращение прямого и обратного операндов

Если операция получения обратного значения выполняется два раза подряд, то в результате входной операнд не будет изменен. Это позволяет исключить операционные вершины получения обратного значения из вычислительной структуры (рис. 13,а).

Если на выходе операционной вершины расположена операция получения обратного значения, то она может быть перенесена на все входы данной операционной вершины. Верно и обратное: если на всех входах вершины расположены операции получения обратного значения, то они могут быть заменены одной операцией получения обратного значения на выходе данной операционной вершины (рис. 13,б).

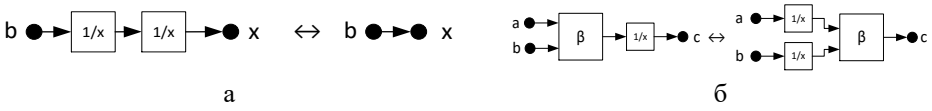


Рис. 13. Эквивалентные преобразования: а – преобразование нескольких операций взятия обратного значения; б – перенос операции получения обратного значения через вершину

Для прямой и обратной операционных вершин также определены преобразование ассоциативных вершин (рис. 14,а) и преобразования дистрибутивных вершин (рис. 14,б). В некоторых случаях дистрибутивность может быть односторонней. Тогда порядок следования операндов будет играть важное значение, как, например, в случае с операцией «деление», которая является дистрибутивной справа.

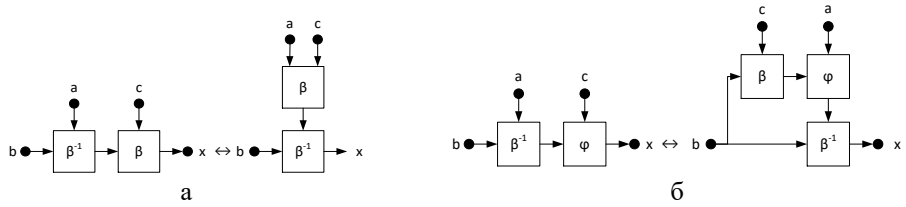


Рис. 14. Эквивалентные преобразования: а – преобразование ассоциативных обратных операционных вершин; б – преобразование дистрибутивных обратных операционных вершин

Далее рассмотрим эквивалентное преобразование «пирамиды» обратных операционных вершин с общим входным операндом, показанное на рис. 15.

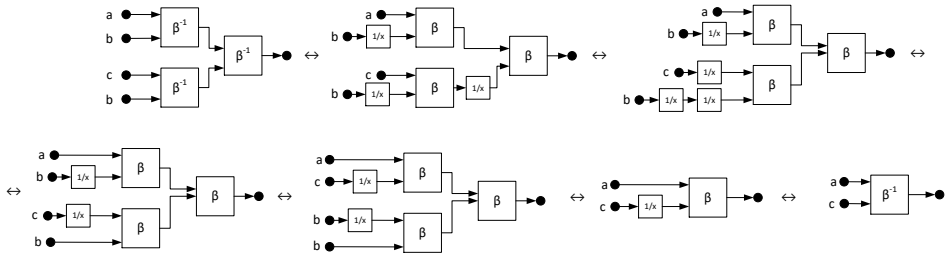


Рис. 15. Преобразование пирамиды обратных операционных вершин с общим входным операндом

Данное преобразование позволяет заменить пирамиду обратных операционных вершин одной обратной операционной вершиной путем применения преобразований, рассмотренных ранее. Данную цепочку преобразований будем называть преобразованием «пирамиды» обратных операционных вершин с одним общим входным операндом.

Использование данных эквивалентных преобразований позволяет в рассматриваемом далее типе нелинейных вычислительных структур избавиться от обратных операционных вершин и упростить структуру задачи.

Преобразования дробных вычислительных структур. Еще одним примером нелинейных вычислительных структур являются «дробные» вычислительные структуры. В ряде случаев подобные структуры могут быть оптимизированы. Для наглядности изложения рассмотрим данное преобразование для операций «+», «*» и «/», которые в общем случае соответствуют операциям «φ», «β» и «β^{-1}». Рассмотрим преобразования подобных структур на примере следующего нелинейного уравнения:

$$y_i = \frac{y_{i-1} * a_i + b_i}{y_{i-1} * c_i + d_i}.$$

Соответствующая данному уравнению вычислительная структура представлена на рис. 16.

