К. А. Пыжов*

БЛОК РЕДУКЦИИ В КОМПИЛЯТОРЕ SISAL 3.0

ВВЕДЕНИЕ

Функциональное программирование — парадигма, значительно отличающаяся от последовательной модели программирования: функциональная программа является по сути композицией функций, в которой каждая функция сама по себе есть композиция функций или примитивных операторов (арифметические операции, условные выражения и т.д.). Это означает, что разработчик не должен задумываться над явным описанием параллельных процессов; разбиение программы на процессы определяется иерархией функций и зависимостями по данным в программе.

SISAL — функциональный язык, дающий широкие возможности для автоматического анализа и управления параллелизмом на этапе трансляции. Имена в SISAL определяют «идентификаторы» значений, а не ячейки памяти. Значения, вычисляемые и используемые в программе, являются динамическими объектами, т.е. идентификаторы связаны с конкретными ячейками памяти только во время «существования» этих значений при выполнении программы. Это определяет динамический характер графа потока данных в представлении программы, при этом вершины графа представляют операторы, а дуги — значения, передаваемые между вершинами. «Пространство существования» значения — это множество дуг, принадлежащих всем путям от определения этого значения до последнего его использования. Значения, определяемые дугами графа, могут иметь связанные с ними имена, а могут и не иметь.

Значения всех семантических элементов языка SISAL определяются только значениями формальных аргументов и вложенных выражений. Это исключает сторонние эффекты и позволяет производить более эффективный анализ программ, чем в случае императивных языков. Кроме того, поскольку SISAL является языком однократного присваи-

[0]*pyjov@ngs.ru

Работа выполнена при финансовой поддержке Министерства образования РФ (научная программа "Университеты России", грант № УР.04.01.201).

вания, он не содержит операций записи в память. В связи с этим отпадает необходимость исследования таких зависимостей по данным, как антизависимость и зависимость «по выходу».

С другой стороны, получение эффективного кода при трансляции таких программ вызывает некоторые трудности, связанные с необходимостью применения специфических оптимизаций, с задачей автоматического распределения потока управления, а также с невысокой популярностью языков «потока данных» вообще, и вследствие этого с очень небольшим по сравнению с императивными языками количеством разработанных алгоритмов и технологий трансляции. Тем не менее, языки типа SISAL вполне могут конкурировать с традиционными языками при исполнении на параллельных архитектурах ([4, 6, 7]). При этом особое внимание необходимо уделять оптимизации программы на разных уровнях.

1. ТРАНСЛЯТОР SISAL 3.0 В СИСТЕМЕ SFP

Транслятор языка SISAL, рассматриваемый в данной статье, является частью системы программирования SFP, использующей в качестве входного языка функциональный язык Sisal 3.0. Система SFP должна предоставить программисту на его рабочем месте удобную среду для разработки функциональных программ, предназначенных для последующего исполнения на ЭВМ с параллельной архитектурой. В рамках этой среды программист должен иметь возможность, с одной стороны, создавать программу без учета целевой параллельной архитектуры, а с другой — производить настройку программы на ту или иную целевую параллельную архитектуру с целью достижения высокой эффективности исполнения разработанной программы на суперЭВМ. Процедура настройки состоит в реализации оптимизирующей трансляции разработанной функциональной программы в программу на языке Си, в процессе которой программа подвергается необходимым оптимизирующим преобразованиям под управлением пользователя. 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 приближено к внутреннему представлению импера-

тивных программ. Кроме того, для визуализации внутренних представлений и преобразований программ используется система визуализации иерархических графов HIGRES [3].

Блок оптимизации на представлении IR1 выполняет такие высокоуровневые преобразования, как удаление общих подвыражений, свертка констант, упрощение условных выражений, удаление «мертвого» кода и т.д. Поскольку на этапе первого внутреннего представления отсутствует информация о конкретной архитектуре и распределении потока управления, все преобразования IR1 являются редуцирующими, т. е. гарантированно не снижают эффективность программы [1]. Такие специфические преобразования, как удаление копирований, выполняются на следующем представлении IR2. Оптимизации, связанные с потоком управления, и машинно-зависимые оптимизирующие преобразования относятся к представлению IR3.

