Д. С. Гордеев

ВИЗУАЛИЗАЦИЯ ВНУТРЕННЕГО ПРЕДСТАВЛЕНИЯ ПРОГРАММ В СИСТЕМЕ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ SFP

ВВЕДЕНИЕ

Для решения сложных вычислительных задач можно эффективно использовать многопроцессорные системы, однако для этого нужны специальные средства. В настоящее время в Институте систем информатики им. А. П. Ершова СО РАН ведётся работа над проектом системы функционального программирования SFP [2]. Система функционального программирования SFP предназначена для создания и отладки программ, переносимых на произвольную параллельную архитектуру. Пользователь системы сможет создавать и отлаживать программы на своём рабочем месте и запускать на исполнение на доступном по сети вычислителе с параллельной архитектурой.

В рамках проекта по созданию системы функционального программирования SFP было разработано графовое внутреннее представление IR1 [4]. Далее будут использоваться термины теории графов, с которыми можно ознакомиться в книге [1]. Разработанное внутреннее представление обладает следующими свойствами:

1. Машинная независимость для представления параллелизма и для значений типов данных.
2. Полнота внутреннего представления, позволяющая произвести его ретрансляцию в семантически эквивалентную программу на исходном языке.
3. Возможность ретрансляции после преобразований внутреннего представления, сохраняющих его корректность для исходного языка.
4. Лёгкость исполнения заданных внутренним представлением вычислений без проведения дополнительных преобразований структуры внутреннего представления.
5. Структурированность объектов внутреннего представления для задания естественной иерархии одних конструкций исходного функционального языка в другие.

6. Явное задание любого типа неявных действий с помощью объектов внутреннего представления.
7. Лёгкое введение новых объектов внутреннего представления для задания новых конструкций языков программирования и типов данных.

Основой структуры внутреннего представления являются потоковые графы, которые позволяют задавать явные информационные (семантические) связи (дуги) между операциями (вершинами) и делают процесс интерпретации осуществимым без дополнительных преобразований. Благодаря этому нет побочных эффектов вычислений (ввиду отсутствия понятия переменной) — естественного свойства чисто функциональных языков. Потоковые графы также позволяют задавать параллелизм на уровне отдельных информационно независимых операций, не зависящий от машинной архитектуры.

Внутреннее представление IR1 было разработано на основе языка IF1 [8]. Этот язык основывается на ациклических иерархических графах. Язык IF1 разрабатывался как графовый язык, который мог бы стать результатом трансляции компиляторов нескольких функциональных языков. С помощью такого графового языка становятся возможными объединение и единые преобразования для программ, написанных с помощью разных функциональных языков. Язык IF1 недостаточен для представления программ функциональных языков, но предполагается, что язык IF1 является достаточной основой для более обобщенных промежуточных представлений, требуемых для описания более широкого множества функциональных языков. Программа на языке IF1 состоит из множества графов специального вида, среди которых выделено подмножество графов, соответствующих множеству функций исходной SISAL [3] программы. Граф в IF1 состоит из объектов трёх видов: вершин, дуг и рамки графа. Вершинам и рамке графа приписаны упорядоченные множества входов портов (в которые входят дуги, но не более одной для одного порта) и выходных портов (из которых выходят дуги). Входные порты рамки графа рассматриваются как выходные порты (результаты вычислений) графа, а выходные порты рамки графа, входные порты (параметры) графа. Каждой дуге графа приписан тип (если IF1 задаёт строго типизированный язык) пересылаемого значения. 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 входными (выходными) портами составной вершины и входными (выходными) портами этих графов задаются типом (или семантикой) операции составной вершины. Структурированность вычислений, задаваемых IF1 графом, основывается на иерархичности IF1 графов, заданной посредством графов составной вершины. Ясно, что вычисления, заданные подобным образом, определяют частичный порядок над общей последовательностью выполнения вершин-операций одного IF1 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 IF1 граф допускает лёгкую интерпретируемость (исполнение), так как он задаёт потоковую модель вычислений, для исполнения которой (с некоторыми ограничениями) можно использовать даже суперкомпьютеры с потоковой архитектурой.

