В. Н. Касьянов, Ю. В. Бирюкова, В. А. Евстигнеев

ФУНКЦИОНАЛЬНЫЙ ЯЗЫК SISAL 3.0[1]

ВВЕДЕНИЕ

Название языка Sisal является аббревиатурой английского выражения Streams and Iterations in a Single Assignment Language [1–11]. Создание языка — результат сотрудничества четырех организаций: Ливерморской национальной лаборатории имени Лоренца, Университета штата Колорадо, Манчестерского университета (Великобритания) и Digital Equipment Corporation (DEC). Язык ориентирован на поддержку научных вычислений и представляет собой дальнейшее развитие языка VAL.

Впервые о проекте по созданию языка Sisal упоминается в работе 1983 г. [1], а в 1985 г. появилась первая версия языка Sisal 1.2 [2], которая была реализована на ряде машин, включая Denelcor HEP, Vax 11-780, Cray-1, Cray-X/MP [3]. Для нее имеется также работающий оптимизирующий транслятор OSC [4, 5]. В 1991 г. Ливерморская лаборатория и университет Колорадо опубликовали новую версию языка Sisal 2.0 [6], которая содержала в себе такие идеи современных функциональных языков, как функции высшего порядка, конструкция type set и, наконец, модули. Краткое описание этой версии можно найти в работе [7]. В дальнейшем работа продолжалась, и описание результатов работ над новой (пока никем не реализованной) версией языка — Sisal 90 — увидело свет в 1995 г. [8, 9].

Язык Sisal достаточно широко распространился по США [10]. В целом, по данным Ливерморской лаборатории (на июль 1993 г.), язык использовался в 79 организациях, из которых 16 находились за пределами США.

При разработке языка Sisal авторами проекта преследовались следующие цели:

1. Создать универсальный функциональный язык, который мог бы эффективно исполняться на машинах как последовательной, так и параллельных архитектур, включая новейшие.

2. Создать для представления программ независимое от языка и целевой архитектуры промежуточное представление типа потоковых графов.

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

3. Разработать технику оптимизации для высокопроизводительных параллельных прикладных программ.

4. Разработать микрозадачное окружение для поддержки потокового стиля параллельных вычислений на мультипроцессорах с общей памятью.

5. Добиться эффективности последовательного и параллельного исполнения, сравнимой с императивными языками.

6. Внедрить функциональный стиль программирования для сложных научных применений.

С точки зрения семантики язык Sisal обладает рядом важных свойств. Во-первых, он математически правильный, т. е. функции языка отображают входы в выходы без побочных эффектов. Во-вторых, имена прозрачны для ссылок, т. е. они олицетворяют значения, а не ячейки памяти. В-третьих, Sisal — это язык однократного присваивания.

Обладая всеми качествами, присущими функциональным языкам программирования, Sisal способствует разработке корректных детерминированных программ, которые свободны от совмещения имен, побочных эффектов и ошибок, зависящих от реального времени. Результаты детерминированы, невзирая на архитектуру, операционную систему или обстановку исполнения. В отличие от императивных языков Sisal уменьшает нагрузку на программирование. Пользователь должен определить, что может быть вычислено, и ему достаточно только представлять зависимости по данным между операциями. Компилятор отвечает за планирование операций, передачу значений данных, синхронизацию операций, управление памятью. Легко написать параллельную функциональную программу, так как она кодируется как последовательные императивные программы. Пользователь освобожден от большинства сложностей параллельного программирования и поэтому получает возможность больше сконцентрироваться на конструкции алгоритмов и разработке прикладных программ [12].

В данной работе представлен язык функционального программирования Sisal 3.0, выбранный в качестве начальной версии входного языка системы функционального программирования SFP (СФП), разрабатываемой в ИСИ СО РАН при финансовой поддержке РФФИ (грант N 01-01-00794) и Минобразования. Цель работ — создание системы параллельного программирования SFP на базе языка Sisal, позволяющей прикладному программисту разрабатывать и отлаживать параллельную программу на своем рабочем месте с последующим ее исполнением на супервычислителе, доступном прикладному программисту по сети.

