УДК 004.272.43

МЕТОДЫ АВТОМАТИЧЕСКОГО ОПРЕДЕЛЕНИЯ ЭФФЕКТИВНОСТИ РЕАЛИЗАЦИИ ФРАГМЕНТОВ ПРИКЛАДНЫХ ПРОГРАММ НА ЯЗЫКЕ COLAMO

А. И. Дордопуло[1], И. И. Левин[1], В. А. Гудков[1]

Рассматривается методика выявления неэффективного использования конструкций языка COLAMO и неэффективной организации потоков данных, приводящих к снижению реальной и удельной производительности реконфигурируемой вычислительной системы. Формулируются рекомендации по их устранению. Статья рекомендована к публикации программным комитетом Международной научной конференции "Научный сервис в сети Интернет: поиск новых решений" (http://agora.guru.ru/abrau).

Ключевые слова: реконфигурируемые вычислительные системы, производительность, параллельная программа, эффективность.

1. Введение. Одной из основных технических характеристик вычислительной системы (ВС) является ее производительность, под которой понимается число одновременно выполняемых операций над данными определенной разрядности и формы представления в единицу времени. Для оценки эффективности вычислительной системы и прикладных программ используются понятия пиковой, реальной и удельной производительности.

Под пиковой производительностью понимают теоретически возможное число одновременно выполняемых операций в вычислительной системе, что характеризует конструктивные и технологические потенциальные возможности вычислительной системы по обработке данных. Пиковая производительность является верхней оценкой достижимой производительности ВС. Реальная производительность, под которой понимается число одновременно выполняемых операций в единицу времени при решении прикладной задачи, в большей степени является показателем соответствия архитектуры системы структуре решаемой задачи. Удельная производительность определяется как отношение реальной производительности прикладной программы к занимаемому аппаратному ресурсу и характеризует эффективность реализации прикладной задачи на ВС.

Для кластерных ВС высокие показатели реальной производительности достигаются, как правило, при решении слабосвязанных задач, у которых число информационных обменов между фрагментами задачи при ее решении сравнительно невелико. При решении сильносвязанных задач с интенсивными информационными обменами производительность кластерных ВС не превышает 10% от пиковой производительности.

Такой существенный разрыв между декларируемой пиковой и реальной производительностью кластерных ВС обусловливает применение реконфигурируемых вычислительных систем (РВС) [1] для решения сильносвязанных задач. Для ряда предметных областей РВС демонстрируют высокие показатели реальной производительности, составляющие не менее 60% от пиковой производительности системы. Для достижения таких высоких показателей производительности эффективность реализации прикладных задач на РВС является основополагающим фактором.

Как правило, причиной снижения производительности РВС являются неэффективные параллельные программы. Под неэффективной параллельной программой на языке высокого уровня COLAMO будем понимать программу, содержащую неэффективно используемые конструкции языка и/или неэффективно организованные потоки данных, приводящие к снижению степени распараллеливания программы или к дополнительным аппаратным затратам.

2. Методика выявления неэффективных фрагментов программы. Для достижения высокой реальной и удельной производительности РВС предлагается методика, направленная на выявление неэффективного использования конструкций языка и выдачи рекомендаций по их устранению, состоящая из следующих этапов:

[1] Научно-исследовательский институт многопроцессорных вычислительных систем им. академика А. В. Каляева, Южный федеральный университет (НИИ МВС ЮФУ), ул. Чехова, 2, ГСП-284, 347928, г. Таганрог; А. И. Дордопуло, ст. науч. сотр., e-mail: scorpio@mvs.tsure.ru; И. И. Левин, зам. директора по науке, e-mail: Levin@mvs.tsure.ru; В. А. Гудков, ст. науч. сотр., e-mail: Slava_Gudkov@Mail.ru

1) найти несбалансированные вычислительные структуры фрагментов параллельной программы;
2) найти обратные связи и определить их параметры;
3) найти косвенную адресацию при обращении к элементам массивов;
4) найти неэффективное использование условных операторов в тексте параллельной программы;
5) найти неэффективные записи арифметических выражений;
6) найти неэффективную организацию доступа к элементам потоковых массивов.