Реализация рассмотренных сущностей графового языка IF1 с помощью интерфейсов классов C++ получила название IR1 (Internal Representation). Нужно отметить, что важную особенность системы введенных интерфейсов, реализующих объекты графа IF1, а именно, то, что интерфейсы для реализации входных и выходных портов не различаются. Это позволяет рассматривать единый интерфейс порта с этих двух точек зрения одновременно. Теперь один и тот же порт может служить выходом графа и в то же время являться входом для дуг, принадлежащих этому графу (и наоборот). Такая унификация позволила рассматривать граф в качестве вершины в графе-владельце, ничего не знающем о внутреннем устройстве такой вершины. Таким образом, можно определить IR1 граф как объект, содержащий упорядоченные множества входных (выходов для дуг графа) и выходных (входов для дуг графа) портов и множество (с задаваемым в некоторых случаях порядком) IR1 графов, являющихся вершинами этого графа. Причем эти графы иногда будут называться подграфами относительно родительского графа, их содержащего. Можно представлять составную вершину как IR1 граф без дуг, содержащий в качестве вершин IR1 графы, соответствующие графам составной вершины. Тогда с помощью одного интерфейса IR1 графа можно реализовать несколько сущностей языка IF1: граф, вершина, составная вершина. Таким образом, внутреннее представление IR1 модуля языка SISAL можно представить в виде множества IR1 графов с пометками, соответствующих множеству функций и редукций исходной SISAL программы, и множества именованных типов, которые будут экспортированы из модуля SISAL программы. Внутреннее представление всей

программы на языке SISAL состоит из множества внутренних представлений IR1 для отдельных модулей программы.

Система SFP использует язык SISAL 3.1 и его транслятор во внутреннее представление IR1. Внутреннее представление IR1 является расширением иерархического ориентированного ациклического графа. Расширение заключается в том, что каждая вершина содержит два упорядоченных набора портов. Каждая программа на функциональном языке вычисляет некоторую функцию, и соответствующее внутреннее представление IR1 отражает способ вычисления, а именно, то, что функция вычисляется через другие функции. Иерархический граф подходит для задания такой структуры. Здесь и далее под IR1-графом подразумевается граф, соответствующий внутреннему представлению IR1. В IR1-графе каждая вершина соответствует некоторой функции. Если функция вычисляется с использованием некоторых других функций, то соответствующая вычисляемой функции IR1-вершина является составной, то есть содержит некоторый IR1-граф. Ниже будет приведено более подробное описание внутреннего представления IR1.

Программы в системе функционального программирования SFP транслируются во внутреннее представление IR1. На полученных представлениях можно проводить интерпретацию, отладку, оптимизацию и трансляцию. Всеми этими процессами полезно управлять, поэтому полезно иметь и визуальное представление программ. В настоящее время существует много систем визуализации графов (некоторые указаны здесь [9]), позволяющих получать изображения графов разных типов, в том числе ориентированных ациклических. Однако в большинстве случаев такие системы имеют слабую поддержку визуализации иерархической структуры либо совсем не поддерживают такой стиль изображения.

В данной статье предлагается алгоритм укладки IR1-графов. Для укладки неориентированных графов известно достаточно много разных подходов: силовой метод, метод концентрических окружностей и т.д. Для укладки ациклических ориентированных графов в основном используется поуровневый метод [5] и различные его модификации; ниже будет приведено общее описание данного подхода.

2. ОБЩИЙ АЛГОРИТМ УКЛАДКИ ОРИЕНТИРОВАННЫХ АЦИКЛИЧЕСКИХ ГРАФОВ

Данный подход позволяет для произвольного ациклического ориентированного графа построить поуровневое представление с дугами в виде ломаных или сплайнов, решая следующие три задачи:

a) распределение вершин по уровням таким образом, чтобы все дуги имели одно направление;

b) выбор порядка вершин на уровне, задающий минимальное число пересечений дуг;

c) определение координат вершин на уровне с целью минимизации числа изломов дуг.

