Р. Н. Арапбаев, А. П. Стасенко

ИНДЕКСНЫЙ АНАЛИЗ ЗАВИСИМОСТЕЙ ПО ДАННЫМ В SISAL-ПРОГРАММАХ

1. ВВЕДЕНИЕ

Функциональные языки и языки однократного присваивания, каковым является Sisal, способствуют разработке корректных детерминированных параллельных программ. Функциональные программы свободны от совмещения имен, сторонних эффектов и ошибок, зависящих от времени. Результаты детерминированы, невзирая на архитектуру, операционную систему или исполняемое окружение. В отличие от императивных языков функциональные языки уменьшают нагрузку на программирование. Пользователи могут определить, что может быть вычислено, и могут только кодировать зависимости по данным между операциями.

Компилятор ответственен за планирование операций, передачу значений данных, синхронизирующих операций, управление памятью. Легко написать функциональную программу, которая является, безусловно, параллельной, так как ее можно писать как последовательные императивные программы. Освобожденный от большинства сложностей параллельного программирования пользователь получает больше возможностей сконцентрироваться на конструкции алгоритмов и разработки прикладных программ [1].

В Лаборатории конструирования и оптимизации программ ИСИ СО РАН, в рамках проекта ПРОГРЕСС [2], создается система функционального программирования SFP [3], где в качестве начальной версии входного языка выбран Sisal 3.2 [4]. В системе SFP Sisal-программы преобразуются в специально разработанные внутренние графовые представления [5, 6], на которых проводятся различные оптимизирующие преобразования [7], производится интерпретация программ, и которые транслируются в программы на языке Си.

Возможности любого компилятора зависят от возможностей его анализаторов. Данная работа посвящена построению (разработке) алгоритма для индексного анализа зависимости по данным в Sisal-программах.

Статья построена следующим образом. В разделе 2 даны основные определения. В разделе 3 подробно описывается построение алгоритма зави-

симости по данным в Sisal-программах. В разделе 3 приводится заключение о проделанной работе и список литературы по данной тематике.

Все понятия, не определяемые в этой работе, могут быть найдены в работах[4–6, 8].

2. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Традиционно под зависимостью по данным понимается зависимость, связанная с совпадением ссылок на элементы массивов. Определение зависимостей по данным представляет собой задачу целочисленного программирования [8]. Собранная при анализе зависимостей информация суммируется и представляется в компактном виде, удобном для дальнейшего использования. Чаще всего эта информация представляется в виде ориентированного графа зависимостей по данным (ГЗД). Известны также и другие способы представления: векторы расстояний и направлений, конус зависимости и др.

Для представления Sisal-программ используется представление программы в виде иерархического ациклического потокового графа (например, IF1-представления [5]). В системе SFP внутреннее представление Sisal-программ (будем называть его IR1-представлением [6]) также реализовано как набор классов языка Си++, реализующих средства для работы с IF1-графами.

2.1. Промежуточное представление IR1

Промежуточное представление IR1 [6], основывающееся на ациклических иерархических орграфах, было разработано для описания модели вычислений языка Sisal 3.0. При разработке языка IR1 основной задачей являлось создание графового языка, который мог бы стать результатом трансляции (front-end) компиляторов нескольких функциональных языков. Это иерархическая граф-модель, явно описывающая зависимости по данным на самом нижнем уровне и неявно – между так называемыми «составными вершинами», представляющими управляющие структуры, и их подграфами. С помощью такого графового языка становится возможным объединение и единые преобразования для программ, написанных с помощью разных функциональных языков.

Программа на языке IR1 состоит из множества графов специального вида, среди которых выделено подмножество графов, соответствующих множеству функций исходной Sisal программы. Граф в IR1 состоит из объектов

трёх видов: вершин, дуг и рамки графа, причём вершинам и рамке графа приписаны упорядоченные множества входов (в которые входят дуги, но не более одной для одного входа) и выходов (из которых выходят дуги). Входы рамки графа рассматриваются как выходы (результаты вычислений) графа, а выходы рамки графа как входы (параметры) графа. Тем самым, рамку графа можно рассматривать как «вывернутую наизнанку» вершину. В дальнейшем, как и в оригинальной терминологии IR1, входы могут называться входными портами, а выходы – выходными портами.