1.1. Внутреннее представление IR1

Для представления программ, написанных на языке SISAL, требуется внутреннее представление, отражающее особенности языка, такие как машинно-независимый параллелизм и функциональность. Для одного из первых трансляторов SISAL, системы OSC (Optimizing SISAL compiler [4]), был разработан язык IF1 [8]. Этот язык основывался на ациклических графах и идеально подходил для поставленной цели в рамках языка SISAL 1.0. Развитием IF1 является представление IR1, используемое в настоящий момент в системе SFP [2].

Первоначально в системе SFP использовалась реализация представления IF1, написанная на языке C++ и фактически являвшаяся частью транслятора SISAL 3.0. В свою очередь, блок оптимизации IF1 являлся частью этого представления. Однако впоследствии было принято решение реализовать блок внутреннего представления в виде COM-интерфейсов, что позволило облегчить разработку отдельных частей компилятора и повысить его надежность. В связи с этим потребовалось изменить реализацию блока оптимизаций первого внутреннего представления.

IR1, как и IF1, базируется на потоковых графах. Это наиболее удобная и подходящая форма представления SISAL-программ. Вершины графа представляют операции, операторы и некоторые стандартные конструкции; дуги представляют значения, передающиеся от вершины к вершине. Каждой дуге приписан тип значения, которое она несет. На-

пример, граф для функции

```
function F(i : integer   returns real)
 let
     a := i;
     b := 2
 in
     (a+b)/5

 end let
end function
```

в представлении IR1 имеет вид, представленный на рис. 1.

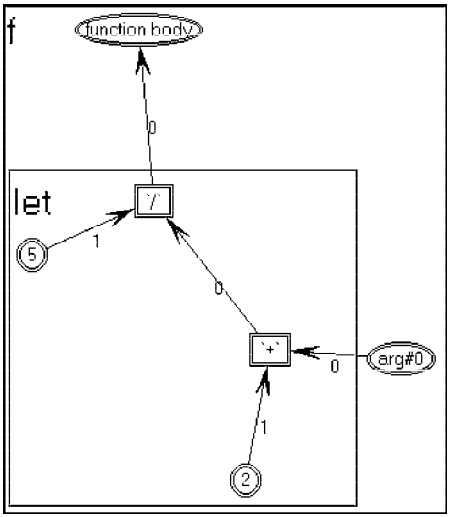

Рис. 1.

Литералы в IR1 представляются специальными вершинами. Эти вершины содержат тип литерала и значение. Каждая вершина имеет входные и выходные порты. Количество портов соответствует количеству аргументов и результатов. Функция является подграфом основного графа программы, содержащим собственно тело этой функции. Каждая функция-подграф имеет входы-аргументы и выходы-результаты.

Пример.

```
function Pythag(x, y : real   returns real)
    sqrt(x*x + y*y)
end function

function main(r : real  returns real)
   Pythag(r, 2.0)
end function
```

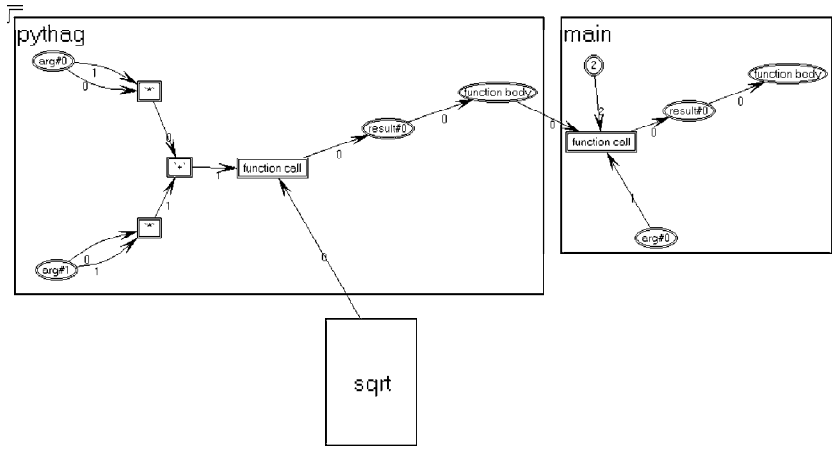

Рис. 2.

