УДК 004.272.44

Оптимизация параллельных списков функционально-потокового языка программирования «Пифагор»

В.С. Васильев[a], И.Н. Рыженко[b], И.В. Матковский[c]

Сибирский федеральный университет, пр. Свободный 78, Красноярск, Россия
[a]rrrFer@mail.ru, [b]rodgi.krs@gmail.com, [c]Alpha900i@mail.ru
Статья поступила 25.05.2014, принята 20.08.2014

Рассматриваются оптимизирующие преобразования параллельных списков в программах, написанных на языке функционально-потокового программирования «Пифагор». Параллельные списки являются выразительным языковым средством, обеспечивая описание независимых данных и функций. Их использование позволяет компактно описать параллельные фрагменты за счет наличия в языке возможностей эквивалентного преобразования в более простые структуры. Параллельные списки могут вкладываться вдруг в друга, формируя сложные иерархические структуры, которые, в ряде случаев, могут быть выявлены и преобразованы в структуры с меньшим уровнем вложенности на этапе трансляции программы. Выполнение операторов программы на языке Пифагор контролируется управляющими автоматами, которые значительно осложняются тем, что на вход соответствующих операторов могут поступать параллельные списки. В случаях, когда удается установить, что параллельный список не содержит вложенных параллельных списков, размерность которых зависит от аргумента функции, возможна интерпретация списка на этапе трансляции. Интерпретация списка не только является самостоятельным оптимизирующим преобразованием, но и повышает эффективность применения других преобразований, что показано в статье на примере оптимизации управляющих автоматов. В ряде случаев такое преобразование приведет к полному удалению параллельных списков за счет дополнительного применения эквивалентных преобразований, закрепленных в модели вычислений. В заключительной части представлены ограничения, накладываемые архитектурой на программу и позволяющие реализовывать трансляцию функционально-потоковых параллельных программ в топологию сверхбольших интегральных схем. Также показана возможность более эффективной оптимизации параллельных списков при трансляции на архитектуру систем на кристалле за счет учета дополнительных ограничений.

Ключевые слова: функциональное программирование, оптимизация кода, разработка СБИС, параллельное программирование.

Optimization of parallel lists of the functional and dataflow programming language PIFAGOR

V.S. Vasiliev[a], I.N. Ryzhenko[b], I.V. Matkovsky[c]

Siberian Federal University, 78 Svobodny ave., Krasnoyarsk, Russia
[a]rrrFer@mail.ru, [b]rodgi.krs@gmail.com, [c]Alpha900i@mail.ru
Received 25.05.2014, accepted 20.08.2014

This research examines the optimizing transformations of the parallel lists in the programmes written on the functional and dataflow programming language PIFAGOR. The parallel lists describes independent data and functions of the PIFAGOR language. This lists are used for define parallel fragments of code. The parallel lists can make hierarchy structures. In some cases this structures can be identified and simplified at the stage of program translation. Running the operator of the program on PIFAGOR is controlled by control machines and become more complicated because of the parallel lists being input on operators. If it is possible to identify that there are no attached parallel lists in the initial parallel list, depending on the argument of function, it is possible to interpret the lists at the stage of program translation. Interpretation of the list is not only independent optimizing transformation but also it is a way to increase the efficiency of the use of other transformations. In the article it is demonstrated by the example of optimization of control machines. In some cases this transformation can give the possibility for a complete delete of the parallel lists due to additional usage of the equivalent transformations determined at the computing model. The conclusion of the article shows the possibility for more effective optimization of the parallel lists for system on a chip/VLSI circuit architecture due to additional restrictions applied by the architecture.

Keywords: functional programming, code optimization, VLSI design, parallel programming.