Каждой дуге графа приписан тип пересылаемого значения, если IR1 задаёт строго типизированный язык. Существует, однако, специальный вид дуги, обозначающий литерал любого типа. Эти дуги не имеют начального выхода, но имеют дополнительное свойство, передающее непосредственное значение литерала в какой-либо входной порт.

Вершины обозначают операции над своими входами (аргументами), результаты которых находятся на выходах вершины. Вершины бывают простыми и составными. Простые вершины (или просто вершины) не имеют внутренней структуры помимо ассоциированной с ними операции. Составные вершины дополнительно содержат упорядоченное множество графов. Их количество и все связи между входами (выходами) составной вершины и входами (выходами) этих графов неявно задаются типом (или семантикой) операции составной вершины. Структурированность вычислений, задаваемых IR1 графом, основывается на иерархичности IR1 графов, заданной посредством графов составной вершины.

Тем самым, IR1 задаёт поток вычислений без какого-либо дополнительного управления и исключительных ситуаций. Поэтому предполагается наличие дополнительного, ошибочного значения для каждого используемого типа данных, возвращаемого вершинами операций, определённых не для всех значений своих аргументов (например, деление на ноль).

Очевидно, что вычисления, заданные подобным образом, определяют частичный порядок над общей последовательностью выполнения вершин-операций одного IR1 графа. Несравнимые между собой операции можно выполнить параллельно, что позволяет естественным (не зависящим от машинной архитектуры) образом задать модель параллельных вычислений, причём на очень низком уровне – уровне отдельной операции. К тому же IR1 граф допускает лёгкую интерпретируемость (исполнение), так как он задаёт потоковую модель вычислений, для исполнения которой (с некоторыми ограничениями) можно даже использовать суперкомпьютеры с потоковой архитектурой. В языке Sisal большую важность имеет уменьшение

излишнего копирования значений (особенно в случае больших массивов), возникающее при выходе нескольких дуг из одного порта.

Ниже приведен пример Sisal-функции (см. пример 1) и её внутреннего представления (см. рис.1). Отметим, что составная вершина gtForAll имеет четыре графа [6]:

1. Граф инициализации.
2. Граф генератора диапазона.
3. Граф тела цикла.
4. Граф предложения возврата.