2. НЕКОТОРЫЕ ПРЕОБРАЗОВАНИЯ IR1

Как уже отмечалось, блок оптимизации выполняет некоторые редуцирующие оптимизирующие преобразования графа представления IR1. Некоторые из них являются локальными (например, свертка константных выражений), другие требуют выполнения нелокальных контекстных условий.

Подстановка параметров функции.

Это преобразование заменяет в графе внутреннего представления функции вершину-параметр на вершину-константу.

Пример.

```
function  f(x, y  : real   returns real)
  let
       s := if  x > y then x*x - y*y
                      else 0
  in
       s
  end let
end function

function first(x, r : real   returns real, real)
        f(x, 0.25),
        second(x, r)
end function

function second(x, r: real   returns real)
   f(x, r)*r
end function

function main(r: real    returns real, real)
 let
      x := 20
 in
     first(x, r)
 end let
end function
```

Внутреннее представление этого фрагмента представлено на рис. 3.

Функция first вызывается с константным значением первого параметра : 20, поэтому к ней применима подстановка параметра. Аналогичные преобразования применяются к функциям f и second. После выполнения этой редукции внутреннее представление рассматриваемого фрагмента примет вид, изображенный на рис. 4.

Редукция условного выражения.

Данное преобразование применяется к условному выражению с тождественно истинным либо тождественно ложным условием. Осуществляются анализ статической вычислимости условия и замена конструкции условного выражения на ветвь, соответствующую тождественно истинному условию.

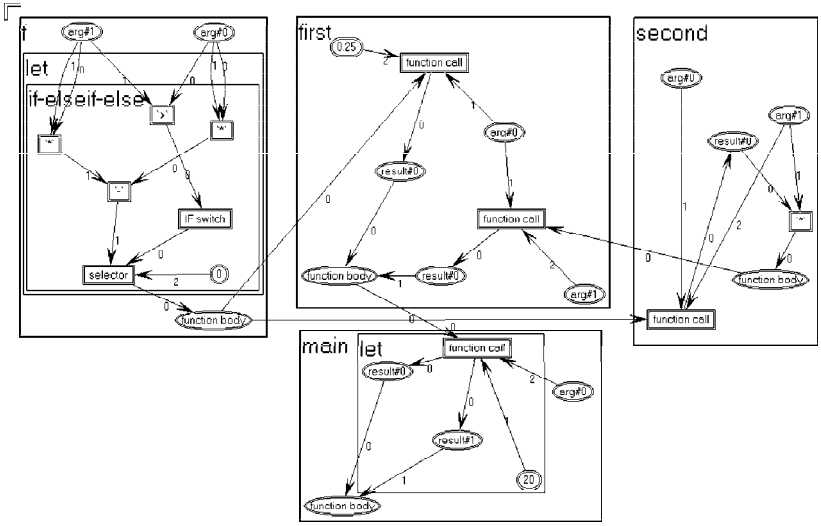
f
arg#1
arg#0
let
if-elseif-else
IF switch
selector
function body
first
0.25
function call
result#0
arg#0
second
arg#1
main
let
result#1
20

Рис. 3.