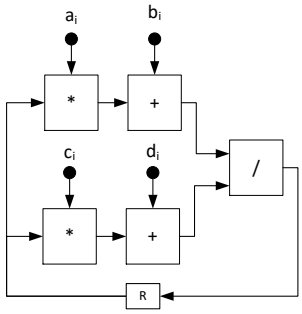
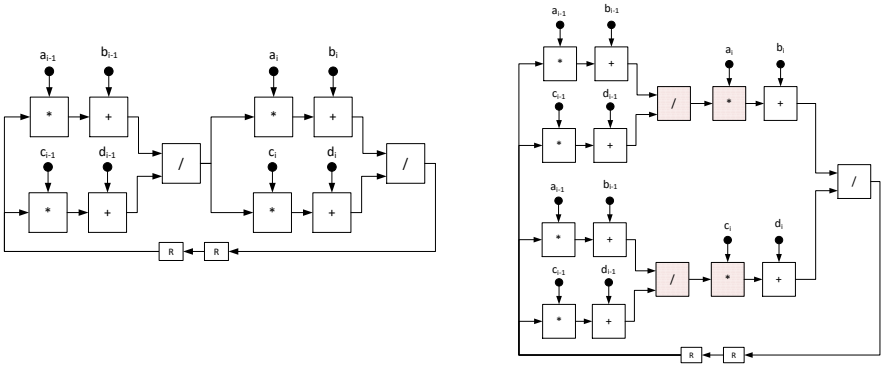


Рис. 16. Исходная нелинейная дробная вычислительная структура

Как можно заметить, в данной вычислительной структуре определена операция «деления», которая является обратной к операции умножения.

Опираясь на преобразования обратных операционных вершин, преобразуем исходную вычислительную структуру. Считая, что латентность каждой отдельной операционной вершины равна 1, получим, что интервал обработки данных в обратной связи равен 3. Попробуем оптимизировать данную вычислительную структуру, применив к ней метод автоподстановки. Развернем исходную вычислитель-

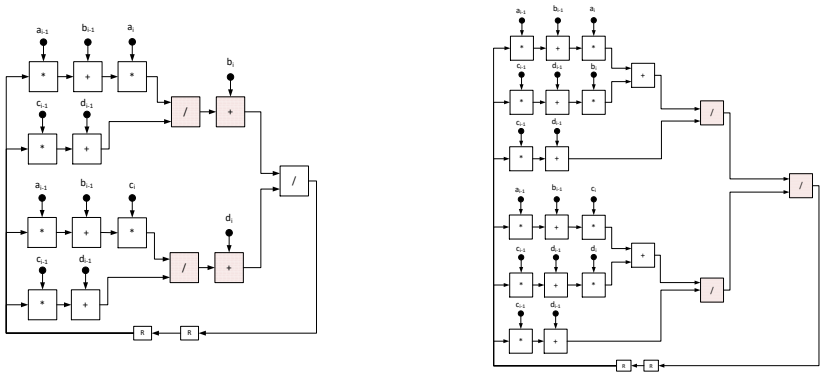
ную структуру на 1 шаг (рис. 17,а), при этом затратив дополнительный аппаратный ресурс и добавив в обратную связь дополнительный регистр. Затем избавимся от ветвлений, применив метод дублирования вычислений, получив вычислительную структуру, показанную на рис. 17,б. Интервал обработки данных для получения на рис. 17,б структуры остался без изменений (6/2 = 3).