В разд. 2 кратко описана система SFP. Раздел 3 посвящен основным синтаксическим и семантическим характеристикам языка Sisal 3.0. В разд. 4 дается обзор доступных материалов, связанных с проблематикой языка Sisal.

1. СИСТЕМА SFP

Система SFP разрабатывается в рамках проекта ПРОГРЕСС [13, 14] и должна предоставить прикладному программисту на его рабочем месте удобную среду для разработки функциональных программ, предназначенных для последующего исполнения на ЭВМ с параллельными архитектурами, доступными через телекоммуникационные сети. В рамках этой среды программист должен иметь возможность, с одной стороны, создавать и отлаживать программу без учета целевой параллельной архитектуры, а с другой — производить настройку отлаженной программы на ту или другую целевую параллельную архитектуру с целью достижения высокой эффективности исполнения разработанной программы на суперЭВМ.

Процедура настройки состоит в реализации оптимизирующей кросс-трансляции разработанной функциональной программы в программу на языке супервычислителя (например, языке Си), в процессе которой программа подвергается необходимым оптимизирующим и реструктурирующим преобразованиям под управлением пользователя. При этом степень участия программиста может быть различной — от предоставления дополнительной информации до прямого управления производимыми преобразованиями. В частности, программист должен иметь возможность визуальной обработки создаваемой Sisal-программы в рамках ее внутреннего представления.

Система SFP включает следующие компоненты: интерфейс, отладчик, front-end транслятор, блоки промежуточных представлений IR1, IR2 и IR3, блоки анализа, преобразования и визуализации IR1-, IR2- и IR3-программ, конверторы промежуточных представлений, back-end трансляторы. Представления IR1 и IR2 соответствуют по своим функциям и возможностям известным промежуточным представлениям IF1 и IF2 для функциональных программ, а IR3 — графовое представление императивных программ.

2. КРАТКОЕ ОПИСАНИЕ ЯЗЫКА SISAL 3.0

Язык Sisal 3.0 продолжает традицию предыдущих версий и остается языком, ориентированным на написание больших научных программ. В нем адаптированы операции по обработке массивов из Фортрана 90, поддерживается мультиязыковое программирование и включены черты, упрощающие работу пользователя, хотя некоторые из них не укладываются в строгую парадигму функционального программирования. Большинство научных приложений имеет отношение к проблемам спецификации, завершения работы программ, ввода/вывода и обработки ошибок. По своей специфике эти задачи не функциональны и не параллельны, поэтому предполагается кодировать их на языке C. Однако ядро языка Sisal 3.0 параллельно и функционально. В таком определении языка можно усмотреть слияние парадигм императивного и функционального программирования, когда конструкции языка либо принадлежат интегрированному ядру, либо являются языковыми расширениями [15].

Наиболее важными и полезными чертами языка являются:

1. Статический полиморфизм и конструкция **type set,** поддерживающая функциональный полиморфизм и повышающая возможности языка по переиспользованию кода. При отсутствии этой конструкции в блоке **program**, компилятор гарантированно разрешит все типы и создаст эффективный код.

2. Функции высших порядков. Это подразумевает, что функции могут быть параметрами и результатами других функций, а также выступать в качестве значений выражений. Преимущества использования таких функций при научном программировании хорошо известны.

3. Определяемые пользователем редукции. Редукции предназначены для работы с циклическими значениями. Теперь наряду со стандартными редукциями пользователь может создать свои, что облегчает его работу, с одной стороны, и может явиться причиной недетерминированности — с другой.

4. Модули. Они введены в язык для того, чтобы можно было поддерживать несколько стилей программирования, в частности функциональный и модульный.

5. Возможность встраивания в Sisal-программы кода, написанного на языке программирования C.

6. Возможность аннотирования программ, что позволяет описывать контекст ее исполнения. Указанное свойство языка важно для эффективного применения преобразований на этапе трансляции программ, а также для их конкретизации.