Распределение вершин по уровням

Каждой вершине присваивается число, указывающее уровень вершины так, чтобы все дуги, соединяющие вершины, следовали от большего номера к меньшему номеру; при этом вершины одного уровня не должны быть соединены друг с другом. В общем случае для этого шага формулируется задача линейного программирования. После этого каждая длинная дуга, то есть соединяющая вершины с уровней, номера которых отличаются больше, чем на единицу, заменяется цепочкой фиктивных вершин таким образом, что каждая следующая вершина в цепочке лежит на уровне с номером, ровно на единицу меньшим, чем предыдущая вершина.

На данном этапе уже можно вычислить конечную вертикальную координату вершины. Ясно, что для любого ациклического графа можно найти такое распределение по уровням.

Такое построение приводит к тому, что существуют дуги, соединяющие вершины только соседних уровней. Таким образом, следующий шаг будет решать задачу переупорядочения вершин только для двух соседних уровней. Кроме того, такое решение избавляет от необходимости контролировать в дальнейшем пересечение дуг и вершин, так как при отображении фиктивные цепочки будут заменены ломаными линиями или сплайнами. Ясно, что в таком случае дуги не будут пересекать вершины.

Определение порядка вершин на уровнях

На данном этапе необходимо выбрать порядок вершин на каждом уровне. Выбор оптимального порядка позволит уменьшить число пересечений дуг. Если в графе уже нет “длинных” дуг, то число пересечений определя-

ется только порядком вершин на уровнях. После построения поуровневого разбиения встаёт задача определения порядка расположения вершин на каждом уровне графа. Задача преследует цель минимизировать число пересечения дуг. Следует заметить, что количество пересечений дуг не зависит от конечных координат вершин на уровне, а только от порядка следования вершин внутри каждого уровня. В общем случае задача является NP-полной уже только для двух уровней. Таким образом, методы, получающие точные решения, подходят только для маленьких графов, алгоритмы, вычисляющие приближённое решение, являются более предпочтительными. При этом есть несколько вариаций:

- фиксирование порядка соседних уровней, это может быть нижний уровень, или верхний и нижний уровни;
- просмотр уровня справа налево либо слева направо, также возможно чередование направления просмотра;
- критерий остановки просмотра, просмотры делаются, пока число пересечений уменьшается, либо пока не превышен заранее заданный предел числа просмотров.

Для получения первоначального порядка используется алгоритм, позволяющий избежать пересечения дуг, если бы исходный граф был деревом. Метод состоит в проведении поиска в глубину, при котором вершины с одного уровня получают номера в порядке их просмотра в ходе такого поиска.

Определение координат вершин на уровнях

На данном этапе для каждой вершины вычисляются вертикальная и горизонтальная координаты; вертикальная координата выбирается с учётом номера уровня, к которому приписана вершина, а горизонтальная выбирается так, чтобы минимизировать число изломов длинных дуг. Обычно дуги изображают в виде прямолинейного отрезка; в таком случае длинные дуги изображаются в виде ломаных линий. После определения порядка вершин на уровне необходимо определить их настоящие горизонтальные координаты. Обычно дуги графа изображаются в виде ломаных, где в точках излома находятся мнимые вершины, так что данная задача решает также задачу проведения длинных дуг. В случае наличия длинных дуг, задача нахождения точных координат решается из соображений минимизации числа изломов длинных дуг. При этом не должен быть изменён порядок следования вершин внутри каждого уровня.

3. МОДИФИЦИРОВАННЫЙ АЛГОРИТМ УКЛАДКИ ГРАФА ДЛЯ IR1-ГРАФОВ

Распределение вершин по уровням

Как уже было описано, IR1-граф представляет собой два упорядоченных множества портов и множество вершин, также являющихся IR1-графами, а дуги задаются неявно, с помощью связей между портами. Под неявным заданием подразумевается, что нет отдельной сущности как “дуга”; вместо этого порты сами хранят информацию о том, каким портам они передают данные либо от какого порта получают данные. Порты могут быть входными или выходными; в данной терминологии принято, что входные порты могут быть инцидентны только одной дуге, а выходные порты могут быть инцидентны нескольким дугам. Обход графа можно начинать либо с входных, либо с выходных портов. Однако в IR1-графе не существует вершин, которые не возвращают значения, то есть без выходных портов, но существуют вершины, которые значения только возвращают, то есть только с выходными портами. Поэтому разумно производить обход снизу или с выходных портов.