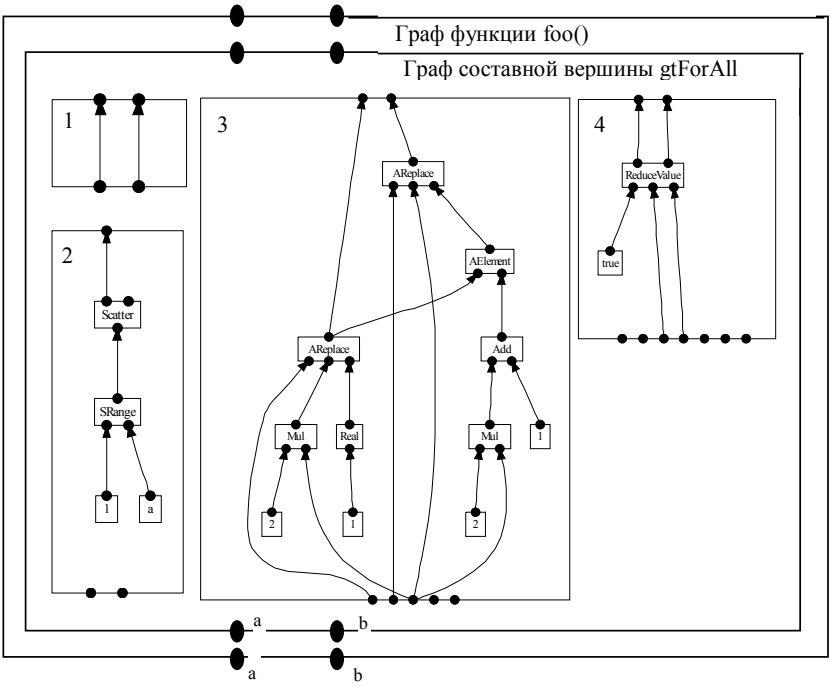

Рис. 1. IR1-граф для примера 1

В этом примере в третьем графе вершины «gtForAll» имеется дублирование массива «a» в двух дугах, исходящих из выходного порта вершины gtAReplace.

Пример 1. Sisal-функция.

```
function foo(a,b: array[real] returns array[real],array[real])
 for i in 1, 10 repeat
  a:= old a[2*i := 1];
  b:= old b[i := a[2*i+1]]
  returns value of a, b
 end for
end function
```

2.2. Определение зависимости по данным

Пусть две операции S_1 и S_2 обращаются к элементу d - мерного массива A и r – число for-циклов («выражений for» в SISAL терминологии) окружающих операции S_1 и S_2: $S_1 \equiv A[f_1(i_1, i_2, \dots, i_r), f_2(i_1, i_2, ..., i_r), \dots, f_d(i_1, i_2,\dots, i_r)]$ и $S_2 \equiv A[g_1(i_1, i_2,..., i_r), g_2(i_1, i_2,\dots, i_r), \dots, g_d(i_1, i_2,\dots, i_r)]$, где $i_1, i_2, ..., i_r$ индексные переменные циклов. Нижние границы $L_1, L_2, ..., L_r$ и верхние границы циклов $U_1, U_2, ..., U_r$ являются целочисленными константами.

Определение 1. Назовем две операции S_1 и S_2 в цикле *зависимыми*, если:

- S_1 и S_2 обращаются к общему участку памяти M;
- S_1 исполняется раньше S_2 в цикле;
- между S_1 и S_2 нет других обращений к участку памяти.

Определение 2. Под равенством $S_1(i) = S_2(j)$ понимается, что данные операции обращаются к общему участку памяти.

Таким образом, система уравнений зависимости по данным для операции S_1 и S_2 имеет вид:

$$\begin{cases} f_1(i_1, i_2, ..., i_r) = g_1(i_{r+1}, i_{r+2}, ..., i_n) \\ f_2(i_1, i_2, ..., i_r) = g_2(i_{r+1}, i_{r+2}, ..., i_n) \\ \quad\vdots \qquad\quad \vdots \qquad\quad \vdots \\ f_d(i_1, i_2, ..., i_r) = g_d(i_{r+1}, i_{r+2}, ..., i_n) \end{cases} \tag{1}$$

$$L_p \le i_p \le U_p, \text{ где } p=1,\dots,n. \tag{2}$$

Между двумя операции S_1 и S_2 имеется *зависимость по данным* тогда, и только тогда, когда существуют целочисленные решения $i_1, i_2..., i_n$ системы линейных диофантовых уравнений (1), удовлетворяющие ограничениям (2).

Следовательно, проблема зависимости данным представляет собой задачу целочисленного программирования. Если имеется *m*-мерный массив **A** и индексные выражения массива линейны, то тогда система (1) может быть записана в следующем виде:

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + c_1 = 0 \\ &a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + c_2 = 0 \\ &\qquad \ldots\ldots\ldots\ldots \\ &a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n + c_m = 0 \end{aligned} \qquad (3)$$

и

$$L_i \le x_i \le U_i \text{ где } i = 1, \ldots, n \qquad (4)$$

Определение 2. Будем также говорить, что индексные выражения *сцепленные* (coupled), если они включают в себя одинаковые индексные переменные цикла.

Если потенциальная зависимость включает *сцепленные* индексы, то для её разрушения необходимо одновременное рассмотрение индексов многомерного массива.