2.1. Краткий обзор синтаксиса языка Sisal 3.0

Исполняемая единица языка состоит из одной или нескольких единиц компиляции, каковыми являются программа, интерфейсы и модули. Все эти единицы могут компилироваться отдельно. Простейшей из них является программа. Она состоит из множества деклараций, которые определяют импортируемые из других модулей единицы, некоторые типы данных и функции. Любая функция в программе может быть точкой начала исполнения. На этом уровне (самом внешнем) параметры функций представляют собой значения, получаемые на уровне операционной системы; результаты исполнения функции также относятся к этому уровню.

Модуль — синтаксический аналог программы, который не может служить начальной точкой для исполнения. Он объединяется с интерфейсом для экспорта некоторых своих типов и имен функций. Интерфейс модуля и сам модуль имеют одинаковое имя.

Язык Sisal 3.0 поддерживает стандартные скалярные типы данных: булевские, целые, действительные одинарной и двойной точности, комплексные одинарной и двойной точности и null. Из структурных типов в языке имеются массивы, потоки, записи, союзы и type set. Для определения сложной структуры данных необходимо указать типы составляющих ее элементов, число элементов при этом не указывается и становится известным только при присвоении структуре значения. Массивы и потоки состоят из однотипных элементов и различаются прямым и последовательным доступом к элементам. Записи и союзы состоят из разнотипных элементов и различаются способом хранения.

Основной синтаксической единицей языка является выражение, оно представляет список значений некоторых типов. Арность выражения — это размер списка значений. Основной конструкцией языка является let-выражение, которое вводит имена значений и заголовки функций, производит над ними некоторые действия и выдает результаты.

В языке присутствуют достаточно большое разнообразие видов циклов: итерационные циклы с инициализацией значений, циклы с предусловием и постусловием, а также явные параллельные циклы, экземпляры тела которых могут параллельно выполняться на различных процессорах.

К выражениям выбора, присутствующим в языке, относятся конструкции if и case, причем в последней имеется возможность выбирать ветвь не только по значению, но и по типу имени.

Функция языка Sisal состоит из заголовка и тела. В заголовке объявляются имена и типы формальных параметров, а также типы возвращаемых результатов. Существует возможность предописания функций и создания рекурсивных функций, а также описания функций, закодированных на языке программирования С.

Достаточно подробно многие из указанных конструкций языка описаны в руководстве пользователя [9], поэтому ниже мы остановимся лишь на тех структурах, о которых не упоминается в этом руководстве.

2.2. Средства модульного программирования

Так как язык Sisal 3.0 поддерживает принципы модульного программирования и раздельной компиляции, введем понятие единицы компиляции.

Единица компиляции — это текст, который может компилироваться без связи с другими текстами. Простейшей формой такой единицы является программа (**program**), содержащая только функции и определения типов. Эти функции вызываются с уровня операционной системы. Входом в программу являются фактические параметры вызываемых функций, а результаты вызываемых функций становятся выходом программы.

Важно отметить, что типы параметров и результатов не могут быть членами конструкции **type set**, функциями и составными значениями, включающими **type set** и функции, т. е. в этом контексте не допустим любой “непрозрачный” тип данных.

Программа может состоять одной или более единиц компиляции, при этом среди них обязательно должна существовать одна, которая использует служебное слово **program** и содержит функции, запускающие вычисления. Некоторые функции и типы, определенные в одной единице, могут использоваться (импортироваться) другими. В общем случае существует пара единиц компиляции: одна — **module**, которая содержит множество функций и определений типов, и другая — **interface**, которая определяет подмножество функций и типов конкретного модуля, доступных другим единицам компиляции. Образование пары осуществляется использованием обеими единицами одного и того же имени, например:

```
interface X
        % Объявление функций и типов,
        % доступных для других единиц компиляции
end interface
module X
```

```
        % Здесь содержатся полные определения функций и типов
        % в соответствующем интерфейсе и, может быть, другие опре-
          деления
end module
```

Интерфейс определяет связи модуля с внешним миром. Объявления импорта позволяет использовать функции или типы, определенные в одном модуле, в других единицах компиляции с помощью служебного слова **from**. Если не указано, какие именно функции и значения импортируются, то предполагается, что используются все описанные в интерфейсе значения и функции, например:

```
module Y
    from X: a, b, c
    . . .
end module
```

При компиляции *Y* некоторое представление о декларации *a, b* и *c* должно быть доступным. Поэтому компиляция *Y* возможна, если интерфейс модуля *X* уже написан и заранее откомпилирован, хотя исходный текст модуля *X* может быть в данное время недоступен.