Для удобства выходным портам графа присваивается нулевой номер уровня. Процесс вычисления номеров уровней является итеративным.

На первом шаге в качестве текущего значения уровня принимается ноль. Далее находится текущее множество вершин: для каждого выходного порта графа вычисляется вершина, которой принадлежит порт, поставляющий данные для выходного порта; множество таких вершин образует текущее множество вершин. Сложность этой операции $O(P^2)$.

Если множество вершин оказалось пустым, то на этом процесс разбиения заканчивается. Если множество таких вершин не пусто, то для каждой вершины устанавливается значение уровня, равное максимуму между текущим значением уровня, увеличенным на единицу, и текущим уровнем вершины. Начальное значение уровня для всех вершин равно –1. Далее для каждой вершины вычисляется множество входных портов, а для каждого такого порта вычисляется дуга, конечным портом которой он является. Далее по дуге вычисляется её начальный порт. Объединение вершин, являющихся владельцами таких начальных портов, даёт текущее множество вершин для следующей итерации. Процесс продолжается, пока текущее множество вершин не окажется пустым. Последний шаг алгоритма разбиения таков: просматривается множество вершин, и все вершины, не получившие номера уровня, получают номер текущего уровня, увеличенный на едини-

цу. Такие вершины могут существовать, так как граф может оказаться несвязным; граф будет таким, когда программа содержит более одной функции.

Сложность этого шага $O(P^2 + V^3 * P)$. Но это очень грубая оценка, и такой вариант более точен: $O(P_{out}{}^2 + L^{\max} * (V_l^{\max})^2 * P_{in}^{\max} *)$, где P_{out} – число выходных портов графа, $L^{\max}$ – число уровней в графе, $V_l^{\max}$ – максимальное число вершин на уровне, $P_{in}^{\max}$ – максимальное число входных портов на вершинах. Однако эта оценка заранее неизвестна.

```
Algorithm
begin
   Level = 0
   Tmp = Vertices = <empty>
   Ports = IR1Graph.OutPorts
   for i := 1 step 1 until Ports.Count do
   begin
      Port := Ports[i]
      for j := 1 step 1 until Port.ParentVertices.Count do
      begin
        V := Port.ParentVertices[j]
        if (V not in Vertices) then Vertices.Add(V)
      end
    end

   while (Vertices is not empty) do
   begin
      for i := 1 step 1 until Vertices.Count do
      begin
          V :=  Vertices[i]
          for j := 1 step 1 until V.ParentVertices.Count do
          begin
            V1 := V.ParentVertices [j]
            if (V1 not in Tmp) then Vertices.Add(V1)
          end
          V.Level := max(V.Level,Level+1)
       end
      Vertices = Tmp;
```

```
            Tmp = <empty>
            Level = Level + 1
          end

        for i := 1 step 1 until IR1Graph.Vertices.Count do
        begin
            Vertex := IR1Graph.Vertices[i]
           if (Vertex not  is marked) then  Vertex.Level = Level+1
         end
      end
```

Для каждой дуги проверяется разность между уровнями начального и конечного портов, и если какой-либо из портов принадлежит вершине, то его уровень определяется уровнем вершины-владельца. Если разность оказывается больше единицы, то вместо дуги вставляется цепочка вершин со специальной пометкой, определяющей то, что эти вершины не будут показываться при конечной визуализации. Сложность операции оценивается так $O(E*V)$, более точная оценка выглядит как $O(E_{long} * L_{\max})$, где E_{long} – число "длинных" дуг, то есть тех, которые соединяют вершины с не соседних уровней, $L_{\max}$ – число уровней. Однако, опять же, эта оценка заранее неизвестна.

```
      Algorithm
       begin
          for i := 1 step 1 until IR1Graph.Edges.Count do
          begin
              Edge := IR1Graph.Edges[i]
              if (Edge.StartPortLevel - Edge.EndPortLevel > 1) then
AddChainInstead(IR1Graph,Edge)
           end
       end
```