При решении этой задачи компиляторы используют тесты на зависимость по данным. Однако прямой подход к решению задачи выявления зависимостей в общем случае невозможен, так как даже для линейных индексных выражений массивов это приводит к NP-полной проблеме отыскания целочисленного решения системы диофантовых уравнений (1) и неравенств (2). Поэтому к настоящему времени разработаны многие тесты, дающие приближенные решения задачи. Среди них на практике наибольшее распространение получили НОД-тест, неравенства Банержи, λ-тест, I-тест и их различные модификации. Точные тесты используют наиболее сложные методы, например: Power-тест, Омега-тест и IR-тест.

Тесты на зависимость используют различные математические инструменты, и каждый из них имеет различную сложность и разрешающую способность. Мощные алгоритмы могут выявлять зависимости по данным с большей точностью, но обычно требуют для этого много времени. Поэтому на практике используется алгоритм зависимости по данным, который состоит из серии тестов исполняемых в определенном иерархическом порядке [9].

3. ПОСТРОЕНИЕ АЛГОРИТМА ИНДЕКСНОГО АНАЛИЗА ЗАВИСИМОСТЕЙ ПО ДАННЫМ

В данном разделе будет описано построение алгоритма индексного анализа зависимостей по данным для Sisal-программ. В этой работе применяется алгоритм индексного анализа на основе системы линейных неравенств [10]. Для формирования этой системы требуется выполнить подготовительные действия по поиску гнезд циклов и индуктивных переменных, нахождению инвариантов цикла, делинеаризации и непосредственному построению системы неравенств. На вход анализатора подается внутреннее представление IR1 анализируемой Sisal-программы. Далее подробно рассматривается каждый этап анализа зависимостей по данным.

3.1. Поиск гнезд циклов, для которых возможен индексный анализ

Первым шагом анализатора является поиск гнезд циклов с генератором диапазона (gtForAll-вершина в терминологии IR1 графа [6]), для которых возможен индексный анализ. Отметим, гнездом называется путь в дереве циклов от корня до одного из листьев, т.е. гнездом циклов является множество циклов, вложенных один в другой. В данной работе рассматриваются только «совершенное» гнездо циклов. Гнездо называется совершенным, если тело каждого следующего цикла, отличного от самого внутреннего, состоит только из следующего цикла гнезда [11].

Поиск реализуется обходом IR1 графа в глубину. Как только находится гнездо for- циклов, далее выполняется шаг подготовки данных гнезда циклов.

3.2. Поиск индексных переменных и подготовка данных гнезда циклов

Диапазонные имена или данные индексных переменных определяются в генераторе диапазона, т.е. находятся на втором графе (граф генератора диапазона) составной вершины gtForAll. При обработке графа генератора диапазона создается матрица LUI содержащая значений границ циклов.

Для правильной проверки на зависимости по данным границы индексов должны быть известными константами, а шаг индексных переменных равным 1.

3.3. Подготовка данных для анализа существования зависимости двух операций обращения к массиву в цикле

На этом этапе входными данными являются операции, для которых будет производиться анализ, и цикл, которому принадлежат данные операции и соответствующее гнездо циклов с уже подготовленными данными. Так как рассматривается только «совершенное» гнездо циклов, кандидаты для зависимости по данным пар операций находятся, только в самом внутреннем теле цикла. По терминологии IR1 это граф тела цикла (третий граф) самой внутренней составной вершины gtForAll.

В представлении IR1 индексные зависимости могут возникать между вершиной gtAReplace (операцией замещения элементов массива) и вершиной gtAElement (операцией выборки элементов массива), если они связаны следующим образом:

1) выходной порт вершины A, которая является вершиной gtAReplace, связан дугой E с первым входным портом вершины B;
2) в первые входные порты этих вершин входят дуги E_1 и E_2, начинающиеся из одного порта.

В первом случае, когда вершина B является вершиной gtAElement, независимость операций A и B позволяет переключить начало дуги E на порт, с которого начинается дуга, входящая в первый порт вершины A. Такое переключение позволяет распараллелить выполнение операций A и B. Во втором случае независимость операций gtAElement и gtAReplace позволяет избежать копирования массива по дугам E_1 и E_2.