На каждом этапе выполнения методики выдаются предупреждения о неэффективном использовании конструкций языка или рекомендации по их устранению.

Несбалансированность вычислительной структуры [2], когда темп обработки входной информации (на всех входах структуры) отличается от темпа выдачи информации, является одной из основных причин снижения степени распараллеливания программы.

Степень распараллеливания параллельной программы в языке COLAMO [3, 4] определяется неявным образом при описании типа доступа к элементам массивов: параллельный (определяется ключевым словом `Vector`) и последовательный (определяется ключевым словом `Stream`) [5]. Тип доступа `Vector` указывает на необходимость размещения элементов массива в разных каналах памяти, доступ к которым может выполняться параллельно, а тип `Stream` указывает на то, что элементы массива располагаются в одном канале памяти, доступ к которым выполняется последовательно. Изменение типа доступа позволяет достаточно просто управлять как степенью распараллеливания программы на уровне описания структур данных, так и скоростью обработки и занимаемым ресурсом.

Снижение степени распараллеливания программы наиболее часто встречается при использовании в вычислительном фрагменте массивов с различными типами доступа, для которых при синтезе вычислительной структуры прикладной программы транслятором языка высокого уровня COLAMO осуществляется согласование скорости обработки потоков данных, что в случае несбалансированной вычислительной структуры, описанной программистом, приводит к снижению скорости обработки до скорости самого медленного потока и непродуктивным затратам дополнительных аппаратных ресурсов на коммутацию информационных потоков данных и смещение их относительно друг друга.

Рассмотрим пример сочетания различных типов переменных в едином вычислительном контуре.

```
Var a : Array Integer [10:Vector] Mem;
Var c : Array Integer [10:Stream] Mem;
Var I : Number;
Const k = 9;
Cadr summa;
    For i:=0 To k Do
        c[i] :=a[i];
Endcadr;
```

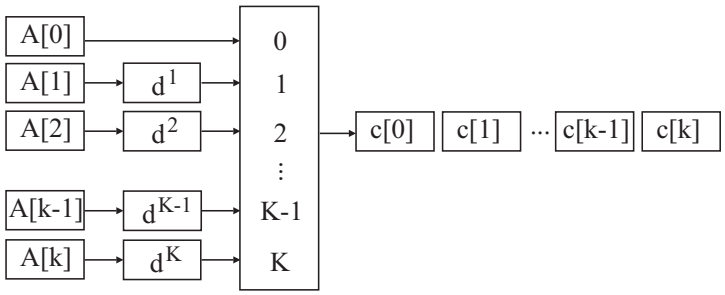

Рис. 1. Программа и эквивалентная ей вычислительная структура со степенью распараллеливания 10

В программе на рис. 1 показано присвоение потоковому массиву C элементов векторного массива A, что приводит к несбалансированной вычислительной структуре и требует согласования потоков данных. Для корректной организации потоков данных в несбалансированной вычислительной структуре (рис. 1) используются блок согласования (мультиплексор MX) и блоки временны́х задержек (d^i — блок задержки на i отсчетов). Такая запись присваивания векторного массива к потоковому массиву приводит к синтезу несбалансированной структуры, падению реальной и удельной производительности вычислительной системы за счет снижения скорости обработки потоков данных и использования дополнительного оборудования.

Для рассматриваемого примера программисту необходимо изменить типы доступа к элементам используемых массивов параллельной программы или ввести дополнительные коммутационные переменные для организации единой степени распараллеливания для всех массивов в пределах оператора.

Несбалансированная вычислительная структура может синтезироваться также и в том случае, когда входящие в единый фрагмент параллельной программы вычислительные операторы по отдельности являются сбалансированными, но между собой не согласованы (одним из наиболее наглядных примеров является полный условный оператор, содержащий логическое выражение в условии, и вычислительные операторы в прямой и альтернативной ветвях, поскольку условные операторы более эффективно реализуются параллельно в виде множества независимых процессов).

