ПРОГРАММИРОВАНИЕ, 2019, № 4, с. 19–27

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.921

МЕТОДЫ И АЛГОРИТМЫ ВИЗУАЛИЗАЦИИ ГРАФОВЫХ ПРЕДСТАВЛЕНИЙ ФУНКЦИОНАЛЬНЫХ ПРОГРАММ

© 2019 г. В. Н. Касьянов[a],*, Т. А. Золотухин[a],, Д. С. Гордеев[a],*****

[a] *Институт систем информатики им. А.П. Ершова СО РАН*
630090 Новосибирск, пр. Академика Лаврентьева, д. 6, Россия

**e-mail: kvn@iis.nsk.su*

***e-mail: tzolotuhin@gmail.com*

****e-mail: ainty@inbox.ru*

Поступила в редакцию 25.01.2019 г.
После доработки 25.01.2019 г.
Принята к публикации 16.02.2019 г.

Статья посвящена задаче визуализации в облачной системе параллельного программирования CPPS внутренних представлений программ на языке функционального программирования Cloud Sisal в виде так называемых IR-графов. IR-граф выражает поток данных в функциональной Cloud Sisal программе и является иерархическим графом, вершины которого соответствуют простым и составным выражениям программы и соединены дугами через порты вершин, соответствующие операндам выражений: аргументам и результатам. В статье рассмотрен эффективный алгоритм построения наглядных изображений IR-графов на плоскости. Описана эффективная реализация в рамках системы Visual Graph алгоритма укладки на плоскости произвольного атрибутированного иерархического графа с портами. Рассмотрены методы наглядного визуального представления IR-графов и их динамических изменений, связанных с отладкой Cloud Sisal программ.

DOI: 10.1134/S0132347419040034

1. ВВЕДЕНИЕ

Используя традиционные языки и методы, очень трудно разработать высококачественное, переносимое программное обеспечение для параллельных компьютеров. В частности, параллельное программное обеспечение не может быть разработано с малыми затратами на последовательных компьютерах и потом перенесено на параллельные вычислительные системы без существенного переписывания и отладки. Поэтому высококачественное параллельное программное обеспечение может разрабатываться только небольшим кругом специалистов, имеющих прямой доступ к дорогостоящему оборудованию.

Однако, используя языки программирования с неявным параллелизмом, такие как функциональный язык Cloud Sisal [1], можно преодолеть этот барьер и предоставить широкому кругу специалистов, которые не имеют доступа к параллельным вычислительным системам, но могут многое сделать в своих прикладных областях исследований, возможность быстрой разработки переносимых параллельных алгоритмов на своем рабочем месте.

Семантика языка функционального программирования Cloud Sisal гарантирует детерминированные результаты для параллельной и последовательной реализаций – то, что невозможно гарантировать для традиционных императивных языков, подобных языку Фортран или С. Более того, неявный параллелизм языка Cloud Sisal снимает необходимость переписывания исходного кода при переносе его с одного компьютера на другой. Гарантировано, что Cloud-Sisal-программа, правильно исполняющаяся на персональном компьютере, будет также давать правильные результаты на высокоскоростном параллельном или распределенном вычислителе.

В статье рассматриваются методы и алгоритмы визуализации графовых представлений программ на языке Cloud Sisal в рамках облачной системы параллельного программирования CPPS (Cloud Parallel Programming System). CPPS – это система, которая разрабатывается как интегрированная облачная среда программирования на языке Cloud Sisal, которая содержит как интерпретатор, поддерживающий диалоговое взаимодействие с пользователем при построении и отладке программы, так и оптимизирующий кросс-компилятор, осуществляющий построение параллельной программы по ее функциональной спецификации [2].

Система CPPS использует внутреннее (промежуточное) графовое представление IR (Intermediate Representation) для Cloud Sisal программ, которое ориентировано на их семантическую и визуальную обработку и основано на так называемых иерархических графах. Иерархический граф помимо вершин некоторого графа, называемого базисным, содержит также вершины, соответствующие некоторым частям базисного графа (фрагментам), множество которых образует иерархию по вложенности (дерево) и содержит весь базисный граф (в качестве корня данного дерева) [3, 4].

IR-графы Cloud Sisal программ в отличие от управляющих графов, обычно используемых в оптимизирующих компиляторах для императивных языков (таких как языки C или Фортран) [5, 6], выражают не потоки управления, а потоки данных в программах, и обладают рядом полезных свойств, включая следующие.