Определение порядка вершин на уровне

В общем случае задача очень сложная, однако, в случае IR1-графов можно использовать достаточно жёсткие ограничения, которые позволяют делать некоторые предварительные заключения. Можно считать, что порядок вершин первого уровня, то есть тех, которые лежат на самом нижнем

уровне, задаётся только порядком выходных портов графа. Так как множества портов на графах являются упорядоченными, и для каждого порта существует ровно одна вершина-предок, после внесения мнимых вершин будет именно ровно одна вершина. Таким образом, порядок первого уровня однозначно вычисляется с использованием порядка выходных портов IR1-графа.

Далее будем считать, что порядок вершин предыдущего уровня зафиксирован. Так как для каждой вершины известен упорядоченный набор портов, то также известен и упорядоченный набор вершин-предков. Таким образом, если для каждой вершины предыдущего уровня выписать свой упорядоченный набор вершин-предков, причем так, что порядок наборов предков соответствует порядку вершин на предыдущем уровне, то получается некоторый упорядоченный набор вершин. Если каждая вершина в полученном наборе встречается только один раз, то можно считать, что порядок вершин следующего уровня получен, можно переходить к следующей итерации. Если существуют копии вершин, непосредственно следующие друг за другом, то такие последовательности копий вершин следует заменить на одну соответствующую вершину. После этого следует просмотреть все вершины слева направо, имеющие ровно один выходной порт, а порядок таких вершин следует зафиксировать, так как их перестановки могут только увеличить число пересечений дуг. Как только встретится вершина с большим количеством выходных портов, просмотр следует прекратить. Далее нужно произвести аналогичный просмотр справа налево, начиная с самой правой вершины. Просмотр следует прервать, если встретится вершина более чем с одним выходным портом, или если встретится вершина с фиксированным порядком. Если порядок всех вершин на уровне зафиксирован, то процедуру следует остановить и перейти к определению порядка на следующем уровне. Если остались вершины с незафиксированным порядком на уровне, то к ним применяется следующий этап итерации. Следует заметить, что для всех таких вершин известно следующее: они имеют больше одного выходного порта, и между двумя любыми вершинами нет вершины с зафиксированным порядком.

Суть дальнейшей процедуры заключается в следующем. Необходимо избавиться от копий, таким образом, чтобы число пересечений дуг было минимально, так как перед началом процедуры структура графа такова, что пересечений дуг нет, поэтому для удаления повторов будут выбираться те варианты, которые добавляют меньшее количество пересечений.

Прежде всего, для каждой уникальной вершины из обрабатываемого множества, вычисляется количество копий и минимальное расстояние до