а б

Рис. 17. Применение эквивалентных преобразований: а – преобразование автоподстановки на 1 шаг; б – преобразование дублирования вычислений

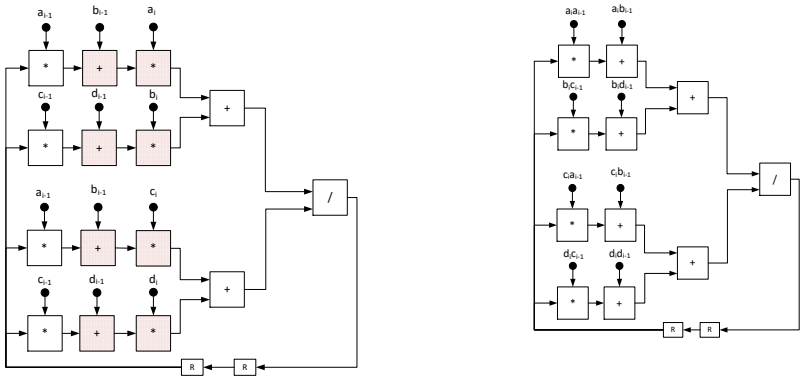
Далее, опираясь на преобразования с прямыми и обратными операционными вершинами, применим для заштрихованных вершин (рис. 17,б) преобразование ассоциативных обратных операционных вершин (рис. 18,а), а затем преобразование дистрибутивных обратных операционных вершин (рис. 18,б). Интервал обработки данных для получения на рис. 18,б структуры остался без изменений (6/2 = 3), при этом наблюдается значительный рост аппаратных затрат.



а б

Рис. 18. Вычислительная структура после применения эквивалентных преобразований: а – после преобразования ассоциативных обратных вершин; б – после преобразования дистрибутивных обратных вершин

Далее воспользуемся эквивалентным преобразованием «пирамиды» обратных операционных вершин (рис. 19,а) и преобразованием дистрибутивных операционных вершин (рис. 19,б):



а б

Рис. 19. Вычислительная структура после применения эквивалентных преобразований: а – после преобразования пирамиды обратных операционных вершин с одним общим входным операндом; б – после преобразования дистрибутивных операционных вершин

Полученная в итоге вычислительная структура обладает меньшим, чем исходная, интервалом обработки данных S = 2 (4/2), что позволяет получить результат вычислений за меньшее количество тактов. Для того чтобы добиться более плотного потока данных, метод автоподстановки может быть применен повторно, до тех пор, пока не будет получен необходимый интервал обработки данных.

Дополнительные аппаратные затраты, необходимые для преобразования нелинейных вычислительных структур, включают в себя дополнительные вершины, получаемые в ходе применения преобразований дистрибутивных операционных вершин, а также ресурс, получаемый при дублировании вычислений. В наиболее общем случае количество необходимого дополнительного аппаратного ресурса может быть рассчитано по следующей формуле:

$$\frac{N^2+N}{2} + \sum_{i=1}^{S}(k-1), \quad (5)$$

где N – исходное количество дистрибутивных операционных вершин, S –интервал обработки данных, k – количество операционных вершин в исходной обратной связи. Сумма соответствует количеству элементов, образованных в результате ветвлений.

При этом к вновь образованным ветвям также могут быть применены дистрибутивные преобразования. Поэтому применение подробных методов целесообразно в случаях, когда другие методы преобразований не подошли, и имеется достаточно высокий запас аппаратных ресурсов.

Заключение. Предложенные методы преобразования нелинейных рекурсивных выражений позволяют без участия пользователя оптимизировать фрагменты вычислительной структуры с высоким интервалом обработки данных и уменьшить время решения прикладной задачи.

Данные преобразования требуют наличия дополнительного аппаратного ресурса, что ограничивает сферу их применения. Метод автоподстановки может быть применен тогда, когда другие методы оптимизация достигли предела критического ресурса (например, при распараллеливании по итерациям закончились каналы ПЛИС).

Разработанные преобразования могут быть применены к различным типам вычислительных структур, таким, как квадратичные, дробные, а также условные. Отличительной особенностью разработанных методов является их применение для информационно-вычислительной структуры задачи, а не для текста исходной программы.