2.3. Средства препроцессирования

В интегрированную среду подготовки программ на языке Sisal 3.0 как обязательный компонент предполагается включить препроцессор, назначением которого является обработка исходного текста программы до ее компиляции. Он также расширяет возможности языка, поддерживая:

1) подстановку имен и макросы,
2) включение файлов,
3) условную компиляцию,

что позволяет сберечь время при разработке программ и создать гибкие, удобочитаемые и более пригодные для сопровождения и отладки программы.

Для управления препроцессором используются директивы. Директивой служит строка исходного файла, в первой позиции которой указан символ #. Директивы могут размещаться в любом месте исходного файла, их действие остается в силе вплоть до конца файла. В соответствии со сказанным ранее имеются три типа директив препроцессора:

1) #define, #undef — первая предусматривает определение макросов или препроцессорных идентификаторов, каждому их которых ставится в соответствие некоторая символьная последовательность. В последующем тексте программы эти идентификаторы заменяются на указанные последовательности символов. Вторая директива отменяет действие первой;

2) #include — позволяет включать в текст программы текст из выбранного файла;

3) #if, #ifdef, #ifndef, #else, #endif, #elif — позволяют организовать условную обработку текста программы. Условность состоит в том, что компилируется не весь текст, а только его части, которые так или иначе выделены с помощью перечисленных директив.

Предполагается, что препроцессор будет работать аналогично препроцессору языка С, поэтому более подробную информацию о его работе можно найти в [16].

2.4. Средства аннотирования программ

Для более гибкого применения оптимизирующих преобразований на этапе трансляции и эффективного использования программ предполагается поддерживать возможность создания на языке Sisal аннотированных программ [17], т.е. таких программных текстов, в которых содержатся как собственно программные части (базовые программы), так и аннотации — формализованные комментарии, предназначенные для спецификации контекстов применений базовых программ.

Синтаксически каждая аннотация — это фрагмент Sisal 3.0 программы, оформленный в виде комментария, в котором в каждой строке после символа % расположен символ $.

Различаются глобальные и локальные аннотации.

Глобальные аннотации формируют полные единицы компиляции (модули аннотаций), которые вполне аналогичны единицам компиляции базовой программы и в которых практически без каких-либо ограничений могут использоваться любые конструкции базового языка Sisal 3.0.

Локальные аннотации являются частью единиц компиляции базовой программы и могут иметь вид объявлений, утверждений и директив (следует отличать их от ранее описанных директив препроцессора).

Объявления позволяют определять дополнительные имена, а также связи модуля с внешним миром.

Каждое утверждение — это логическое выражение, которое всегда истинно при любом допустимом исполнении базовой программы. Утверждения могут использоваться для описания свойств не только объектов (например, диапазонов входных и выходных значений), но и конструкций аннотированной программы.

Синтаксически каждая директива — это либо оператор присваивания, либо вызов функции. С помощью директив можно управлять процессом обработки транслируемой программы. Например, при помощи директив можно указывать характеристики целевой архитектуры или включать применение такого оптимизирующего (или реструктуирующего) преобразования, для которого автоматически не удается проверить справедливость корректности или целесообразности его выполнения.

Каждая локальная аннотация ассоциируется компилятором с первым зарезервированным словом, предопределенной функцией, константой, именем или арифметическим символом, следующими за ней в исходном тексте. Семантика локальных аннотаций зависит от применения.

