УДК 519.6:512.972, 004.27

КЛИМЕНКО И.А.

МЕТОД ОТОБРАЖЕНИЯ ЗАДАЧ НА РЕКОНФИГУРИРУЕМУЮ АРХИТЕКТУРУ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

Предложен метод отображения задач на реконфигурируемую архитектуру вычислительной системы и приведена его формализация. Отображение осуществляется на основании требований задачи ко времени вычисления и возможностей реконфигурируемой вычислительной системы с учетом ограничений, определяемых ее архитектурой.

The method of mapping tasks to the reconfigurable architecture of the computer system and its formalization is proposed. Mapping is based on the task requirements to computing time and on the opportunities of reconfigurable computing system within the constraints defined by the architecture.

1. Введение

В настоящее время в связи с широким использованием программируемых логических интегральных схем в качестве элементной базы для построения вычислительных систем актуальной становится проблема оптимизации отображения задачи на архитектуру ВС в частности на ее реконфигурируемую часть. При этом возможность физической реконфигурации архитектуры такой системы позволяет предложить различные вычислительные топологии адаптивные к типам и классам решаемых задач. Наиболее явные особенности реконфигурируемой архитектуры оказывают влияние на процесс отображения уже на этапе распараллеливания задачи. На этом этапе нет необходимости придерживаться жесткой заранее фиксированной структуры. Достаточно привести задачу к некоторому промежуточному виду после чего настроить под требования задачи архитектуру системы, для наиболее эффективного ее отображения.

Рассмотренная выше проблема отображения становится актуальной в контексте решения задачи динамической реконфигурации вычислительных систем, построенных на программируемой элементной базе (ПЛИС). Современный класс таких систем, называемый реконфигурируемыми вычислительными системами (РВС), интенсивно исследуется и развивается сообществом высокопроизводительных вычислений с целью дальнейшего повышения производительности суперкомпьютерных и кластерных вычислений. Основная концепция создания РВС обеспечение адаптивности архитектуры согласно требованиям решаемой в данный момент времени задаче при сохранении черт систем широкого использования и обеспечении при этом высокой реальной производительности. Но вместе с чрезвычайной привлекательностью перепрограммирования архитектуры РВС имеют определенный ряд проблем, поиск решений которых обуславливают большую актуальность исследований в данной области. Проблемы с одной стороны связаны с динамическими методами и средствами программирования прикладного уровня для реконфигурируемых архитектур, сложностями низкоуровневого схемотехнического программирования, с другой стороны это проблемы физического уровня реконфигурации, к которым относятся ограничения аппаратного обеспечения, высокие накладные расходы и энергозатраты на реконфигурацию.

2. Постановка задачи

На основании вышесказанного в статье предложен метод отображения задач на реконфигурируемую архитектуру вычислительной системы и сделана попытка его формализации. Предложен следующий подход – отображение задачи на реконфигурируемую архитектуру ВС осуществляется на основании заранее требуемых параметров качества решения, которые определяются требованиями со стороны решаемой задачи и возможностями системы с учетом ограничений архитектуры.

Основным требованием со стороны задачи является суммарное время вычисления, которое состоит из времени настройки архитектуры (времени реконфигурации) и времени вычисления.

В качестве ограничений со стороны системы выступают следующие параметры: ограничения аппаратного вычислительного ресурса ПЛИС, минимизация времени реконфигурации, мини-

мизация количества обмена данными на межмодульном уровне.

Любая параллельная задача предполагает операции вычисления и операции обмена данными. Поэтому, для ее представления в реконфигурируемой вычислительной среде необходимо построить такую модель, которая с одной стороны не выходит за пределы поставленных ограничений и при этом имеет минимальное количество связей между уровнями. Данная задача может быть сведена к теории графов.