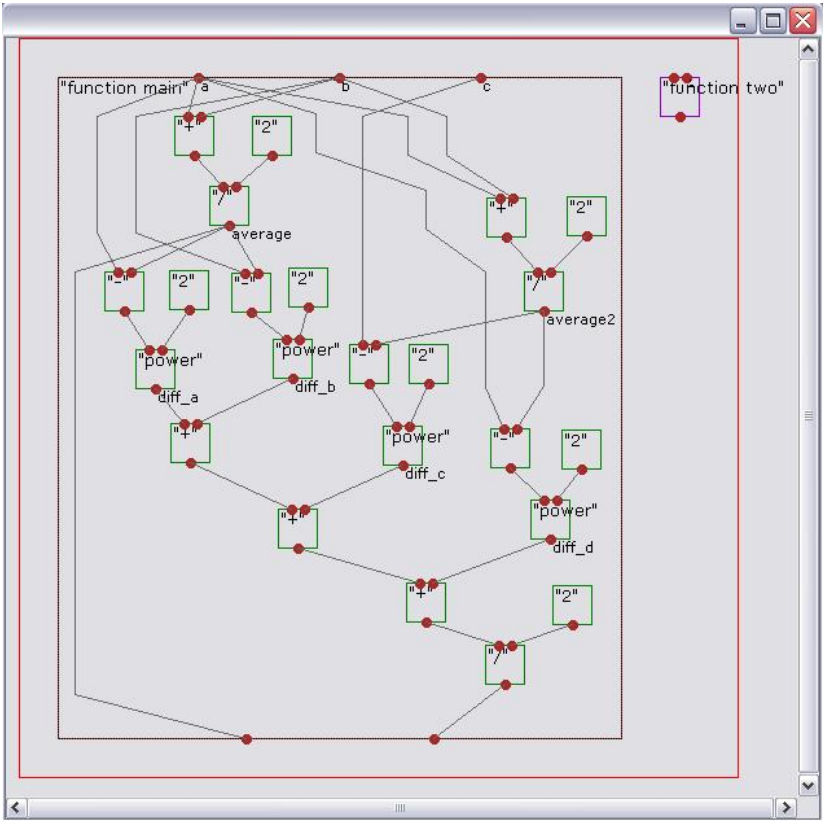
ближайшей копии. Выбор уникальной вершины для удаления следует делать, исходя из минимальности расстояния между копиями, так как это внесёт меньше пересечений. Если есть несколько копий на одинаковом расстоянии друг от друга, для удаления следует выбрать ту, удаление которой внесёт меньше пересечений. После каждого удаления необходимо пересчитывать минимальные расстояния между копиями. Процедура повторяется, пока не будут удалены все копии вершин. После этого можно переходить к обработке следующего уровня.

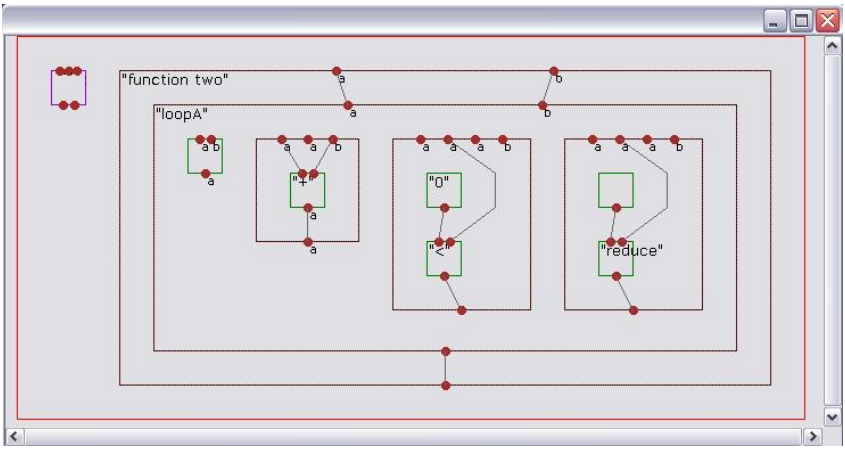
```
Algorithm
begin
   Ports := IR1Graph.OutPorts

      while(CurrentLayer is not Empty) do
      begin //1
         for i := 1 step 1 until Ports.Count do
          begin
            Port :=  Ports[i]
            CurrentLayer.Add(Port.ParentVertex)
           end

         while (true) do
         begin //2
            MergeNearCopies(CurrentLayer)

            if ( not CopiesExists(CurrentLayer) )  then
            begin
               for i := 1 step 1 until CurrentLayer.Count do
                 begin
                   Vertex :=  CurrentLayer [i]
                   for j := 1 step 1 until Vertex.InPorts.Count do
                    Port1 := Vertex.InPorts[j]
                    NewPorts.Add(Port1)
                 end
               break
            end
            else   DeleteApprprpiateVertex(CurrentLayer)

         end //2
```

```
        Ports += NewPorts
        NewPorts := <empty>
      end //1
    end
```

Очевидно, что алгоритм всегда останавливается, так как для каждого уровня генерируется конечное число копий вершин, и за каждую итерацию обязательно удаляется одна копия, а если копий больше нет, то обрабатывается следующий уровень.