Введение. Разработка параллельных программ в настоящее время осуществляется в основном с применением средств архитектурно-зависимого программирования. Это обеспечивает хорошую ручную оптимизацию и позволяет учесть особенности используемых параллельных вычислительных систем (ПВС). Помимо этого для различных ПВС существуют дополнительные специализированные средства, оптимизации разработанных программ и их балансировки. Применение данного подхода ведет к тому, что приходится переписы-

вать ранее разработанные программы при переходе с одной вычислительной системы на другую. Для того, чтобы уменьшить зависимость от конкретных архитектур и повысить переносимость параллельных программ, предлагаются методы и языки архитектурно-независимого параллельного программирования. Одним из них является язык функционально-потокового параллельного программирования «Пифагор», базирующийся на функционально потоковой модели параллельных вычислений [1]. В нем используется управление вычислениями по готовности данных. Для повышения выразительности средств описания параллелизма предложены соответствующие операторы, среди которых присутствует и оператор формирования параллельного списка.

Существующая в модели вычислений алгебра эквивалентных преобразований [2] позволяет упрощать формируемые параллельные списки во время выполнения вычислений. Однако такие преобразования в ряде случаев могут проводиться для получения более оптимального кода и во время трансляции программы, для чего необходима разработка соответствующих методов оптимизации. Используемая в настоящее время четырехслойная модель выполняемой программы [3] позволяет использовать дополнительную оптимизацию на различных этапах формирования исполняемого представления, включая и автоматный слой. Каждый автомат промежуточного представление отражает состояние соответствующего оператора программы как до, так и в процессе его выполнения. Состояния автоматов меняются в зависимости от поступающих управляющих сигналов. Автоматы, отвечающие за обработку оператора преобразования параллельных списков, обладают относительно высокой сложностью. Помимо этого, в ходе вычислений могут формироваться параллельные списки, вложенные друг в друга, что ведет к формированию иерархических зависимостей, замедляющих процесс обработки. Вместе с тем, многие из подобных зависимостей формируются во время написания программы и могут быть идентифицированы соответствующими оптимизаторами во время трансляции, что позволит их трансформировать в более простые конструкции в соответствии с алгеброй преобразований. Кроме того, в ряде случаев размерность параллельных списков известна на этапе трансляции, возможно обнаружение и досрочная интерпретация таких списков.

В силу того, что язык Пифагор предназначен для архитектурно-независимой разработки программ, проектирование ведется без учета особенностей целевой архитектуры. Однако не на всех архитектурах возможна полноценная реализация конструкций языка. В статье на примере топологии сверхбольших интегральных схем показано, что учет особенностей архитектуры позволяет более эффективно проводить оптимизирующие преобразования.

В работе предлагаются подходы к статической оптимизации вложенных параллельных списков на этапе трансляции программы в ходе формирования промежуточного представления, используемого в интерпретаторе ФПП программ. На основе допустимых эквивалентных преобразований обеспечивается анализ структур, сформированных компилятором, после чего проводится требуемая оптимизация как информационного графа программы, так и управляющих автоматов четырехслойной модели. Рассматриваются возможности использования результатов преобразований для повышения эффективности топологии систем на кристалле (СнК), разрабатываемых на основе ФПП программ.

Оптимизации на основе эквивалентных преобразований, определенных моделью вычислений. Применение функции к параллельному списку в модели функционально-потоковых параллельных вычислений и в языке ФПП программирования Пифагор эквивалентно ее применению к каждому элементу списка и формированию параллельного списка результатов вычислений:

$$[d1, d2, \ldots, dn] : f \rightarrow [d1 : f, d2 : f, \ldots, dn : f] .$$

Аналогичный параллельный список получается, если несколько функций обрабатывают один и тот же элемент данных:

$$d : [f1, f2, \ldots, fn] \rightarrow [d : f1, d : f2, \ldots, d : fn] .$$

В более общем случае параллельными списками могут быть представлены как данные, так и функция. В результате формируется параллельный список, размерность которого совпадает с произведением размерностей аргументов оператора интерпретации. Порядок следования элементов определяется в первую очередь данными:

$$[d1, d2, \ldots, dn] : [f1, f2, \ldots, fn] \rightarrow$$

$$[d1 : f1, d1 : f2, \ldots, d1 : fn, d2 : f1, \ldots, dn : fn].$$

Вместе с тем, возможна ситуация, когда внутри параллельного списка могут оказаться аргументы, также являющиеся параллельными списками. В этом случае порядок их раскрытия не оговаривается, что обуславливается возможностью выполнения параллельных вычислений на различных вычислительных ресурсах, не связанных между собой. Поэтому алгебра преобразований допускает последовательную трансформацию аргументов оператора интерпретации без изменения иерархической вложенности параллельных списков. Например, для вложенных данных и одной функции подобные преобразования могут выглядеть следующим образом:

$$[d1, [d2, [d3, d4], d5], d6]:f1 \rightarrow$$

$$[d1:f1, [d2, [d3, d4], d5]:f1, d6:f1] \rightarrow$$
$$[d1:f1, [d2:f1, [d3, d4]:f1, d5:f1], d6:f1] \rightarrow$$
$$[d1:f1, [d2:f1, [d3:f1, d4:f1], d5:f1], d6:f1] .$$

В большинстве операций обработки данных дальнейшие вычисления, использующие вложенные параллельные списки, сводятся к их преобразованию в один линейный параллельный список, что обуславливается

особенностью модели вычислений. В частности, подобная ситуация происходит, когда параллельный список появляется внутри списка данных, порождающего в результате вектор:

([d1, [d2, [d3, d4], d5], d6]) →

(d1, d2, d3, d4, d5, d6) ,

или когда к параллельному списку применяется функция внутри списка данных:

[d1, [d2, [d3, d4], d5], d6]:(.) →

(d1, d2, d3, d4, d5, d6) .

Поэтому в тех случаях, когда возможность избавления от вложенных параллельных списков просматривается во время трансляции программы, этот вопрос можно решить в ходе соответствующей оптимизации, получив в результате более простую структуру без каких-либо преобразований на этапе вычислений:

[d1, [d2, [d3, d4], d5], d6] →

[d1, d2, d3, d4, d5, d6] .

Кроме того, одноэлементный параллельный список согласно модели вычислений заменяется на значение своего единственного элемента:

[a] → a .

Одноэлементные параллельные списки могут появляться в программах в результате применения различных оптимизирующих преобразований, а также часто используются для изменения порядка вычислений.

Листинг 1. Не оптимизированный код

```
test << funcdef X {
  Y1 << [1, [2, X, 3], 4];
      Z1 << Y1 : foo;
Y2 << [1, [2, X : [], 3], 4];
      Z2 << Y3 : bar;
    (Z1, Z2) >> return;
              }
```

В приведенной на листинге 1 функции вложенный параллельный список Y1 может быть оптимизирован на этапе трансляции, т. к. он содержит лишь константы и аргумент (который не может являться параллельным списком). В связи с этим становится известным размер параллельного списка и появляется возможность на этапе трансляции выполнить его интерпретацию.

Размерность параллельного списка Y2 на этапе трансляции определить невозможно, т. к. она будет зависеть от поданного на вход функции аргумента и станет известной лишь во время выполнения программы. В связи с этим невозможно определить номера входов, по которым в узел Y2 поступят константы 3 и 4, поэтому оптимизация списков Y2 и Z2 описанными методами невозможна.

Листинг 2. Частично оптимизированный код

```
test << funcdef X {
    Y1 << [1, 2, X, 3, 4];
Z1 << [1:foo, 2:foo, X:foo, 3:foo,
          4:foo];
  Y2 << [1, [2, X : [], 3], 4];
      Z2 << Y3 : bar;
    (Z1, Z2) >> return;
              }
```

После интерпретации параллельный список Y1 может быть полностью удален из программы как неиспользуемый код. Параллельный список Z1 также будет содержать 5 элементов и тоже может быть устранен, т. к. помещается в список данных.