3. Модель задачи

Для описания модели исходной задачи предлагается модель программирования М-задач [1, 2], которая используется для структурирования параллельных программ со смешанной параллельностью. Смешанная параллельность, характерная большинству современных требовательных к продуктивности задач, касается наличия в параллельной программе как параллелизма задач так и параллелизма данных. Множество М-задач, каждая из которых работает над различной частью приложения, объединяется в М-программу. Параллелизм данных позволяет отдельным М-задачам выполнятся в стиле *SPMD* на непересекающемся наборе процессоров, параллелизм задач (стиль *MPMD*) позволяет множеству М-задач выполняться одновременно, обмениваясь данными друг с другом. Преимущество этого подхода, в том что он позволяет увеличивать доступную степень параллелизма, определяя соответствующее строение М-задачи и ограничивая передачу данных в ее пределах. Таким образом, уменьшая коммуникационные издержки и увеличивая масштабируемость.

Значительные преимущества от смешанного параллелизма программ получают многоядерные параллельные вычислительные средства за счет высокой степени параллельности своих архитектур. Такая парадигма многоядерных вычислений может быть расширена до использования в реконфигурируемых параллельных вычислительных системах, если уровень многоядерных узлов представить реконфигурируемыми вычислительными структурами [3]. При этом, по сравнению с многоядерными архитектурами, которые требуют адаптации задачи к жёсткой архитектуре системы с сомнительным достижением максимальной продуктивности, получаем дополнительную степень параллелизма за счет гибкой гетерогенной архитектуры, которая адаптируется к структуре прикладной задачи и позволяет реализовать вычисления с максимальной продуктивностью.

В литературе рассмотрено большое количество различных методов и средств отображения М-задач на архитектуру многоядерных вычислительных структур. Это достаточно актуальная сегодня область в сфере высокопроизводительных вычислений. Однако при использовании в РВС эти методы требуют модификаций с учетом возможностей реконфигурации вычислительных узлов и ограничений, накладываемых архитектурными особенностями РВС.

Параллельные части М-программы представляются макро-графами потоков данных *MDG* (*Macro Dataflow Graphs*) с вершинами, представляющими крупно-модульные М-задачи, и с рёбрами, указывающими на зависимости между этими вершинами (рис.1) [1, 2].

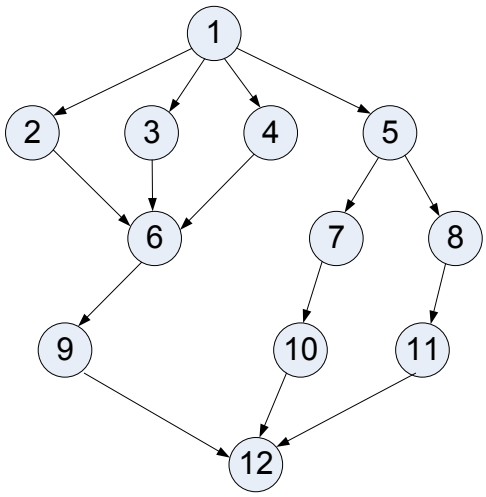

Рис. 1. Граф М-программы

В качестве примера рассмотрим граф М-программы, которая представлена графом $G_M = (V, E)$, где V – множество вершин которые представляют собой М-задачи программы, E – множество рёбер, при этом ребро $e \in E$ соединяет М-задачи M_1 и M_2, если между ними есть отношение по данным или по управлению.

4. Метод отображение задач на архитектуру РВС

Для решения задач со смешанной параллельностью в реконфигурируемых вычислительных системах с реализацией многоуровневого параллелизма архитектуры предлагается разбить процесс отображения задачи на несколько этапов, учитывающих возможности и

ограничения каждого уровня параллелизма системы.

Основные этапы метода отображения задачи на реконфигурируемую архитектуру, предлагаемого в данной работе представлены на рис. 2.

Рис.2. Основные этапы метода отображения задачи архитектуру РВС

На этапе разбиения графа М-задачи на уровни обеспечиваются такие требования со стороны задачи, как минимизация времени вычисления. Со стороны системы данное требование обеспечивается минимизацией коммуникаций на межмодульном уровне, для достижения чего предлагается такое разбиение графа М-задачи, которое обеспечивает минимальное количество связей между уровнями.