Применение алгоритма





Код соответствующей SISAL-программы:

a.dec.sis

```
function main[integer,integer,integer returns integer,integer]
function two[integer,integer returns integer]
```

a.def.sis

```
definition a
```

```
function main(a:integer; b:integer; c:integer  returns integer, integer)
let average := (a + b) / 2;
   average2 := (a + b) / 2;
   diff_a  := (a - average) ** 2;
   diff_b  := (b - average) ** 2;
   diff_c  := (c - average2) ** 2;
   diff_d  := (a - average2) ** 2;
   diff_e  := two(diff_c,diff_a)
```

```
in average, (diff_a + diff_b + diff_c + diff_d)/2
end let
end function

function two(a:integer; b:integer returns integer)
  for repeat
    a := old a + b
  while a > 0
  returns value of a
  end for
end function
```

ЗАКЛЮЧЕНИЕ

В статье описывается алгоритм укладки иерархических ациклических ориентированных графов специального вида, соответствующих внутреннему представлению программ на функциональном языке SISAL 3.1. С использованием алгоритма реализован компонент визуализации, позволяющий получать изображения внутреннего представления и осуществлять некоторые манипуляции с изображением: изменение масштаба, перемещение вершин, перемещение по графу, сворачивание-разворачивание составных вершин.

СПИСОК ЛИТЕРАТУРЫ

1. **Касьянов В. Н., Евстигнеев В. А.** Графы в программировании: обработка, визуализация и применение. – СПб.: БХВ-Петербург, 2003. – 1104 с.
2. **Kasyanov V. N., Stasenko A. P.** A Functional Programming System SFP: Sisal 3.1 Language Structures Decomposition // Parallel Computing Technologies – Lect. Notes Comput. Sci. – Springer, 2007. – Vol. 4671. – P. 62–73.
3. **Стасенко А. П., Синяков А. И.** Базовые средства языка SISAL 3.1. – Новосибирск, 2006. – 56 с. – (Препр. / РАН. Сиб. Отд-ние. ИСИ; № 132)

4. **Стасенко А. П.** Внутреннее представление системы функционального программирования SISAL 3.0. – Новосибирск, 2004. – 54 с. – (Препр. / РАН. Сиб. Отд-ние. ИСИ; № 110)
5. **Sugiyama K., Tagawa S., Toda M.** Methods for Visual Understanding of Hierarhical System Structures // IEEE Transactions on Systems, Man and Cybernetics. – 1981. – Vol. 11, Iss. 2. – P. 109–125.
6. **Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, Kiem-Phong Vo**. A Technique for Drawing Directed Graph // IEEE Transactions on Software Engineering. – 1993. – Vol. 19 , Iss. 3. – P. 214–230.
7. **Juonger M., Mutzel P.** 2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms. – 1997. – Vol. 1, N 1. – P. 1-25.
8. IF1: an intermediate form for applicative languages / Skedzielewski S., Glauert J — Livermore, CA, 1985. — (Lawrence Livermore National Laboratory; M-170).
9. **Лисицын И. А.** Системы визуализации и редактирования графовых объектов: обзор. – Новосибирск, 2000. – 42. с. – (Препр. / РАН. Сиб. Отд-ние. ИСИ; № 76).
10. **Гордеев Д. С.** Визуализация в системе функционального программирования SFP. Технологии Microsoft в теории и практике программирования // Тез. докл. Конф.-конкурса работ студентов, аспирантов и молодых учёных. – Новосибирск, 2007. – С. 103.
11. **Гордеев Д.С**. Визуализация внутреннего представления программ в системе функционального программирования SFP // Материалы II Международной научной студенческой конференции «Научный потенциал студенчества – будущему России». Том третий. Математика. – Ставрополь, 2008. – С. 180.