Листинг 3. Полностью оптимизированный код

```
test << funcdef X {
  Y2 << [1, [2, X : [], 3], 4];
      Z2 << Y3 : bar;
(1:foo, 2:foo, X:foo, 3:foo, 4:foo, Z2)
          >> return;
              }
```

Четырехслойная модель интерпретатора в контексте оптимизации. В настоящее время программа на языке Пифагор при выполнении представляется в интерпретаторе в виде четырех слоев [3, 4]:

- слой реверсивного информационного графа (РИГ) отражает зависимости по данным, существующие между операторами программы;
- слой управляющего графа (УГ) содержит зависимости по управлению;
- константы и значения, вычисленные во время выполнения программы, хранятся в слое данных;
- логика работы операторов и их текущее состояние описываются автоматным слоем.

Во время трансляции программы выполняется начальное формирование всех слоев, в это же время возможно проведение над ними оптимизирующих преобразований.

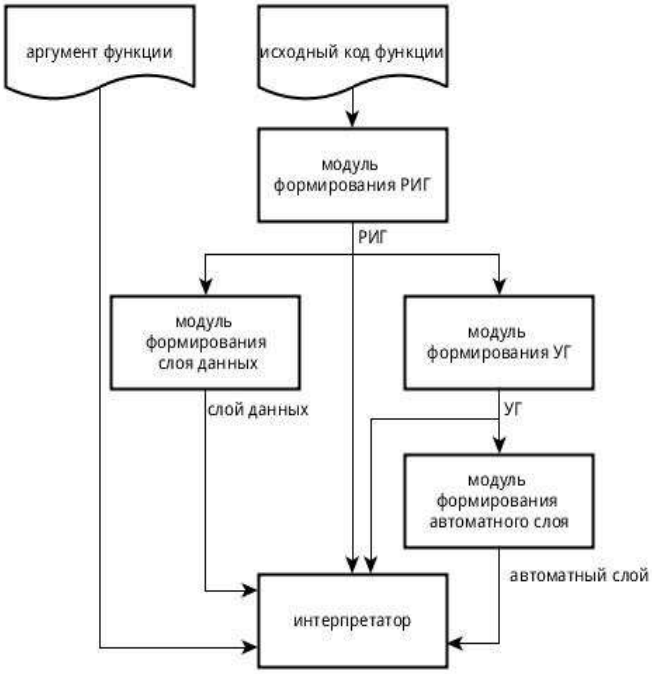

Рис. 1. Формирование слоев интерпретатора

Во время работы программы происходит вычисление узлов РИГ. По завершении обработки узла РИГ результат помещается в слой данных, и вырабатывается управляющий сигнал, который передается всем вершинам УГ, зависящим по управлению от обработанного узла. При поступлении сигнала изменяется состояние соответствующего автомата, и выполняется обработка поступившего значения.

Часть операторов зависит по данным от констант, а, т. к. константа находится в состоянии постоянной готовности (не требует вычисления), то состояние автоматов соответствующих операторов может быть изменено до начала выполнения функции. Такое преобразование является оптимизирующим и оказывает ощутимый эффект в рекурсивных функциях.

В языке Пифагор используется динамическая типизация, поэтому автомат должен учитывать возможность поступления на вход параллельного списка, что значительно усложняет автомат. В качестве примера рассмотрим автомат операции интерпретации (рис. 2).

Состояние *1* является начальным – находясь в нем, автомат ожидает поступления данных или функции. Если поступает сигнал готовности данных – автомат переводится в состояние *2*, в котором он будет находиться, накапливая данные до тех пор, пока не придет сигнал о готовности функции. Аналогичным образом автомат ведет себя, находясь в состоянии *3*, но собирает сигналы о готовности функций. При переходе в состояние *4* готова часть данных и часть функций, поэтому автомат может инициировать начало вычислений (выполнение готовых функций над готовыми данными). Однако переход в состояние *4* не означает того, что были получены все данные и все функции, поэтому автомат по-прежнему собирает сигналы о готовности и выполняет вычисления с поступившими данными и функциями до тех пор, пока не поступят все сигналы.