Один из эффективных подходов отображения М-программы на архитектуру мультиядерной вычислительной системы, рассмотренный в работе [1] – *алгоритм отображения по уровням*. В рамках которого граф М-программы делится на уровни независимых М-задач, и каждый уровень отображается один за другим. Предложенный способ используется нами на первом этапе для разбиения графа. Самая большая гибкость для решения задач отображения достигается при использовании как можно меньшего количества уровней. Это может быть реализовано при использовании жадного алгоритма, который работает сверху вниз по графу М-задачи и помещает так много вершин, насколько это возможно, в текущий уровень [1]. Результат такого разбиения представлен на рис. 3. Данный граф $G_M = (V, E)$ будет исходным для дальнейших преобразований.

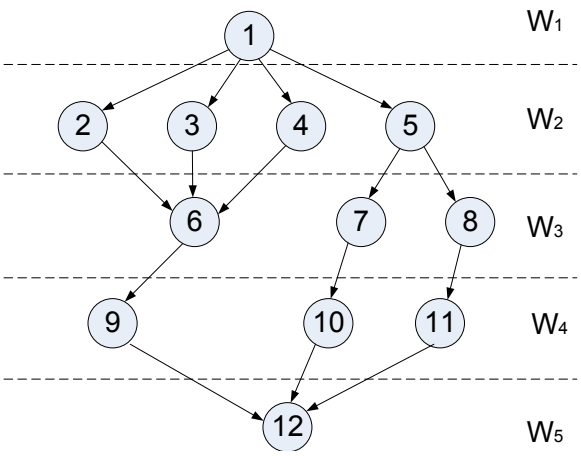

Рис. 3. Разделение графа М-задачи на уровни

3.1. Кластеризация

Алгоритм отображения по уровням, описанный в работе [1] требует модификации для эффективного использования в реконфигурируемой архитектуре. В связи с этим в работе предлагается объединение нескольких уровней графа G и формирования некоторого множества подграфов – кластеризация. Каждый подграф реализуется на одном и том же наборе конфигураций ПЛИС. Это приведет к локализации операций обмена данными в пределах вычислительного модуля. В то же время для отображения уровней в пределах каждого подграфа на архитектуру вычислительного модуля остается целесообразным реализация отображения по уровням.

Для обеспечения качества сервиса учитываются такие требования со стороны задачи, как минимизация времени вычисления. Со стороны системы данное требование обеспечивается минимизацией коммуникаций на межмодульном уровне, минимизацией коммуникаций на внутримодульном уровне и уменьшением количества последовательных реконфигураций архитектуры. Для достижения чего предлагается такое разбиение графа М-задачи на подграфы, которое обеспечивает минимальное количество связей между кластерами, и минимальное количество уровней подграфа.

Как видно на графе (рис. 3), общий объем связей между М-задачами фиксированный. Согласно модели архитектуры подмножества М-задач отображаются на один/несколько вычислительных модулей, а в их пределах – на од-

ну/несколько конфигураций ПЛИС. Тогда межзадачные связи могут быть поделены на два типа – внешние межмодульные связи и внутримодульные связи между конфигурациями ПЛИС одного модуля. Считаем, что при выполнении кластеризации максимизация коммутаций внутри вычислительного модуля приводит к минимизации межмодульных связей.

Тогда исходный граф G_M разбивается на множество подграфов $C_i \in G_M$ ($i = \overline{1,n}$, где n – количество подграфов), таким образом что бы количество связей внутри подграфа было максимальным [4].

В качестве альтернативного средства может быть предложена кластеризация с минимизацией количества связей между подграфами на базе метода определения минимального сечения подграфа, описанного в работе [5].

Полученный граф представлен на рис. 4 *а*. Для внешнего алгоритма управления, реализуемого центральным процессором граф М-задачи соответствует представленному на рис. 4, *б*. Количество уровней уменьшилось до трех, связи между вершинами соответствуют оставшимся межмодульным связям. Граф М-задачи будет отображаться на архитектуру системы и выполняться согласно алгоритму отображения по уровням. В качестве минимальной вычислительной единицы архитектуры для отображения рассматривается вычислительный модуль. Управление решением подграфа осуществляется соответствующими средствами вычислительного модуля, которые аналогичным образом реализуют алгоритм отображения по уровням.