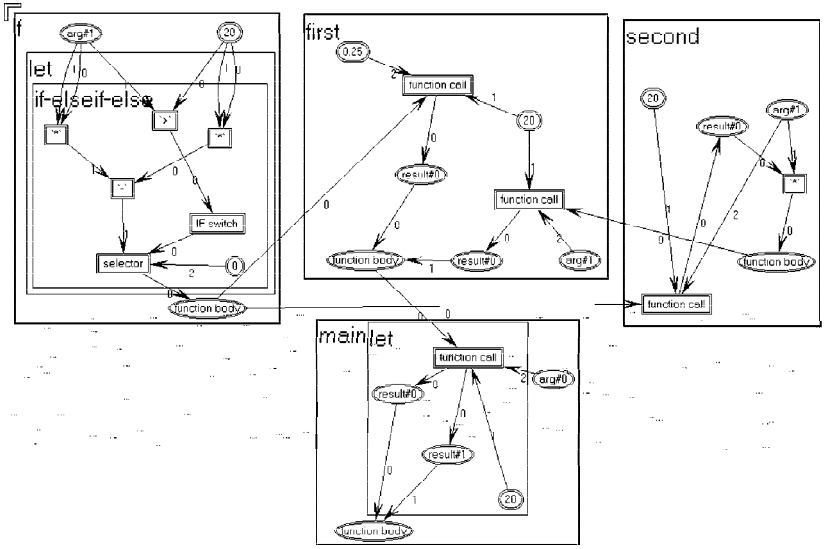
f
arg#1
20
let
if-elseif-else
IF switch
selector
function body
first
0.25
function call
result#0
second
main
let
result#1

Рис. 4.

Пример.

```
function f(x:real   returns real)                %x=0
 let
      p := 0.5 - sin(x);
      s := if p >=0 then  sqr(p)
                          else 0
           end if
in
 sin(s)  end let
end function
```

или на внутреннем представлении:

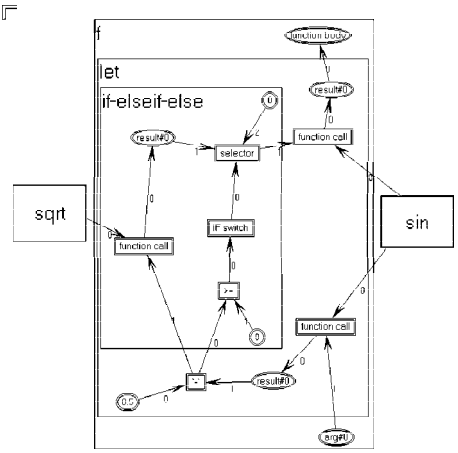

Рис. 5.

Условное выражение в присваивании переменной s имеет недостижимую else-ветвь (в контексте $x = 0$), поэтому ее можно отсечь. Фрагмент, полученный в результате такой редукции, представлен на рис. 6.

Редукция выражения CASE.

Редукция заключается в замене конструкции case на ветвь, соответствующую истинному логическому значению. Производится проход по всем условным ветвям и сравнение управляющего выражения (которое является константой) с тестовыми значениями ветвей. Если тестовыми значениями является диапазон, происходит проверка принадлежности управляющего выражения этому диапазону значений.

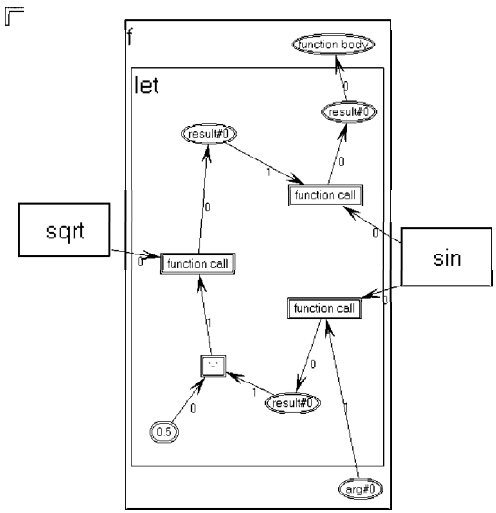

Рис. 6.

Редукция пустых циклов.

Если удается статически доказать, что выражение for ни разу не выполняет тело цикла, такой цикл удаляется.