Несбалансированность вычислительных структур прикладной параллельной программы может быть вызвана неэффективной организацией вычислений, большинство проявлений которой связано с организацией рекурсивных вычислений и неэффективным использованием распределенной памяти РВС.

Рекурсивные вычисления, под которыми понимается вычисление одного элемента массива на основе значения предыдущего (ранее вычисленного) элемента этого же массива, при структурной реализации представляют собой замкнутые вычислительные контуры (циклы) с отличным от других фрагментов темпом подачи данных на вход контура — скважностью. Скважность представляет собой натуральное число и является отношением количества тактов работы вычислительной структуры к количеству данных, которые можно подать на вход вычислительной структуры за это количество тактов, т.е. характеризует временной интервал в тактах, после которого можно подать на вход вычислительной структуры следующее данное. Для линейных вычислительных операторов скважность равна 1, при наличии в вычислительном контуре петель обратной связи скважность, как правило, увеличивается, а темп подачи соответственно снижается, что приводит к снижению реальной производительности. Скважность можно определить по формуле

$$s = \left\rceil \frac{L}{N} \right\lfloor, \tag{1}$$

где L — общая латентность вычислительного контура и N — значение временно́й задержки в цепи обратной связи.

Если в вычислительной структуре фрагмента параллельной программы существуют несколько циклов, то для вычисления скважности S необходимо выполнить следующие действия: выполнить обход информационного графа и найти все циклы; для каждого цикла рассчитать его скважность s_i; определить общую скважность S для всех найденных циклов в информационном графе по правилу $S = \max\limits_{i=1,\ldots,m}(s_i)$, где m — количество найденных циклов. Если вычисленная скважность $S \neq 1$, то темп подачи данных при синтезе вычислительной структуры будет снижен в S раз.

```
Var a,c,d,e : array Integer [100 : Stream] Mem;
Var b : array Integer [100 : Stream] Com;
Var i, n : Number;
Define n=70;
Cadr Example;
    For i := 30 To n Do
        Begin
            b[i] := b[i - 10] + a[i] - c[i];
            e[i] := b[i] + d[i];
        End;
Endcadr;
```

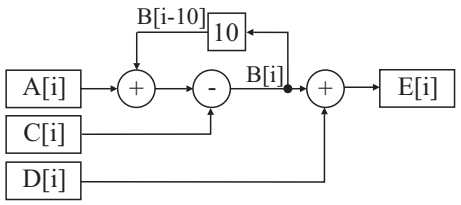

Рис. 2. Программа и эквивалентная ей вычислительная структура с обратной связью

Рассмотрим пример программы, содержащей фрагмент с обратной связью, и ее вычислительную структуру на рис. 2. Общая латентность кольца (L) в вычислительной структуре складывается из латентности сумматора (равна 22), вычитателя (равна 21) и элемента задержки (латентность равна 10); в результате имеем $L = 63$. Значение временно́й задержки N между элементами, образующими обратную связь, определяется временно́й задержкой, установленной для организации обращения к элементам b[i] и b[i-10], и равно 10. Следовательно, согласно формуле (1), скважность в вычислительной структуре равна 6.

Признаком неэффективного использования распределенной памяти РВС является обращение к ней по произвольному адресу. Обращение к распределенной памяти по произвольному адресу (косвенная адресация) используется в прикладных задачах для доступа к массивам данных или табличным значениям параметров задачи. Признаками использования косвенной адресации при обращении к элементам массива может быть использование массивов, функций или индексных переменных.