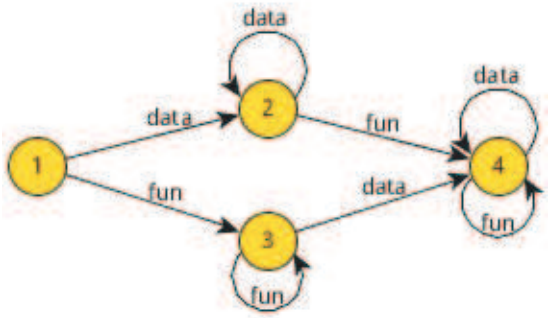

Рис. 2. Автомат операции интерпретации

Если размерности обрабатываемых данных и функций равны единице (на вход не подаются параллельные списки), то автомат может быть упрощен, т. к. в этом случае отсутствует необходимость в накоплении и подсчете поступающих сигналов (рис. 2).

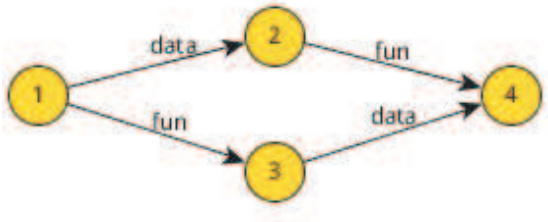

Рис. 3. Упрощенный автомат операции интерпретации

Возможна ситуация, когда на вход оператора поступает параллельный список функций, но все они выполняются над единственным элементом данных, или параллельный список данных обрабатывается единственной функцией. Для таких случаев тоже возможно описать упрощенные версии автоматов.

Замена сложных автоматов более простыми возможна на этапе оптимизации в тех случаях, когда удается точно установить размерность поступающих на вход оператора аргументов. При этом может учитываться то, что размерности аргумента функции, констант, оператора формирования последовательного списка всегда равны единице.

Пример оптимизации параллельных списков. Раскрытие параллельных списков на этапе оптимизации повысит эффективность замены сложных автоматов более простыми, т. к. в ряде случаев удается точно установить невозможность появления параллельного списка в качестве аргумента операции интерпретации.

К таким случаям относится, например, интерпретация списка констант – такая ситуация возникает всякий раз, когда в программе используется ветвление. В качестве примера рассмотрим функцию вычисления модуля числа.

Листинг 4. Вычисление модуля числа

```
test << funcdef X {
({X:-}, X): [((X,0):[<,=>]):?]:.
        >>return
            }
```

Параллельный список, в который помещается результат операции селекции, используется для задания порядка операций и может быть оптимизирован как одноэлементный параллельный список. Оптимизация проводится на реверсивном информационном графе, в котором нет необходимости в списках, задающих порядок. На листинге 5 показано, что за счет введения в программу дополнительных переменных (никак не осложняющих РИГ) возможно избавиться от лишних списков, но делать это вручную неудобно.

Листинг 5. Оптимизация списков, задающих порядок операций

```
test << funcdef X {
  (X, 0) >> pair;
  [<, =>] >> oper;
(pair:oper):? >> selector;
({X:-}, X):selector:. >>return
              }
```

Параллельный список *oper* используется для группировки операций сравнения, которые могут одновременно (параллельно) выполняться над списком *pair*. Размерность этого списка не изменится во время выполнения программы, поэтому возможна его обработка на этапе трансляции.

Листинг 6. Интерпретация параллельных списков

```
test << funcdef X {
  (X, 0) >> pair;
  [<, =>] >> oper;
([pair:<, pair:=>]):? >> selector;
  ({X:-}, X):selector:. >>return
              }
```

В результате интерпретации был добавлен новый параллельный список, в который во время выполнения программы попадут результаты выполнения параллельных операций. Видно, что этот список помещен в список данных, поэтому может быть удален. Кроме того, параллельный список *oper* теперь является не используемым кодом и тоже может быть оптимизирован.