По определению, индексная зависимость по данным в Sisal-программах возникает между операцией замещения элементов массива (gtAReplace-вершина по терминологии IR1) и операцией выборки над массивами (gtAElement-вершина), если они обращаются к одной и той же ячейке памяти.

После нахождения соответствующих операций gtAReplace и gtAElement, создается уравнение зависимости от их индексных выражений в виде матрицы Coeff.

После того, как была подготовлена информация о гнезде циклов (матрица границ, вектор номеров переменных) и информация об анализируемых операциях, требуется сделать несколько финальных преобразований. Они необходимы для правильной работы алгоритма, анализирующего операции на предмет зависимости.

Пусть система линейных неравенств, определяющих границы, задается матрицей *LUI* (с размером n, m, где n – число переменных в гнезде циклов, m – количество неравенств).

Разделение матрицы границ на матрицы верхних U и нижних L границ.

В результате выполненных выше этапов получается система уравнений зависимости вида (1) и неравенств (2). Далее выполняется шаг вызова алгоритма анализа зависимостей.

3.4. Особенности используемого алгоритма анализа зависимостей

К особенностям алгоритма можно отнести использование новой стратегии тестирования [10], для выявления зависимости по данным. Алгоритм этого метода состоит из серии эффективных и недорогостоящих тестов на зависимость.

В данной стратегии, в зависимости от значений основных параметров задачи (размерность массивов, количество вложенных циклов, значения коэффициентов индексных переменных и значения границ циклов), в первую очередь выделяют часто встречающие и легко разрешимые случаи. Соответственно каждому случаю применяется один быстрый и точный тест или серия эффективных тестов.

Пусть, на вход алгоритма подается гнездо циклов, в котором r – количество вложенных циклов и операции цикла обращаются к элементам d-мерного массива. Кроме того, считаются постоянными и известными значения коэффициентов индексных переменных a_{11}, a_{12} ,…, a_{mn} и значения границ циклов L_1, L_2, ..., L_n ,U_1, U_2, ...,U_n , где n=2*r и m=d. Задача нашего алгоритма выявить зависимости по данным между операцией в итерациях гнезда циклов, т.е. алгоритм должен возвращать ответ «да/нет» о существовании целочисленных решений i_1, i_2..., i_n системы линейных диофантовых уравнений (1), удовлетворяющих ограничениям (2).

Для этого сначала выделены часто встречающие и легко разрешимые случаи задачи зависимости по данным:

> r=1, d=1, т.е. внутри единственного цикла, операторы обращаются к элементам одномерного массива. В этом случае уравнение зависимости (1) выглядит так: $a_1 x_1 + a_2 x_2 = a_0$ и $L \le x_1, x_2 \le U$. Для уравнения целесообразно применить самый быстрый и точный *SIV-тест* [6].

> r>1, d=1, уравнение зависимости имеет вид: $a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = a_0$, где $L_i \le x_i \le U_i$, $i=1,\ldots,n$. Этот случай несколько усложняет решение, следовательно, применяется серия одномерных тестов на зависимость: тест Банержи, I-тест и IR-тест [10]. Каждый следующий тест выполняется только в том случае, если был получен неточный от-

вет (maybe) предыдущим тестом, кроме того, после применения теста Банержи выполняется проверка коэффициентов индексных переменных для уточнения ответов теста [10].

d=2 и имеются *сцепленные индексы*. Система уравнений зависимости имеет вид:

$$a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n = a_{1,0}$$
$$a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n = a_{2,0}$$
$$\text{и } L_i \le x_i \le U_i \qquad \text{где } i=1,\ldots,n.$$

Этот случай доминирует в реальных последовательных программах, но применение обычных одномерных тестов на зависимость в этом случае бесполезно, так как имеются сцепленные индексные переменные. Поэтому применяется серия многомерных тестов: λ–тест, многомерный I-тест и модифицированный λ–тест [12]. Метод запоминания результатов предыдущих тестов и использование их для последующих тестов, оптимизирует данный случай.