2.5. Проблемные области

С точки зрения реализации языка Sisal 3.0 можно выделить две проблемные области. Первая затрагивает вопросы ввода/вывода. Все предыдущие версии языка не содержат операторов ввода/вывода, программа (версия Sisal 2.0) или функция main (версии Sisal 1.2 и Sisal 90) получают входные параметры и выдают выходные значения на уровне операционной системы; вывести промежуточные данные практически невозможно. Этот факт затрудняет процесс отладки и сопровождения программ. Поэтому в версии Sisal 3.0 предлагается использовать стандартную библиотеку функций ввода/вывода на языке C. Однако это порождает вторую проблемную область — из-за введения императивных процедур невозможно гарантировать детерминированность программ, которая заложена в семантике языка Sisal.

3. БЛИЗКИЕ РАБОТЫ

Приведем краткую характеристику доступных работ, которые так или иначе связаны с языком Sisal. Чтобы как-то систематизировать этот материал, выделим организации, работающие над проблематикой языка, для каждой из которых дадим обзор имеющихся опубликованных материалов.

3.1. Ливерморская национальная лаборатория им. Лоренца

Организация является одним из основных разработчиков языка Sisal и методов его реализации, базирующихся на промежуточных представлениях транслируемых программ IF1 [18, 19] и IF2 [20].

Оба представления являются иерархической графовой формой и подобны потоковым графам. Представление IF2 отличается от IF1 тем, что поддерживает прямые операции с памятью. Над представлениями можно провести машинно-независимые оптимизирующие преобразования, например исключение общих подвыражений, удаление инвариантов цикла. К ним возможно применение и машинно-зависимого анализа: нахождение векторных операций или распараллеливание графа для мультипроцессорных систем.

Силами Ливерморской лаборатории реализована версия Sisal 1.2, для нее существуют оптимизирующий компилятор OSC [4, 5], инструментальная система TWINE [21], а также интерпретатор.

Аппликативные языки, каковым является Sisal, по своей семантике идеально подходят для разработки алгоритмов для параллельных архитектур. Однако, чтобы гарантировать отсутствие сторонних эффектов и наличие прозрачности для ссылок в аппликативной программе, операции, которые изменяют значения данных, вынуждены работать с копиями этих данных. С этой точки зрения использование массивов, что характерно 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. Существует несколько подходов к разрешению данной проблемы.

Ренеллетти предложил алгоритм, который работает на внутреннем представлении IF1. Граф исходной программы анализируется в два прохода с целью получения для каждого значения, вычисляемого программой, выражения его размера. Эта информация о размере, вычисляемая во время исполнения программы, используется для выделения буферов памяти для агрегатных структур, которые вычисляются позже и записывают свои элементы в уже отведенную для них память. Такое раннее выделение областей памяти под элементы данных делает ненужным размещение в памяти промежуточных буферов, тем самым сокращая количество операций копиро-

вания. Алгоритм не работает только в тех случаях, когда невозможно вычислить размер данных до их непосредственного вычисления [22].

3.2. Университет штата Колорадо

Основные направления исследований ученых университета штата Колорадо связаны с эффективной реализацией языка Sisal 1.2.

С этой проблематикой тесно связана задача эффективного использования памяти вычислительной машины, которая обсуждалась в предыдущем пункте. Девид Канн предложил алгоритм, работающий на промежуточном представлении IF2 с теми агрегатными структурами, для которых не пригоден алгоритм Ренеллетти. Размер этих структур не известен во время компиляции, поэтому невозможно выделить буфер в памяти, в который во время вычисления записывается структура. Алгоритм делится на три фазы: первая фаза готовит каждый граф к анализу, вторая удаляет ненужные операции по подсчету ссылок на агрегаты, а третья удаляет ненужные операции копирования и идентифицирует агрегаты, на которые ссылается только одна операция. Практические результаты показали, что две последние фазы алгоритма удаляют около 98% операций по подсчету ссылок и около 82% операций копирования [5].

3.3. Манчестерский университет (Великобритания)

Исследования сотрудников Манчестерского университета связаны с разработкой формального алгоритма эффективного использования памяти под агрегатные структуры для языков однократного присваивания [23, 24]. Данный алгоритм рассматривает все операции над агрегатными структурами. Для них вычисляется несколько формальных характеристик, которые помогают выделить те операции, которые могут выполняться над самими данными, а не над их копиями. Алгоритм может работать с вложенными агрегатами, используя метод подсчета числа операций, работающих с этими данными. Этот же алгоритм предусматривает, где это возможно, применение стратегии предраспределения памяти под агрегатные структуры, сходной со стратегией Ренеллетти, описанной выше.