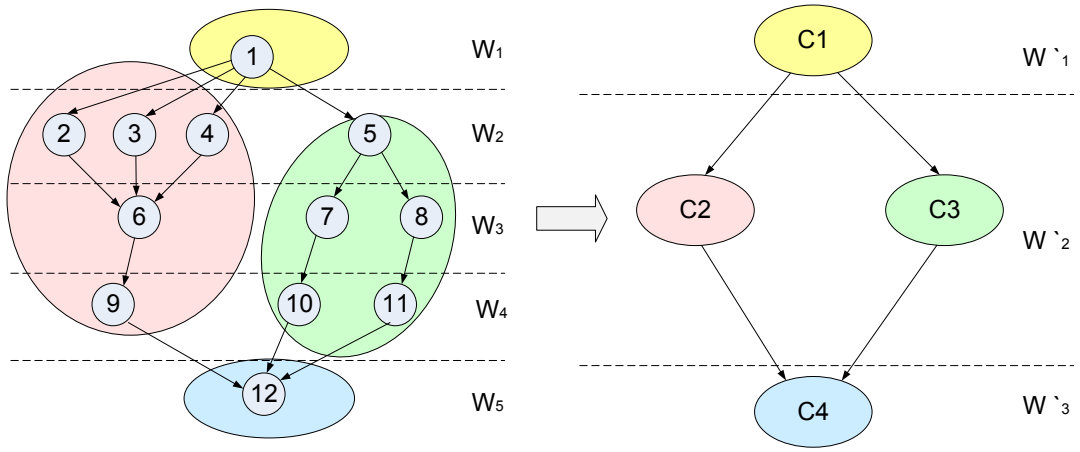

Рис. 4. Кластеризация графа М-задачи

Предварительную оценку эффекта от проведенной кластеризации можно оценить на основании параметра выигрыша P_{COMM}, который определяет время межзадачного взаимодействия, сохраненное за счет максимизации внутримодульных связей, и выражается следующей формулой [4]:

$$P_{COMM} = \frac{U_{COMM}}{B_{IO}} \times 2,$$

где U_{COMM} – объем межзадачных связей, перешедших на внутримодульный уровень, B_{IO} – доступная пропускная способность ввода\вывода между конфигурациями ПЛИС и основной памятью системы. На данном этапе для упрощения формализации кластеризации мы делаем предположение о том что время внутримодульного взаимодействия минимально и пренебрегаем этой величиной. Такое предположение приемлемо, поскольку топология вычислительного модуля еще не рассматривается и невозможно оценить соотношение коммуникаций между конфигурациями ПЛИС и внутрикристальными взаимодействиями, которые по времени действительно могут быть сведены к нулю. Однако в дальнейшем величина параметра прибыли P_{COMM}, должна быть откорректирована с учетом времени внутримодульного обмена для связей U_{COMM} и различных топологий вычислительных модулей.

3.2. Оптимизация

Дальнейшая оптимизация структуры кластера локализуется на уровне архитектуры вычислительного модуля. Со стороны задачи учитывается требование минимизации времени вычисления, которое обеспечивается системой

путем уменьшения времени реконфигурации архитектуры, со стороны системы учитываются ограничения аппаратных ресурсов вычислительного модуля за счет введения ограничения на количество вершин каждого уровня подграфа

Не рассматривая на данном этапе топологии вычислительных модулей сделаем некоторые предположения относительно архитектуры вычислительного модуля и введем абстрактный шаблон архитектуры. Это позволит формализовать оптимизацию графа М-задачи и в дальнейшем рассмотреть наиболее эффективные топологии вычислительных модулей для решения различных классов задач. Определим общее количество вычислительного ресурса одного модуля, как некоторую вычислительную площадь $N = \{G, S\}$, состоящую из групп G взаимосвязанных виртуальных вычислительных структур S. Количество g групп соответствует количеству конфигураций ПЛИС одного вычислительного модуля, количество s виртуальных вычислительных структур ограничено аппаратными возможностями одной конфигурации ПЛИС. При этом все вычислительные структуры S_l ($l = \overline{1,s}$) однотипны, их количество s в каждой группе одинаково так же, как и количество g групп одинаково во всех вычислительных модулях. Модель обобщенной архитектуры вычислительного узла представлена на рис.5.