1. Явно заданные информационные (семантические) связи (дуги) между операндами операций (портами вершин) делают процесс интерпретации осуществимым без дополнительных преобразований. Это влечет отсутствие побочных эффектов вычислений (ввиду отсутствия понятия переменной) – естественного свойства чисто функциональных языков.

2. Представлен параллелизм на уровне отдельных информационно независимых операций, который не зависит от машинной архитектуры.

Предполагается, что графовые представления Cloud-Sisal-программ должны демонстрироваться пользователям системы CPPS наряду с их текстовыми представлениями, и могут использоваться ими для целей визуальной отладки Cloud-Sisal-программ и управляемой их оптимизации.

Возникает необходимость автоматического построения наглядного изображения IR-графа Cloud-Sisal-программы, а также наглядного визуального отображения в нем процессов ее исполнений. Данная задача является нетривиальной, так как основным пользователем системы является человек, для которого построенное изображение IR-графа и демонстрируемые его изменения должны обладать рядом свойств, упрощающим процесс понимания пользователем функциональной программы и приводящим к корректному и более полному его представлению о том, как происходит вычисление в данной программе. Часть из этих свойств образует обязательные требования к виду изображения, а другая часть имеет вид так называемых эстетических критериев, по которым следует улучшать изображение.

В настоящее время вопросам визуализации графов и графовых моделей посвящена обширная литература (см., например, [7, 8]), а на рынке широко представлены наукоемкие программные продукты, использующие методы визуализации информации на основе графовых моделей. В основном это многочисленные специализированные системы или встроенные компоненты систем, ориентированные на визуализацию определенных подклассов графов, но есть и универсальные системы, предназначенные для визуализации графов общего вида и широкого назначения, такие как aiSee, yEd, Higres, Cytoscape и Visual Graph.

Основные трудности решения построения наглядного изображения IR-графов связаны с тем, что в отличие от стандартной задачи укладки графа на плоскости [9, 10, 11] вершины IR-графа соединяются дугами через свои порты, а также то, что они могут иметь разный размер, поскольку вершины-фрагменты должны содержать изображения тех частей графа, которые им соответствуют. Причем, если вопросы укладки графов с вершинами разного размера ранее исследовались и в большинстве универсальных систем визуализации существуют алгоритмы укладки для некоторых подклассов так называемых простых иерархических графов, то задача укладки графа с портами является новой.

Методы визуализации процесса вычислений зависят от решаемой задачи и ее предметной области, в данном случае это предметная область графов потока данных, где результатом визуализации в том или ином виде является последовательность изображений состояния графовой модели со всеми графически отображаемыми атрибутами и структурой. Другими словами, результатом является динамический атрибутированный иерархический граф с портами, а визуализация динамических графов исследовалась только для графов, в которых нет фрагментов и портов, и проводилась в основном как автономная (offline) или интерактивная (online) [12]. В первом случае строится последовательность изображений графа, при визуализации которой между двумя соседними изображениями используются техники интерполяции для получения промежуточного изображения, и набор таких промежуточных изображений описывает трансформацию между двумя соседними автономными изображениями графа. Во втором случае используется динамическое вычисление очередного изображения исходя из текущего состояния графовой модели и следующего события, которое следует графически отобразить.

Структура статьи следующая. В разделе 2 описываются IR-графы и их изображения. Раздел 3 содержит общую схему алгоритма укладки IR-графа на плоскости, основные этапы которого, связанные с распределением вершин по уровням, выбором порядков расположения вершин на уровнях и определением координат вершин на каждом уровне, рассмотрены в разделах 4, 5 и 6 соответственно. Реализация алгоритма укладки в рамках системы Visual Graph описана в разделе 7. Раздел

8 содержит описание модели изменений IR-графа, а вопросы ее реализация описаны в разделе 9.

2. IR-ГРАФЫ И ИХ ИЗОБРАЖЕНИЯ

Вершины IR-графа соответствуют выражениям программы, а дуги отражают передачи данных между портами вершин, упорядоченные множества которых приписаны вершинам в качестве их аргументов (входных портов или входов) и результатов (выходных портов или выходов). В силу свойства языка Cloud Sisal этот граф является ациклическим[1] и не содержит двух дуг, заходящих в один и тот же вход.

Вершины обозначают операции над своими входами (аргументами), результаты которых находятся на выходах вершин. Существует, однако, специальный вид вершин, обозначающих литералы (константы) различных языковых типов, каждая из которых имеет один выход и пустое множество входов.