3.4. Университет Nice Sophia Antipolis (Франция) и университет Аделаиды (Австралия)

Внимание ученых этих университетов привлекла версия языка Sisal 2.0, реализацией которой они и занимаются [25, 26]. К 1996 г. разработано формальное определение динамической семантики значительной части языка с использованием правил вывода Typol системы Centaur, что послужило строгому пониманию дизайна языка, т.е. исключило двоякую трактовку конструкций, обеспечило ценной информацией как разработчиков, так и пользователей языка, помогло сравнить Sisal с другими функциональными языками. К тому же система спецификаций Centaur позволяет автоматически создать структурный редактор и интерпретатор, которые в дальнейшем могут развиться в интерактивную оболочку для программирования на языке Sisal. Система Centaur дает возможность также разработать инструменты анимации для показа процесса вычислений.

Конечной целью данных исследований является формальная характеристика методов распараллеливания и алгоритмов компиляции языка с целью создания гибкой оболочки с элементами визуализации для программирования на языке Sisal.

3.5. Кипрский университет

Сотрудники Кипрского университета работают над созданием системы Mustang для автоматического распараллеливания Fortran-программ путем отображения их в семантику языка однократного присваивания [26–31]. Предполагается, что в начале исходная Fortran-программа транслируется в промежуточное представление IF1. Во время трансляции используется система Parafrase 2 для грамматического разбора исходной программы, проведения анализа зависимостей по данным и определения возможностей для распараллеливания, которые потом явно представляются в IF1-программе. Затем IF1-программа пропускается через соответствующие блоки оптимизирующиго Sisal-компилятора (OSC). На данный момент создан и оттестирован действующий прототип системы Mustag.

СПИСОК ЛИТЕРАТУРЫ

1. **McGraw, J. R. et. al.** Sisal: Streams and iterations in a single assignment language, Language Reference Manual, Version 1.1 / Lawrence Livermore Nat. Lab. Manual M-146. — Livermore, CA 1983.
2. **McGraw, J. R. et. al.** Sisal: Streams and iterations in a single assignment language, Language Reference Manual, Version 1.2 / Lawrence Livermore Nat. Lab. Manual M-146 (Rev. 1). — Livermore, CA 1985.
3. **Skedzielewski S. K., Welcome M. L.** Data flow graph optimization in IF1 // Lect. Notes Comput. Sci. — 1985. — Vol. 201. — P. 17–34.
4. **Cann D.C.** The optimizing SISAL compiler: Version 12.0. — Livermore, Apr.2, 1992. — (Prepr. / Lawrence Livermore National Laboratory; UCRL-MA-110080).
5. **Cann D. C.** Compilation techniques for high performance high performance applicative compilation // Technical Rep. CS-89-108. — Colorado State University, 1989.
6. **Bohm A. P. W., Oldenhoeft R. R., Cann D. C., Feo J. T.** The SISAL 2.0 Reference Manual. — Livermore, CA, 1991. — (Prepr. / Lawrence Livermore Nat. Lab.; UCRL-MA-109098, LLNL).
7. **Евстигнеев В. А., Городняя Л. В., Густокашина Ю. В.** Язык функционального программирования SISAL // Интеллектуализация и качество программного обеспечения. — Новосибирск, 1994. — С. 21–42.
8. **Feo D. T., Miller P. J., Skedzielewski S. K., Denton S. M.** Sisal 90 User's Guide / Lawrence Livermore Nat. Lab. Draft 0.96. — Livermore, CA, 1995.
9. **Бирюкова Ю. В.** SISAL 90 руководство пользователя. — Новосибирск, 2000. — 84с. — (Препр./ РАН. Сиб. Отд-е. ИСИ; № 72).
10. **Feo J. T.** Sisal. — Livermore, CA, 1992. — (Prepr. / Lawrence Livermore Nat. Lab.; UCRL-JC-110915, LLNL).
11. **Feo J. T., Cann D.C, Oldehoeft R.R.** A report on the Sisal language project // J. on Parallel and Distributed Computing. — 1990. — Vol. 10. — P. 349–366.
12. **McGraw J. R.** Parallel functional programming in Sisal: fictions, facts, and future. — Livermore, CA, 1993.— (Prepr. / Lawrence Livermore Nat. Lab.; LLNL).
13. **Kasyanov V. N., Evstigneev V.A. et al.** The system PROGRESS as a tool for parallelizing compiler prototyping // Proc. of Eighth SIAM Conf. on Parallel Processing for Scientific Computing (PPSC-97). — Minneapolis, 1997. — P. 301–306.
14. **Kasyanov V.N., Evstigneev V.A. et al.** Support tools for supercomputing and networking // Lect. Notes Comput. Sci. — 1999. — Vol.1593. — P. 431–434.
15. **Feo D. T., Miller P. J., Skedzielewski S. K., Denton S. M., Solomon C. J.** Sisal 90 // Proc. High Performance Functional Computing — Livermore, 1995. — P. 35-47.
16. **Трой Д.** Программирование на языке Си для персонального компьютера IBM PC / Пер. Б.А. Кузьмина под ред. И. В. Емелина. — М.: Радио и связь, 1991.