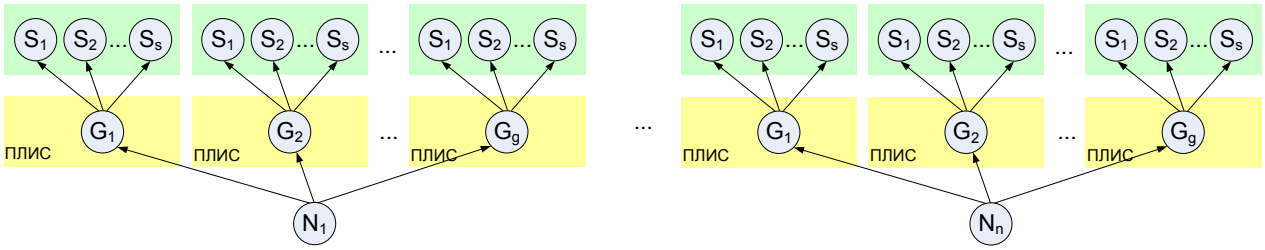

Рис. 5. Модель архитектуры РВС

Предопределенное количество вычислительного ресурса одного модуля, равное $(g \times s)$ вычислительных структур, а исходя из этого, и каналов связи между конфигурациями ПЛИС ограничивает количество вершин, которые могут быть размещены на одном уровне подграфа *Ci*. Для достижения цели оптимизации структуры подграфа, согласно ограничениям аппаратных ресурсов, полученный граф может быть трансформирован без нарушения структуры связей – что подразумевает перемещение вершин с одного уровня на другой (рис. 6). С другой стороны, согласно алгоритму отображения по уровням каждый уровень подграфа будет последовательно отображен на архитектуру, которая должна быть предварительно реконфигурирована, и выполнен. Таким образом время выполнения М-задач каждого уровня подграфа будет равно

$$T_{sum\ \max Wj} = (T_{commVCi} + T_{reconfVCi})_{\max}, \qquad (1)$$

где $T_{sum\max Wj}$ – максимальное суммарное время выполнения М-задач, входящих в текущий уровень W_j ($j = \overline{1,w}$, где w – количество уровней), $T_{commVCi}$ – время вычисления и $T_{reconfVCi}$ – время реконфигурации М-задачи соответствующей некоторой вершине V подграфа C_i. Согласно этому неограниченное увеличение количества уровней подграфа во время трансформации приведет к значительному увеличению суммарного времени вычисления в первую очередь за счет накладных расходов на реконфигурацию архитектуры. Это нарушает исходное требование минимизации суммарного времени вычисления.

В этой связи определим условия оптимизации структуры подграфа:

1. Количество вершин в одном уровне – максимально возможное, но не больше, чем $(v_{\max} = g \times s)$ вершин.

2. Количество уровней – максимальное количество уровней ограниченно коэффициентом *чувствительности ко времени* М-задачи K_T:

– $(K_T = 1)$ – задача требует минимально возможного времени вычисления, что выражается в формировании минимально возможного количества уровней, возможно, с применением дополнительных средств минимизации количества уровней или уменьшения времени реконфигурации;

– ($K_T = 0$) – задача не чувствительна к времени вычисления, допускается произвольное количество уровней, оптимизация выполняется только по критерию 1.