Реализация данных преобразований в оптимизирующем синтезаторе схемотехнических решений позволит проводить все преобразования в автоматическом режиме без участия пользователя и сократить срок разработки эффективных прикладных программ, содержащих обратные связи, с нескольких дней до нескольких минут.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Гузик В.Ф., Каляев И.А., Левин И.И.* Реконфигурируемые вычислительные системы: учеб. пособие / под общ. ред. И.А. Каляева. – Ростов-на-Дону: Изд-во ЮФУ, 2016. – 472 с. – ISBN 978-5-9275-1980-7.
2. *Compton K.* Reconfigurable Computing: A Survey of Systems and Software // ACM Computing Surveys. – 2002. – Vol. 34, No. 2. – P. 171-210.
3. *Попов А.Ю.* Проектирование цифровых устройств с использованием ПЛИС: учеб. пособие. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2009. – 80 с.
4. *Krishna G, Sahadev R.* Fundamentals of FPGA Architecture // Advanced Engineering Technical and Scientific Publisher. – 2017. – Part 2. – P. 12-30.
5. *Каляев А.В.* Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений. – М.: Янус-К, 2003. – 380 с.
6. Intel® Quartus® Prime Standard Edition User Guide 18.1. Getting Started. UG-20173 2018.09.24. – P. 44-47. – URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/archives/ug-qps-getting-started-18-1.pdf (дата обращения: 01.10.2020).
7. Xilinx Vivado Design Suite. User Guide. Synthesis. UG901 (v2017.1) April 19, 2017. – P. 7-38. – URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug901-vivado-synthesis.pdf (дата обращения: 25.09.2020).
8. Synopsys Identify Microsemi Edition Instrumentor User Guide, January 2018. – P. 50-51. – URL: https://www.microsemi.com/document-portal/doc_download/136672-synopsys-identify-rtl-l2016-09m-2-debugger-instrumentor-for-libero-soc-v11-8 (дата обращения: 28.09.2020).
9. *Дудко С.А.* Метод преобразования рекуррентных выражений в информационном графе // XVI Ежегодная молодежная научная конференция «Юг России: вызовы времени, открытия, перспективы»: материалы конференции (г. Ростов-на-Дону, 13–28 апреля 2020 г.). – Ростов-на-Дону: Изд-во ЮНЦ РАН, 2020. – 168 с.
10. *Агапова Е.Г.* Вычислительная математика: учеб. пособие / под ред. Т.М. Попова. 2017. – Хабаровск: Изд-во Тихоокеан. гос. ун-та, 2017. – 92 с.
11. *Lindenhovius B., Mislove M., Zamdzhiev V.* Mixed linear and non-linear recursive types // Proceedings of the ACM on Programming Languages. – 2019. – Vol. 3, Article 111.
12. *Васильев А.В., Мазуров В.Д.* Высшая алгебра: В 2 ч. // Конспект лекций. – Новосибирск: Изд-во Новосиб. гос. ун-т., 2010. – Ч. 1. – 143 с.
13. *Aditya R., Zulfikar M.T., Manik N.I.* Testing Division Rings and Fields Using a Computer Program // Procedia Computer Science. – 2015. – Vol. 59. – P. 540-549.
14. *Тюгашев А.А.* Основы программирования. Ч. I. – СПб.: Университет ИТМО, 2016. – 160 с.
15. *Nielsen F.* A Concise and Practical Introduction to Programming Algorithms in Java, Undergraduate Topics in Computer Science. – Springer-Verlag London Limited, 2009.
16. *Акчурин А.Д., Юсупов К.М.* Программирование на языке Verilog: учеб. пособие. – Казань, 2016. – 90 с.
17. *Nabulsi M., Al-Husainy M.* Using Combinational Circuits for Control Purposes // Journal of Computer Science. – 2009. – No. 5 (7). – P. 507-510.
18. *Wang X.* Estimation of Number of Bits in Binary Representation of an Integer // International Journal of Research Studies in Computer Science and Engineering. – 2015. – Vol. 2. – P. 28-31.
19. *Харрис Д.М., Харрис С.Л.* Цифровая схемотехника и архитектура компьютера. – 2-е изд., ДМК-Пресс, 2018. – 792 с.
20. *Воеводин В.В.* Линейная алгебра. – 2-е изд. – М.: Главная редакция физико-математической литературы, 1980.