Использование косвенной адресации в одном из каналов памяти ведет к снижению темпа подачи данных для этого канала памяти, что замедляет темп подачи данных для всех каналов памяти независимо от того, какой тип адресации для них используется. На рис. 3 показаны параллельная программа с использованием косвенной адресации и эквивалентная ей вычислительная структура. Обращение к переменной A является примером ярко выраженной косвенной адресации, так как адрес записи данных переменной A определяется значением элемента массива d[i].

Использование нескольких индексных переменных при обращении к элементам массива, показанное на рис. 4, также расценивается как косвенная адресация, так как использование нескольких индексных

```
Var a,b,c,d : Array Real [10 : Stream] Mem;
Var i : Number;
Cadr ExampleIndexConst;
    For i := 0 to 9 do
        Begin
            a[d[i]] := b[i] + c[i];
    End;
EndCadr;
```

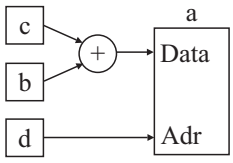

Рис. 3. Программа с использованием косвенной адресации и эквивалентная ей вычислительная структура

```
Var a,b,c,d : Array Real [100 : Stream] Mem;
Var i, j : Number;
Cadr ExampleTwoIndex;
    For i := 2 to 9 do
        For j := 5 to 90 do
            Begin
                a[i * j] := b[j] + c[j - 1];
            End;
EndCadr;
```

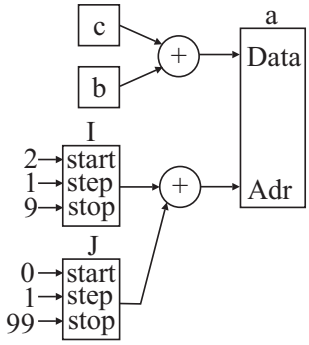

Рис. 4. Использование нескольких индексных переменных

переменных не позволяет однозначно определить шаг изменения адреса памяти на каждой итерации циклов, поскольку этот шаг в общем случае не является величиной постоянной, хотя и обладает некоторой закономерностью изменений.

Автоматическое выявление косвенной адресации при обращении к элементам массивов в параллельной программе при трансляции позволяет выдать программисту предупреждения об использовании косвенной адресации при обращении к элементам массивов.

Одной из важнейших конструкций языка COLAMO, неэффективное использование которой может привести к снижению реальной и удельной производительности вычислительной системы, является условный оператор. Синтезируемая для условного оператора вычислительная структура зависит от способа хранения переменных, используемых в условном выражении. Рассмотрим полный условный оператор, содержащий как прямую (`Then`), так и альтернативную (`Else`) ветви, каждая из которых содержит один оператор присваивания. Информационный граф для структурной реализации условного оператора представлен на рис. 5.

If α Then
$a_i = F_1(c_1, c_2, \ldots, c_n)$
Else
$a_j = F_2(d_1, d_2, \ldots, d_m)$

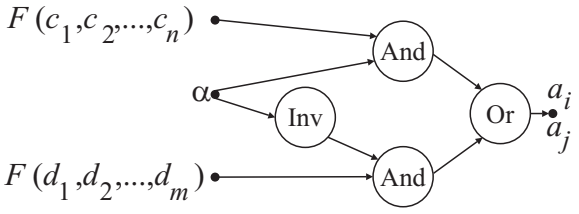

Рис. 5. Информационный граф структурной реализации условного оператора

Для реализации условного оператора требуются два блока логического умножения (`And`), один блок инверсии (`Inv`) и один блок логического сложения (`Or`). Наиболее часто при структурной реализации условного оператора неэффективные затраты оборудования встречаются в следующих случаях:

1) если в одной из ветвей коммутационной переменной присваивается значение, равное “0”;

2) если условный оператор является неполным (т.е. отсутствует альтернативная ветка);

3) если в ветвях условного оператора используются разные коммутационные переменные в качестве результата присваивания.

Рассмотрим первый случай, информационный граф которого представлен на рис. 6.