Листинг 7. Оптимизированная функция вычисления модуля числа

```
test << funcdef X { // 1
  (X,0) >> pair; // 2
(pair:<, pair:=>):? >>  selector; // 3
 ({X:-}, X) :selector:. >>return // 4
              } // 5
```

В результате применения такого преобразования:

- удаляется часть параллельных списков;
- автоматы интерпретации могут быть заменены более простыми аналогами;
- в ряде случаев большую эффективность будет иметь оптимизация повторяющегося кода, т. к. операции теперь выполняются над более мелкими элементами.

Алгоритм интерпретации параллельных списков. Оптимизация выполняется согласно следующему алгоритму:

1 Выбрать узел PL параллельного списка, такой, что все его элементы имеют кратность, равную единице (не являются параллельными списками). Если узла PL в РИГ нет – переход к п. 2;

1.1 Найти узел IPL интерпретации узла PL и узел функции интерпретации FPL. Если IPL в РИГ нет – переход к п. 2;

1.2 Получить множество EPLs узлов, дуги которых входят в PL;

1.3 Добавить в РИГ такое множество IEPLs узлов, что узел IE принадлежит IEPLs, если IE является узлом интерпретации узла E, принадлежащего EPLs, с узлом FPL;

1.4 Добавить в РИГ узел LIEPL, являющийся узлом группировки в параллельный список;

1.5 Добавить в РИГ дуги, выходящие из узлов множества IEPLs и входящие в узел LIEPL;

1.6 Получить множество UEPLs дуг, выходящих из узла IPL;

1.7 Для дуг множества UELPs заменить узел-источник данных на LIEPL

1.8 Удалить узел IPL из РИГ;

1.9 Переход к п. 1.1;

2 Конец.

Применение метода оптимизации параллельных списков при проектировании СБИС. В рамках работ по реализации системы высокоуровневого синтеза сверхбольших интегральных схем (СБИС) осуществляется разработка архитектурно-независимого подхода и инструментальных средств поддержки проектирования для создания архитектуры кристалла. При разработке механизма перехода на уровень регистровых передач осуществляется поддержка сквозной верификации проекта с распараллеливанием на уровне операций [5]. Особенности архитектуры цифровых СБИС накладывают ряд ограничений на языковое представление проекта, в частности:

- функции не могут возвращать параллельные или задержанные списки;
- оператор селекции (?) должен применяться к спискам данных, имеющим ровно один ненулевой элемент (в результате этот оператор вернет одноэлементный параллельный список);
- аргументами функции не могут быть другие функции или задержанные списки;
- в программах не должен использоваться оператор преобразования последовательного списка в параллельный (в связи с тем, что он может порождать параллельный список, размер которого будет определен лишь во время выполнения программы).

В настоящее время ведется разработка средств адаптации Пифагор-программ для архитектуры СнК/СБИС [5]. Особенности архитектуры накладывают ряд ограничений на исполняемые программы, в частности:

- функции не могут возвращать параллельные или задержанные списки;
- оператор селекции (?) должен применяться к

спискам данных, имеющим ровно один ненулевой элемент (в результате этот оператор вернет одноэлементный параллельный список);

- аргументами функции не могут быть другие функции или задержанные списки;
- в программах не должен использоваться оператор преобразования последовательного списка в параллельный (в связи с тем, что он может порождать параллельный список, размер которого будет определен лишь во время выполнения программы).

Существующими ограничениями гарантируется невозможность возникновения новых параллельных списков во время работы программы. Все остальные параллельные списки, заданные программистом, могут быть устранены полностью.

В результате раскрытия параллельного списка в соответствии с правилами модели вычислений на месте оператора интерпретации списка создается новый параллельный список, содержащий результаты интерпретации элементов списков-аргументов. Новый список будет расположен в информационном графе «ближе» к возвратной вершине, чем списки-аргументы. Однако параллельный список не может возвращаться функцией, поэтому он гарантированно должен встретить на пути либо другой оператор интерпретации (в этом случае процесс повторяется рекурсивно), либо оператор преобразования параллельного списка в список данных (при этом параллельный список может быть удален в соответствии с эквивалентными преобразованиями, описанными выше).