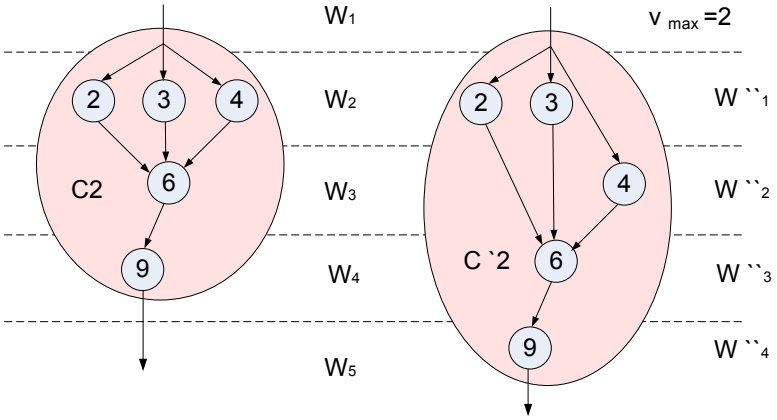

Рис. 6. Оптимизация структуры подграфа К2

Количество задач, возложенных на каждую конфигурацию, зависит от выделенных ресурсов для каждой задачи. При этом время реконфигурации каждой задачи $P_{cnf\ task}$ равно доле времени от полной конфигурации всего FPGA устройства $T_{cnf\ FPGA}$ и пропорционально количеству используемых задачей ресурсов [4]:

$$P_{cnf\ task} = \frac{S_{reg}}{S_{full}} \times T_{cnf\ FPGA},$$

где S_{req} – ресурсы используемые одной задачей, S_{full} – полный ресурс FPGA.

Мы рассматриваем только полную конфигурацию FPGA, при этом отображение по уровням, даже в случае решения нескольких задач на одной конфигурации ПЛИС подразумевает их одновременную загрузку в микросхему и выполнение согласно (1). Тогда ($S_{reg} = S_{full}$) и для оценки времени реконфигурации используем следующее выражение:

$$P_{cnf\ task} = T_{cnf\ FPGA}.$$

При таком подходе обеспечение соответствия критерию оптимизации 2 возможно только путем минимизации количества уровней подграфа. Для расширения возможностей с точки зрения уменьшения времени реконфигурации можно рассматривать ряд дополнительных средств. Например, пересмотр структуры подграфа с целью уменьшения количества вершин на критичном уровне, расширение вычислительного ресурса за счет ресурсов других вычислительных модулей, частичная реконфигурация [6], поиск изоморфных структур и реализация их на одних и тех же конфигурациях ПЛИС [6], кеширование конфигураций [7], планирование конфигураций заранее [8], использование адаптивных топологий архитектуры вычислительного модуля [6]. Однако это выходит за рамки данной работы.

Что касается учета ограничений внутримодульных коммуникационных связей при оптимизации подграфа М-программы, вопрос так же остается открытым в данной работе, поскольку они напрямую зависят от топологии архитектуры вычислительного модуля, которая на данном этапе не рассматриваются.

3.3. Отображение

Этап отображения представлен в общем виде, поскольку принятая за основу абстрактная модель архитектуры не детализирует топологию вычислительных структур и не рассматривает требований к архитектуре со стороны классов задач. Отображение М-программы на соответствующую реконфигурируемую архитектуру – это отображение F группы М-задач М на структуру A, описывающую архитектуру вычислительного модуля, то есть, $F : M \to A.$ Согласно методологии описанной в работе [1], определим последовательность физических виртуальных вычислительных структур для отображения (рис. 5)

$$s_1, s_2, \ldots, s_{(s \times g)} \qquad (2)$$

Отображающая функция F присваивает группам М-задач M_i, $i = \overline{1, v}$ (где v – количество М-задач на одном уровне кластера) физические виртуальные структуры из последовательности (2):

$$F(M_i) = \left\{ s_j, s_{j+1}, \ldots, s_{j+|M_i|-1} \,\middle|\, j = 1 + \sum_{k=1}^{i-1} |M_k| \right\}.$$

4. Выводы

1. Предложенный метод отображения параллельных задач на реконфигурируемую архитектуру, за счет учета требований со стороны задачи и возможностей вычислительной системы, позволяет получить оптимальное отображение задачи с целью повышения производительности. При этом, в качестве основных критериев оптимизации отображения задачи приняты минимизация времени вычисления, значительным образом определяемое количеством межмодульных взаимодействий и временем реконфигурации архитектуры, и ограничения аппаратных ресурсов, которые являются критическими параметрами, влияющими на производительность реконфигурируемых вычислительных систем.