Из информационного графа (рис. 6) видно, что в случае невыполнения условия коммутационной переменной будет присвоено значение “0”. Однако согласно особенностям коммутационной переменной, если в нее не выполняется запись, ее значение всегда будет иметь значение “0”, в том числе и в случае

If α Then
$a_i = F_1(c_1, c_2, \ldots, c_n)$
Else
$a_j = 0$

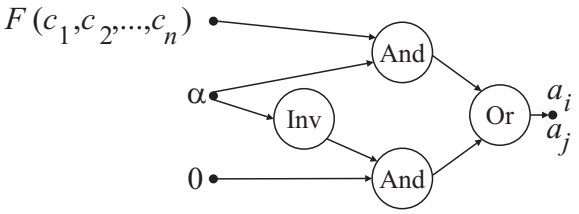

Рис. 6. Информационный граф условного оператора

невыполнения условия α. Поэтому блок инверсии и блок логического умножения можно не использовать, а просто подать константу “0” на выход.

Удаление альтернативной ветви `Else` из программы на рис. 6 не приведет к сокращению аппаратных затрат, поскольку структурная схема, реализующая условный оператор, всегда включает в себя реализацию обеих ветвей условного оператора и аппаратные затраты на реализацию обеих ветвей (рис. 7).

If α Then
$a_i = F_1(c_1, c_2, \ldots, c_n)$

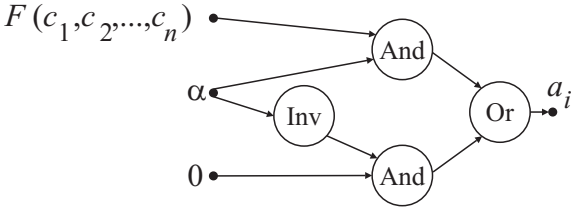

Рис. 7. Неполный условный оператор

При отсутствии альтернативной ветви `Else` на вход блока логического суммирования будет подаваться значение “0”, что приводит к нерациональному использованию оборудования (блок инверсии и блок логического умножения), как и в первом случае (рис. 6).

Рассмотрим третий вариант использования условного оператора на примере программы на рис. 8.

If α Then
$a_i = F_1(c_1, c_2, \ldots, c_n)$
Else
$b_j = F_2(d_1, d_2, \ldots, d_m)$

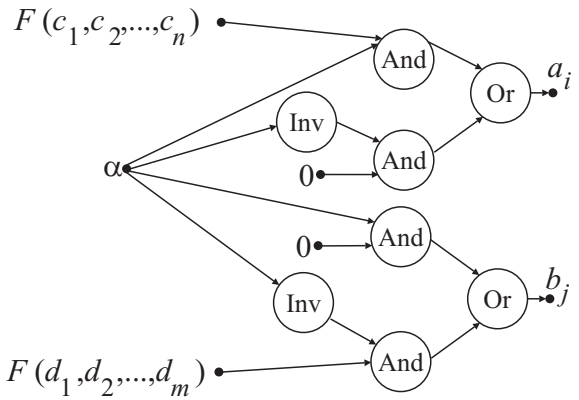

Рис. 8. Программа и эквивалентная ей вычислительная структура

Использование в ветвях условного оператора разных переменных в результате присваивания приводит к структурной реализации условного оператора для обеих коммутационных переменных, расположенных в ветвях условного оператора. Из вычислительной структуры на рис. 8 видно, что для вычисления массива a на вход блока логического суммирования подается значение “0”, что соответствует отсутствию альтернативной ветви `Else`. Для вычисления массива b на вход блока логического суммирования также подается значение “0”, так как в операторе условного оператора в прямой ветви отсутствует присваивание переменной b.

Для эффективной структурной реализации программ в случаях, показанных на рис. 6–8, рекомендуется условный оператор привести к логическому умножению, существенно сократив аппаратные затраты. Для организации выбора значения коммутационной переменной по условию программа должна иметь вид, показанный на рис. 9.