REFERENCES

1. *Guzik V.F., Kalyaev I.A., Levin I.I.* Rekonfiguriruemye vychislitel'nye sistemy: ucheb. posobie [Reconfigurable computer systems: a tutorial], under the general ed. I.A. Kalyaeva, Rostov-on-Don: Izd-vo YuFU, 2016, 472 p. ISBN 978-5-9275-1980-7.
2. *Compton K.* Reconfigurable Computing: A Survey of Systems and Software, *ACM Computing Surveys,* 2002, Vol. 34, No. 2, pp. 171-210.
3. *Popov A.Yu.* Proektirovanie tsifrovykh ustroystv s ispol'zovaniem PLIS: ucheb. posobie [Designing digital devices using FPGAs: a tutorial]. Moscow: Izd-vo MGTU im. N.E. Baumana, 2009, 80 p.
4. *Krishna G, Sahadev R.* Fundamentals of FPGA Architecture, *Advanced Engineering Technical and Scientific Publisher,* 2017, Part 2, pp. 12-30.
5. *Kalyaev A.V.* Modul'no-narashchivaemye mnogoprotsessornye sistemy so strukturno-protsedurnoy organizatsiey vychisleniy [Modular-scalable multiprocessor systems with structural and procedural organization of calculations]. Moscow: Yanus-K, 2003, 380 p.
6. Intel® Quartus® Prime Standard Edition User Guide 18.1. Getting Started. UG-20173 2018.09.24, pp. 44-47. Available at: https://www.intel.com/content/dam/www/programmable/us/ en/pdfs/literature/ug/archives/ug-qps-getting-started-18-1.pdf (accessed 01 October 2020).
7. Xilinx Vivado Design Suite. User Guide. Synthesis. UG901 (v2017.1) April 19, 2017. – P. 7-38. Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug901-vivado-synthesis.pdf (accessed 25 September 2020).
8. Synopsys Identify Microsemi Edition Instrumentor User Guide, January 2018. – P. 50-51. Available at: https://www.microsemi.com/document-portal/doc_download/136672-synopsys-identify-rtl-l2016-09m-2-debugger-instrumentor-for-libero-soc-v11-8 (accessed 28 September 2020).
9. *Dudko S.A.* Metod preobrazovaniya rekurrentnykh vyrazheniy v informatsionnom grafe [Transforming method of recursive expressions in an information graph], *XVI Ezhegodnaya molodezhnaya nauchnaya konferentsiya «Yug Rossii: vyzovy vremeni, otkrytiya, perspektivy»: materialy konferentsii (g. Rostov-na-Donu, 13–28 aprelya 2020 g.)* [XVI Annual Youth Scientific Conference "South of Russia: Challenges of Time, Discoveries, Prospects": conference proceedings (Rostov-on-Don, April 13-28, 2020)]. Rostov-on-Don: Izd-vo YuNTS RAN, 2020, 168 p.
10. *Agapova E.G.* Vychislitel'naya matematika: ucheb. posobie [Computational mathematics: a tutorial], ed. by T.M. Popova. 2017. Khabarovsk: Izd-vo Tikhookean. gos. un-ta, 2017, 92 p.
11. *Lindenhovius B., Mislove M., Zamdzhiev V.* Mixed linear and non-linear recursive types, *Proceedings of the ACM on Programming Languages,* 2019, Vol. 3, Article 111.
12. *Vasil'ev A.V., Mazurov V.D.* Vysshaya algebra: V 2 ch. [Abstract Algebra: In 2 parts], *Konspekt lektsiy* [Lecture notes]. Novosibirsk: Izd-vo Novosib. gos. un-t., 2010, Part 1, 143 p.
13. *Aditya R., Zulfikar M.T., Manik N.I.* Testing Division Rings and Fields Using a Computer Program, *Procedia Computer Science,* 2015, Vol. 59, pp. 540-549.
14. *Tyugashev A.A.* Osnovy programmirovaniya [Basics of programming]. Part I. Saint Petersburg: Universitet ITMO, 2016, 160 p.
15. *Nielsen F.* A Concise and Practical Introduction to Programming Algorithms in Java, Undergraduate Topics in Computer Science. Springer-Verlag London Limited, 2009.
16. *Akchurin A.D., Yusupov K.M.* Programmirovanie na yazyke Verilog: ucheb. posobie [Verilog Programming: a tutorial]. Kazan', 2016, 90 p.
17. *Nabulsi M., Al-Husainy M.* Using Combinational Circuits for Control Purposes, *Journal of Computer Science,* 2009, No. 5 (7), pp. 507-510.
18. *Wang X.* Estimation of Number of Bits in Binary Representation of an Integer, *International Journal of Research Studies in Computer Science and Engineering,* 2015, Vol. 2, pp. 28-31.
19. *Kharris D.M., Kharris S.L.* Tsifrovaya skhemotekhnika i arkhitektura komp'yutera [Digital Design and Computer Architecture]. 2nd ed., DMK-Press, 2018, 792 p.
20. *Voevodin V.V.* Lineynaya algebra [Linear Algebra]. 2nd ed. Moscow: Glavnaya redaktsiya fiziko-matematicheskoy literatury, 1980.