В оставшихся случаях уравнение зависимости имеет вид (1) с ограничениями (2). Каждое уравнение рассматривается в отдельности, и для него последовательно применяется серия одномерных тестов: тест Банержи, I-тест и IR-тест. Этот подход дает более точный ответ, если индексные переменные *не сцеплены.* На практике доля сцепленных индексных переменных в ссылках трехмерных массивов и выше незначительна.

Учитывая все случаи, была собрана и реализована библиотека тестов на зависимость. Библиотека состоит из следующих тестов: ZIV-тест, SIV-тест, НОД-тест, Банержи-тест, I-тест, IR-тест, λ-тест, многомерный I-тест и модифицированный λ-тест. Кроме тестов на зависимость в библиотеке имеются алгоритмы для уточнения ответов теста Банержи. Все алгоритмы имеют линейную временную сложность. Из-за высокой стоимости в библиотеку не вошли точные тесты.

Проиллюстрируем сказанное выше примерами.

Рассмотрим предыдущий пример 1. В этом примере выполняются две операции замещения элементов одномерного массива «a»:

```
S1:  a:= old a[2*i:= 1];
S2:      b:= old b[i:= a[2*i+1]].
```

Уравнение зависимости имеет вид:

$2x_1 - 2x_2 = 1$, где $0 \le x_1, x_2 \le 10$.

По схеме новой стратегий к уравнению применяется *сильный SIV-тест*, так как, d=1, r=1 и a_1=a_2.

Стоит отметить, что SIV-тест [5] применяется, когда в индексном выражении массива используется одна индексная переменная. SIV формы индексных выражений делятся на две категорий: сильные и слабые.

Индексное выражение SIV для индекса i, называется *сильным*, если оно имеет вид ($ai + c_1$, $ai' + c_2$), т.е. если оно линейное и коэффициенты двух экземпляров индекса i равны и являются константами. Для *сильных* индексных выражений SIV, *расстояние зависимости* определяется следующем образом:

$$d = i' - i = \frac{c_1 - c_2}{a}$$

Зависимость существует, тогда и только тогда, когда d - целое число и $|d| \le U\text{-}L$, где U и L – верхние и нижние границы цикла соответственно. Таким образом, сильный SIV тест является точным и эффективным тестом.

В данном случае значение $|d|$ –не целое число и, следовательно, зависимости по данным не существует. Тест возвращает ответ «нет», а алгоритм стратегии останавливается и дает ответ об отсутствии зависимости. Тем самым в примере 1, используя информацию о независимости, как показано в разделе 3.3, можно исключить излишнее копирование массива «a» в цикле.

Рассмотрим второй пример, где определяются зависимости по данным для обращений к элементам многомерного массива.

Пример 2.

```
function foo2(b: array[array[real]]
       returns array[array[real]])
 for i in 1, 100 cross j in 1, 100 repeat
   b := old b [3*i+2*j, 2*j := old b[i-j+6, i+j] ]
   returns value of b
 end for
end function
```

В этом примере операции обращаются к элементам двухмерного массива **B** и индексы массивов являются *сцепленными*. Уравнения зависимости представлены в виде системы:

$$\begin{cases} 3x_1 + 2x_2 - x_3 + x_4 = 6 \\ 2x_2 - x_3 - x_4 = 0 \end{cases}$$

где $1 \le x_1, x_3, x_2, x_4 \le 100$.

В данном случае по схеме стратегии принимается решение о применении серии многомерных тестов: λ-тест, многомерный I-тест и модифицированный λ-тест. Зависимость разрушается только после применения модифицированного λ-теста.

3.5. Интерпретация и использование результатов анализа в целях оптимизации

Результатом анализа является установление независимости операций в пространстве итераций гнезда циклов. Данные о зависимости используется при распараллеливании, векторизации и различных оптимизациях циклов, а также при оптимизации работы с памятью (устранение излишнего копирования значений).