Эффективность параллельной программы зависит не только от грамотного использования условных операторов, но и от эффективного использования арифметических операторов. Для арифметических выражений наиболее распространенным примером неэффективной реализации является их повторная за-

$$a_i = F_1(c_1, c_2, \ldots, c_n)\&\alpha$$

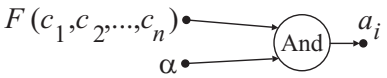

Рис. 9. Программа и эквивалентная ей вычислительная структура

пись, которая иллюстрируется в программе 1.4.

```
Var a,b,c,d,e,r : Array Integer [100 : Stream] Mem;
Var i, j, n : Number ;
Define n=99 ;
Cadr EqualPart;
    For i := 2 to n do
        Begin
            a[i] := F1(b[i], c[i]) - d[i] + e[i];
            r[i] := F1(b[i], c[i]) + (d[i] * e[i]);
        End;
Endcadr;
```

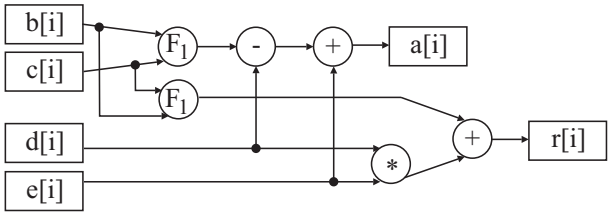

Рис. 10. Программа (1.4) и ее вычислительная структура

Согласно принципам языка COLAMO в вычислительной структуре (рис. 10) программы (1.4) будет реализовано две функции F_1(b[i], c[i]), несмотря на то что они одинаковы, а это приводит к необоснованным затратам оборудования.

Для эффективной реализации программы (1.4) необходимо использовать коммутационную переменную, которой будет присвоено выражение F_1(b[i], c[i]), а сами выражения в операторах должны быть заменены на коммутационную переменную. После выполнения этих преобразований программа (1.4) примет вид (1.5).

```
Var a,b,c,d,e,r : Array Integer [100 : Stream] Mem;
Var t : Integer Com;
Var i, j, n : Number ;
Define n=99 ;
Cadr EqualPart;
    For i := 2 to n do
        Begin
            t := F1(b[i], c[i]);
            a[i] := t - d[i] + e[i];
            r[i] := t + (d[i] * e[i]);
        End;
Endcadr;
```

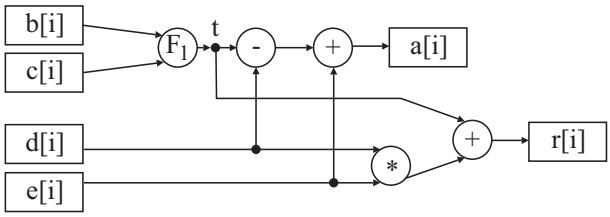

Рис. 11. Программа (1.5) и ее вычислительная структура

Поскольку в операторах программы (1.5) используется только один вызов функции F_1, то и в вычислительной структуре (рис. 11) будет реализована только одна функция, что позволяет сократить аппаратный ресурс, необходимый для реализации задачи.

Таким образом, если в операторах параллельной программы найдены общие подвыражения, то выдается рекомендация о необходимости использования коммутационных переменных в качестве промежуточных переменных между операторами и выделенными выражениями.

Зачастую в операторах параллельной программы используется параллельный доступ к разным элементам одного и того же потокового массива. Однако элементы потокового массива могут обрабатываться только последовательно, поэтому для организации одновременного доступа к разным элементам массива необходимо использовать смещение на уровне информационных потоков данных (рис. 12).

В программе на рис. 12 выполняются три одновременных доступа к потоковому массиву C. Для корректной организации одновременного доступа необходимо использовать две временны́е задержки (смещения) d^5 и d^8. При этом суммарную величину временны́х задержек, необходимых для организации потоков данных, можно сократить за счет использования коммутационных массивов и, тем самым, сократить аппаратные затраты.