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

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

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

УДК 004.382.2 DOI 10.18522/2311-3103-2020-7-121-129

А.В. Касаркин

МЕТОД РЕШЕНИЯ ГРАФОВЫХ NP-ПОЛНЫХ ЗАДАЧ НА РЕКОНФИГУРИРУЕМЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ НА ОСНОВЕ ПРИНЦИПА РАСПАРАЛЛЕЛИВАНИЯ ПО ИТЕРАЦИЯМ

При решении графовых NP-полных задач на многопроцессорных системах рост оборудования не приводит к пропорциональному росту производительности системы, поэтому не всегда удается решить задачу за приемлемое время. Целью работы, описанной в статье, является минимизация времени решения задачи поиска максимальных клик графа с использованием реконфигурируемых вычислительных систем (РВС). При решении задачи на РВС методом распараллеливания по слоям рост производительности также замедляется, несмотря на лучшую степень масштабируемости по сравнению с многопроцессорными реализациями. В статье предложен метод создания параллельно-конвейерных программ для реконфигурируемых вычислительных систем на основе распараллеливания по итерациям для решения графовых NP-полных задач. Рассмотрено, что использовать битовый способ представления множеств (как в методе распараллеливания по слоям) для метода распараллеливания по итерациям не является эффективным. Новый метод отличается организацией вычислений, а именно – обработкой неупорядоченных множеств, доступ к элементам которых осуществляется не по адресам (как в массивах), а по значениям (именам вершин и именам дуг графа). Показано, что новый метод на основе распараллеливания по итерациям, несмотря на более низкую удельную производительность, связанную с тем, что вычислительным подструктурам из-за символьного представления множеств необходимо обработать большее число промежуточных данных, обеспечивает практически линейный рост реальной производительности РВС при значительно большем количестве вычислительного ресурса по сравнению с методом распараллеливания по слоям.

Теория множеств; графовые NP-полные задачи; задача о клике; максимальные клики графа; реконфигурируемые вычислительные системы; программируемые логические интегральные схемы (ПЛИС); информационный граф; структурная реализация; конвейер; суперкомпьютеры.

A.V. Kasarkin

A METHOD FOR SOLVING GRAPH NP-COMPLETE TASKS ON RECONFIGURABLE COMPUTER SYSTEMS BASED ON THE ITERATION PARALLELIZING PRINCIPLE

When we solve graph NP-complete tasks on multiprocessor systems, the growth of hardware resource does not lead to the proportional increase of the system performance, and hence, the task solution time is not always reasonable. The aim of our research, given in the paper, is minimization of the solution time of the task of maximal clique enumeration on reconfigurable computer systems (RCS). When we solve tasks on RCSs with the help of the method of parallelizing by layers, the growth of performance also slows down in spite of better scalability in comparison with multiprocessor implementations. In the paper, we suggest a method of parallel-pipeline application development for reconfigurable computer systems. The method is based on parallelizing by