4. ЗАКЛЮЧЕНИЕ

Язык Sisal по своей семантике идеально подходят для разработки алгоритмов для параллельных архитектур. Однако, чтобы гарантировать отсутствие сторонних эффектов и наличие прозрачности ссылок в аппликативной программе, операции, изменяющие значения данных, вынуждены работать с копиями этих данных. С этой точки зрения использование массивов, характерное для научных вычислений, очень дорого по расходам времени и памяти, причем эти расходы иногда перекрывают выгоды от параллельного вычисления. Поэтому необходимо решать задачу эффективного использования памяти вычислительной машины. Необходимость решения таких проблем порождает активное использование различных методов оптимизации, в том числе – анализ зависимостей по данным в Sisal-программах.

Основным результатом данной работы является практическая реализация анализатора зависимости по данным в Sisal-программ в рамках системы SFP.

СПИСОК ЛИТЕРАТУРЫ

1. **McGraw J. R.** Parallel functional programming in Sisal: fictions, facts, and future. – Livermore, CA, June 1993. – (Technical Report / Lawrence Livermore National Laboratory; UCRL-JC-114360)

2. **Kasyanov V.N., Evstigneev V.A. et.al.** The system PROGRESS as a tool for parallelzing compiler prototyping // proc. Of Eighth SIAM Conf. On Parallel Processing for scientific Computing (PPSC-97) – Minneapolis, 997. – P. 301–306.
3. **Kasyanov V. N., Stasenko A. P., Gluhankov M. P., Dortman P. A., Pyjov K. A., Sinyakov A. I.** SFP – An interactive visual environment for supporting of functional programming and supercomputing // WSEAS Transactions on Computers. – Athens: WSEAS Press, 2006. – Vol. 5, N 9. – P. 2063–2070.
4. **Касьянов В.Н., Стасенко А.П.** Язык программирования Sisal 3.2. // Методы и инструменты конструирования программ. – Новосибирск, ИСИ СО РАН, 2007. – С. 56–134.
5. **Густокашина Ю.В., Евстигнеев В.А.** IF1 – промежуточное представление Sisal-программ // Проблемы конструирования эффективных и надежных программ. – Новосибирск, 1995. – С. 70–78.
6. **Стасенко А.П.** Внутреннее представление системы функционального программирования **Sisal** 3.0. – Новосибирск, 2004. – 54 с. – (Препр. / РАН. Сиб. отд-е. ИСИ; № 110).
7. **Пыжов К.А.** Блок редукции в компиляторе 3.0. // Методы и инструменты конструирования и оптимизации программ. – Новосибирск: Институт систем информатики имени А. П. Ершова СО РАН, 2005. – С. 185–196.
8. **Евстигнеев В.А.** Анализ зависимостей: состояние проблемы // Системная информатика: Сб. науч. тр. / Ин–т систем информатики СО РАН. – Новосибирск: Наука, 2000.– Вып. 7. – С. 112–173.
9. **Евстигнеев В.А., Арапбаев Р.Н., Осмонов Р.А.** Анализ зависимостей: основные тесты на зависимость по данным // Сиб. журн. вычисл. математики / РАН. Сиб. отд-ние. – Новосибирск, 2007. – Т. 10, № 3. – С. 247–265.
10. **Арапбаев Р**.Н., Осмонов Р.А. Анализ зависимостей: новая стратегия тестирования // Труды Международной конференции. «Параллельные вычислительные технологии (ПаВТ’2007)». – Челябинск: Ид-во ЮУрГУ,2007. – Т. 2. – С. 16–27.
11. **Евстигнеев В.А., Касьянов В.И.** Оптимизирующие преобразования в распараллеливающих компиляторах // Программирование. – №6. – 1996. – С. 12–26.
12. **Арапбаев Р.Н.**, Осмонов Р.А. Анализ зависимостей по данным для многомерных массивов на базе модифицированного λ-теста // Проблемы интеллектуализации качества систем информатики. – Новосибирск, ИСИ СО РАН, 2006. – С. 7–23.