Использование коммутационных массивов позволяет организовать одновременный доступ к элементам массива c[i], c[i-5], c[i-8], но при этом временны́е задержки, необходимые для выравнивания потока данных, будут организованы не параллельно, как показано на рис. 12, а последовательно (рис. 13).

Количество коммутационных массивов, реализующих смещение, определяется величиной $n-1$, где n — число одновременных обращений к массиву c[i] (рис. 12), а величина временно́й задержки между элементами коммутационных массивов рассчитывается на основании смещений между элементами этого

```
Var a,b : Array Real [100 : Stream] Mem;
Var c : Array Real [100 : Stream] Com;
Var I : Number;
Cadr ExampleF;
    For i := 10 to 70 do
        Begin
            c[i] := b[i];
            a[i] := F (c[i], c[i-5], c[i-8]);
        End;
Endcadr;
```

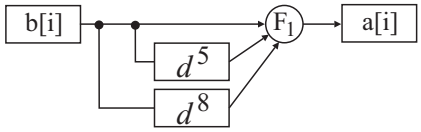

Рис. 12. Параллельная программа со смещением при адресации массивов и эквивалентная ей вычислительная структура

```
Const n = 99;
Var a,b : Array Real [100 : Stream] Mem;
Var c,d,e : Array Real [100 : Stream]
Mem;
Var I : Number;
Cadr Example;
    For i := 0 to n do
        Begin
            c[i] := b[i];
            d[i] := c[i-5];
            e[i] := d[i-3];
            a[i] := F (c[i], d[i], e[i]);
        End;
Endcadr;
```

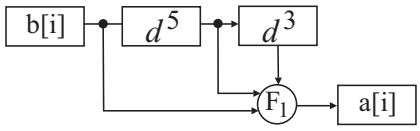

Рис. 13. Программа (рис. 12) с использованием коммутационных массивов и эквивалентная ей вычислительная структура

массива. При этом суммарная величина полученных временны́х задержек не должна превышать значения максимальной временно́й задержки.

Из вычислительных структур на рис. 12 и 13 видно, что суммарная величина временно́й задержки между элементами на рис. 12 складывается из величин d^5 и d^8 и равна 13, а на рис. 13 суммарная величина временно́й задержки складывается из величин d^5 и d^3 и равна 8.

Таким образом, при выявлении параллельной организации доступа к элементам потокового массива (рис. 12) необходимо перейти к последовательной организации доступа к элементам массива за счет использования промежуточных коммутационных массивов.

3. Заключение. Рассмотренная методика, направленная на выявление неэффективного использования конструкций языка и выдачи рекомендаций по их устранению, позволит создавать эффективные параллельные прикладные программы и, как следствие, повысит реальную и удельную производительность реконфигурируемых вычислительных систем.

Работа выполнена при финансовой поддержке Минобрнауки России по государственному контракту от 24.12.2010 г. № 07.514.12.4001 в рамках ФЦП “Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007–2013 годы”.

СПИСОК ЛИТЕРАТУРЫ

1. *Дордопуло А.И., Каляев И.А., Левин И.И., Семерников Е.А.* Высокопроизводительные реконфигурируемые вычислительные системы нового поколения // Вычислительные методы и программирование. 2011. **12**, № 2. 240–247.
2. *Самофалов К.Г., Луцкий Г.М.* Основы теории многоуровневых конвейерных вычислительных систем. Москва: Радио и связь, 1989.
3. *Каляев А.В., Левин И.И.* Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений. М.: Янус-К, 2003.
4. *Левин И.И., Гузик В.Ф., Сафронов О.О.* Язык программирования высокого уровня для многопроцессорной системы с программируемой архитектурой // Распределенная обработка информации. Новосибирск, 1991.
5. *Каляев И.А., Левин И.И., Семерников Е.А., Шмойлов В.И.* Реконфигурируемые мультиконвейерные вычислительные структуры. Изд. 2-е, перераб. и доп. Ростов-на-Дону: Изд-во ЮНЦ РАН, 2009.

Поступила в редакцию
16.10.2012