17. **Касьянов В. Н.** Оптимизирующие преобразования программ. — М.: Наука, 1988.
18. **Skedzielewski S. K., Glauert J.** IF1 — An intermediate form for applicative languages. Manual M-170 / Lawrence Livermore National Laboratory — Livermore, CA, 1985.
19. **Густокашина Ю.В., Евстигнеев В.А.** IF1 — промежуточное представление Sisal-программ // Проблемы конструирования эффективных и надежных программ. — Новосибирск, 1995. — С. 70–78.
20. **Welcome M., Skedzielewski S., Yates R.K., Ranelletti J.** IF2 — An applicative language intermediate form with explicit memory management / Lawrence Livermore Nat. Lab. Manual M-195. — Livermore, CA, 1986.
21. **Miller P. J.** TWINE: a portable, extensible SISAL execution kernel // Proc. Second SISAL User's Conf. — Livermore, 1992. — P. 243–256.
22. **Ranelletti J. E.** Graph transformation algorithms for array memory optimization in applicative languages. — Livermore, CA, 1987. — (Prepr. / Lawrence Livermore National Laboratory; UCRL-53832).
23. **Li Z., Kirkham C.** Efficient implementation of aggregates in united functions and obiects. — University of Manchester, 1995.
24. **Li Z., Kirkham C.** Efficient storage reuse of aggregates in single assignment languages. — University of Manchester, 1996.
25. **Attali I., Caromel D., Guider R., Wendelborn A. L.** Optimizing Sisal programs: a formal approach // Proc. Europar'96. –1996. — P. 1123–1124.
26. **Attali I., Caromel D., Wendelborn A. L.** A formal semantics and an interactive environment for Sisal // Tools and Environments for Parallel and Distributive Systems. — Kluver Academic Publishers, 1996.– P. 231–258.
27. **Cann D. C., Evripidou P.** Advanced Array Optimizations for high performance functional languages // IEEE transactions on parallel and distributed systems. — 1995. — Vol. 6, No. 3.
28. **Evripidou P., Gaudiot J.** Incorporating input/output operations into dynamic dataflow graphs // Parallel computing. — 1995. — Vol. 21. — P. 1285–1311.
29. **Evripidou P., Barry R.** Mapping Fortran programs to single assignment semantics for efficient parallelization // Parallel Processing Letters. — 1998. — Vol.8, N 3. — P. 407–418.
30. **Lachanas A., Evripidou P.** Exploiting course grain parallelism from Fortran by mapping it to IF1 // Lect. Notes Comput. Sci. — 1998. — Vol. 1470.
31. **Evripidou P., Lachanas A.** Venus compiler: mapping Fortran to single assignment semantics for efficient parallelization / Tech. Rep. TR-99-3. — University of Cyprus, 1999.