Если функция вычисления модуля числа (листинг 7) готовилась бы к исполнению на архитектуре СнК/СБИС, то интерпретация оператора селекции возвращала бы не параллельный список, а индекс элемента задержанного списка, код которого должен выполниться. В связи с этим автомат операции интерпретации списка с переменной *selector* мог бы быть упрощен.

Заключение

В статье проведен обзор вариантов оптимизации параллельных списков, основанных на эквивалентных преобразованиях модели вычислений языка программирования Пифагор, в том числе интерпретация параллельных списков.

Показано, что, несмотря увеличение информационного графа на этапе оптимизации в результате раскрытия параллельных списков, программа становится более эффективной, а в ряде случаев появляются дополнительные возможности по оптимизации.

В силу особенностей языка и ограничений архитектуры цифровых СБИС возможно полное устранение параллельных списков, что позволяет заменить все автоматы программы более простыми версиями, которые позволяют отказаться от ожидания к поступлению в качестве аргумента параллельного списка.

Литература

1. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии 2005. Т. 10. № 1. С. 71-89.

2. Легалов А. И. Казаков Ф.А., Кузьмин Д.А., Водяхо А.И.Модель параллельных вычислений функционального языка // Изв. ГЭТУ. 1996. Т. 500. С. 56-63.

3. Васильев В.С. Оптимизация программ функционально-потокового языка Пифагор // Перспективы развития информационных технологий: сб.материалов XX междунар. науч.-практ. конф. Новосибирск: ЦРНС, 2014. С. 7-14.

4. Легалов А.И., Матковский И.В., Кропачева М.С., Удалова Ю.В., Васильев В.М. Технологические аспекты создания, преобразования и выполнения функционально-потоковых параллельных программ // Научный сервис в сети Интернет: все грани параллелизма: тр. междунар. суперкомпьютерной конф. (23-28 сент. 2013 г., г. Новороссийск). М.: Изд-во МГУ, 2013. С. 443-447.

5. Легалов А.И., Непомнящий О.В., Рыженко И.Н. Технология архитектурно-независимого, высокоуровневого синтеза сверхбольших интегральных схем // Докл. Акад. наук высш. шк. Рос. Федерации. 2014. № 1 (22). С. 93-103.

Reference

1. Legalov A.I. Funktsional'nyi yazyk dlya sozdaniya arkhitekturno-nezavisimykh parallel'nykh programm // Vychislitel'nye tekhnologii. 2005. T. 10. № 1. P. 71-89.

2. Legalov A.I., Kazakov F.A., Kuz'min D.A., Vodyakho A.I. Model' parallel'nykh vychislenii funktsional'nogo yazyka // Izv. GETU. 1996. T. 500. P. 56-63.

3. Vasil'ev V.S. Optimization of programs of functional and stream Pythagoras language // Perspektivy razvitiya informatsionnykh tekhnologii: sb.materialov XX mezhdunar. nauch.-prakt. konf. Novosibirsk: TsRNS, 2014. P. 7-14.

4. Legalov A.I., Matkovskii I.V., Kropacheva M.S., Udalova Yu.V., Vasil'ev V.M. Tekhnologicheskie aspekty sozdaniya, preobrazovaniya i vypolneniya funktsional'no-potokovykh parallel'nykh programm // Nauchnyi servis v seti Internet: vse grani parallelizma: tr. mezhdunar. superkomp'yuternoi konf. (23-28 sent. 2013 g. Novorossiisk). M.: Izd-vo MGU, 2013. P. 443-447.

5. Legalov A.I., Nepomnyashchii O.V., Ryzhenko I.N. Tekhnologiya arkhitekturno-nezavisimogo, vysokourovnevogo sinteza sverkhbol'shikh integral'nykh skhem // Dokl. Akad. nauk vyssh. shk. Ros. Federatsii. 2014. № 1 (22). P. 93-103.