Удаление «мертвого» кода.

В результате выполнения некоторых преобразований (например, редукции условного выражения) в графе внутреннего представления IR1 могут появиться функции, нигде не вызываемые, т.е. не связанные дугами с другими частями графа программы. Данное редуцирующее преобразование удаляет такие подграфы-функции из графа программы.

Свертка константных выражений.

Производится вычисление и подстановка значений, которые могут быть вычислены статически.

Рассмотрим следующий фрагмент программы:

```
function f(i, j: integer   returns integer)
    let
        p:=2
    in
        4*p + 3.5*i + j
```

```
        end let
    end function
```

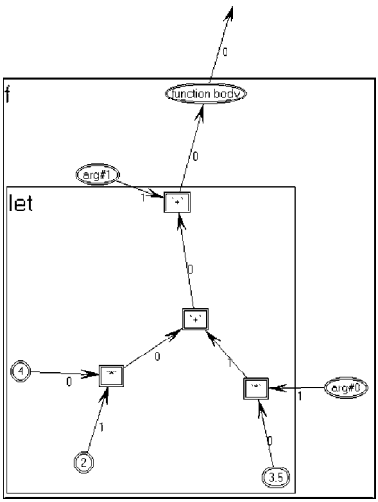

Рис. 7.

Выражение $4*p$ является в данном случае константным, поэтому его естественно вычислить на стадии трансляции. Фрагмент, полученный в результате такого преобразования, представлен на рис. 8.

Алгебраические упрощения.

Реализована редукция выражений вида E*1 и E+0, где E — любое допустимое выражение языка Sisal.

Упрощение условий.

В ряде случаев в условных операторах часть условия или все условие целиком является статически вычислимым. Такие конструкции возникают не только непосредственно из транслируемой программы, но и в результате применения других редукций, скажем, тех же константных вычислений и подстановок.

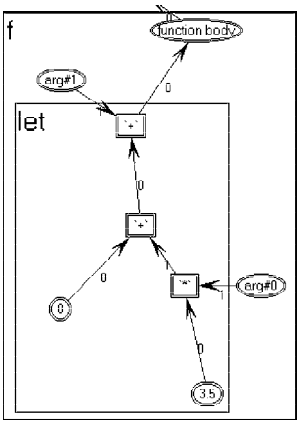

Рис. 8.

ЗАКЛЮЧЕНИЕ

Был реализован блок редуцирующих преобразований для системы SFP, включающий описанные выше оптимизирующее преобразования. Блок являлся частью реализации представления IF1, которое использовалось в первоначальной версии транслятора. Сейчас стоит задача переноса оптимизмрующих преобразований на новое представление IR1.

СПИСОК ЛИТЕРАТУРЫ

1. **Касьянов В. Н.** Оптимизирующие преобразования программ. — М.: Наука, 1986.
2. **Стасенко А. П.** Внутреннее представление системы функционального программирования SISAL 3.0. — Новосибирск, 2004. — 54 с. — (Препр. / ИСИ СО РАН; № 110).
3. **Лисицын И. А.** Применение системы HIGRES для визуальной обработки иерархических графовых моделей // Проблемы систем информатики и программирования. — Новосибирск, 1999.
4. **Sarkar V., Cann D.** POSC — a Partitioning and Optimizing SISAL Compiler. — ACM Digital Library, 1990.
5. **Simpson R.** SISAL Compiler User Manual. — Livermore, CA, 1986.

6. **Gaudiot J.-L., Bohm W., Najjar W., DeBoni T., Feo J.** The SISAL Model of Functional Programming and its Implementation. — IEEE, 1997.
7. **Gharachorloo K., Sarkar V., Hennessy J.L.** A Simple and Efficient Implementation Approach for Single Assignment Languages. — ACM Digital Library, 1988.
8. **Skedzielewski S., Glauert J.** IF1: an Intermadiate Form for Applicative Languages. — Livermore, CA, 1985.