2. Для решения задач со смешанной параллельностью в вычислительных системах с реализацией многоуровневого параллелизма архитектуры, в том числе с возможностью реконфигурации вычислительных узлов, целесообразно разбить процесс оптимизации отображения задачи на несколько этапов, учитывающих возможности и ограничения каждого уровня параллелизма системы. Предложенный метод, в связи с этим, предполагает нескольких этапов выполнения. На этапах разбиения графа задачи на уровни и кластеризации осуществляется минимизация времени вычисления задачи за счет уменьшения количества операций обмена данными на межмодульном уровне. Дальнейшая оптимизация структуры кластера локализуется на уровне архитектуры вычислительного модуля – учитываются ограничения аппаратных ресурсов вычислительного модуля, осуществляется минимизация времени вычисления, за счет уменьшения времени реконфигурации архитектуры. Минимизация времени реконфигурации в данной работе не рассматривается, для этого предлагается использовать известные, описанные в литературе методы. Этап отображения задачи представлен в общем виде без детализации топологии вычислительного модуля.

3. Топология вычислительных модулей является важным фактором, влияющим на продуктивность вычислений в реконфигурируемых вычислительных системах. В связи с чем дальнейшая работа над методом предполагает разработку типовых архитектур эффективных для решения различных классов задач и на их основе модификацию этапа отображения.

Список литературы

1. Dümmler J. Scalable computing with parallel tasks / J. Dümmler, T. Rauber, G. Rünger // Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS '09), (November 16, Portland, Oregon) – Article No. 9 – ACM New York, NY, USA, 2009. – P. 1-10.
2. Bansal S., Kumar P., Singh K. An improved two-step algorithm for task and data parallel scheduling in distributed memory machines // Parallel Computing. – Volume 32, Issue 10. – 2006. – P. 759–774.
3. Smith M.C. Optimization of Shared High-Performance Reconfigurable Computing Resources / M.C. Smith, G.D. Peterson // ACM Transactions on Embedded Computing Systems. – Vol. 11, No. 2, Article 36. – 2012. – 22 p.
4. Reconfigurable Computing / M. Huang, V.K. Narayana, H. Simmler [and all] // ACM Transactions on Reconfigurable Technology and Systems (TRETS). – Vol. 3, Issue 4, Article 20. – ACM New York, NY, USA, 2010. – 25 p.
5. Кулаков Ю.А. Формирование структуры корпоративной сети / Ю.А. Кулаков, Халиль Х. А. Аль Шкерат // Гірнича електромеханіка та автоматика: Наук.- техн. Зб. – 2003. – Вип. 70. – С. 86 - 91
6. Panella A. A design workflow for dynamically reconfigurable multi-FPGA systems / A. Panella, M.D. Santambrogio, F.Redaelli [and all] // Proceeding in 18th VLSI System on Chip Conference (VLSI-SoC), (27-29 September, 2010). – IEEE/IFIP, 2010. – P. 414 – 419.
7. El-Araby E. Exploiting Partial Runtime Reconfiguration for High-Performance Reconfigurable Computing / E. El-Araby E., I. Gonzalez, T. El-Ghazawi // ACM Transactions on Reconfigurable Technology and Systems (TRETS). – Vol. 1, Issue 4, Article 36. – ACM New York, NY, USA, 2009. – 23 p.
8. Resano J. Efficiently Scheduling Runtime Reconfigurations / J. Resano, J.A. Clemente, C. Gonzalez, [and all] // ACM Transactions on Design Automation of Electronic Systems (TODAES). – Vol. 13, Issue 4, Article 58. – ACM New York, NY, USA, 2008 – 12 p.