Вершины бывают простыми и составными. Простые вершины (или просто вершины) не имеют внутренней структуры помимо ассоциированной с ними операции. Составные вершины (или фрагменты) соответствуют составным выражениям Cloud Sisal программы, таким, как, например, циклическое выражение или тело функции, и дополнительно непосредственно содержат упорядоченные множества вершин, соответствующих выражениям, из которых они состоят. Для каждого фрагмента F количество и типы того множества вершин M, которые непосредственно содержатся в нем, а также того множества дуг (p, q), которые существуют между его портами и портами этих вершин, задаются типом (или семантикой) составного выражения, но удовлетворяют следующему свойству: p – выход некоторой вершины из M или вход фрагмента F, а q – вход некоторой вершины из M или выход фрагмента F. Понятно, что фрагмент F помимо указанных выше вершин M и инцидентных их портам дуг, которые непосредственно содержатся в F, будет содержать и другие вершины и дуги графа, если среди вершин из M есть фрагменты, но в силу свойств языка Cloud Sisal среди них нет дуг, которые инцидентны портам фрагмента F и не являются непосредственно вложенными в F.

IR-графы являются атрибутированными. Каждый атрибут состоит из уникального имени атрибута и его значения. Атрибуты сопоставлены элементам (вершинам, дугам и портам) графа и используются не только для выражения семантики Cloud Sisal программы, но для ее визуальной обработки. Для составных вершин, например, именем атрибута может быть “тип вершины” со значениями “цикл”, “функция” и др., а также атрибут “цвет границы” со значениями цвета соответственно.

Предполагаются следующие правила изображения IR-графов:

1. Простые вершины без входов изображаются в виде прямоугольников, содержащих обозначения констант.

2. Простые вершины с входами изображаются в виде прямоугольников с прямоугольными выступами сверху, изображающими входы вершины в их упорядоченности слева направо, и прямоугольными выступами снизу, изображающими выходы вершины. Внутри прямоугольников находятся обозначения операций.

3. Составные вершины изображаются в виде прямоугольников с полосой слева для указания типа составной вершины, а также с прямоугольниками сверху и снизу для изображения входов и выходов этой вершины в их упорядоченности слева направо. Каждый прямоугольник, изображающий некоторый порт фрагмента, состоит из прямоугольного выступа наружу и прямоугольного выступа внутрь этого прямоугольника. Внутри прямоугольника, изображающего составную вершину, находятся изображения всех вершин и всех дуг, в ней содержащихся, и только они.

4. Изображения двух разных вершин либо не пересекаются, либо одно из них целиком лежит в другом.

5. Дуги изображаются в виде кривых (сплайнов) со стрелками, которые соединяют соответствующие порты и не пересекают изображения вершин и портов по своим внутренним точкам.

3. ОБЩАЯ СХЕМА АЛГОРИТМА УКЛАДКИ

Ниже при описании алгоритма укладки предполагается, что исходный граф является дэгом, т.е. ациклическим ориентированным графом. Если исходный граф является неориентированным и/или содержит циклы, то он сначала всегда может быть приведен к дэгу путем задания и/или смены ориентации у части его ребер и дуг, а затем по построенной укладке дэга обратным преобразованием, примененным к изображениям измененных дуг, можно будет получить укладку исходного графа [11, 6].

Алгоритм состоит в последовательном выполнении шагов построения изображений содержимого фрагментов исходного графа, начиная с самого внутреннего, на каждом из которых происходит укладка некоторого фрагмента с использованием размеров и расположений непосредственно вложенных в него вершин и дуг.

Само построение изображения одного текущего фрагмента базируется на методике так на-

[1] Здесь и ниже без определений используются стандартные термины из теории графов (см., например, [6]).

зываемого поуровневого подхода к построению укладки ациклического ориентированного графа, которая была предложена K. Сугиямой (K. Sugiyama), и состоит из следующих трех основных этапов [11, 6]:

1. Распределение вершин по уровням так, чтобы дуги следовали одному направлению, т.е. определение y-координат для вершин.

2. Выбор порядка вершин на уровне с целью минимизации пересечений дуг.

3. Определение x-координат вершин на уровне с целью минимизации общей длины дуг и количества сгибов.

Таким образом, данный подход позволяет разбить задачу нахождения расположения ациклического графа на три достаточно независимых подзадачи, решение каждой из которых может опираться на свои методы и улучшать изображение по тем или иным эстетическим критериям.

При размещении ациклических графов обычно во внимание принимается следующие эстетические критерии: совпадающее направление дуг; минимизация площади изображения; равномерность распределения вершин по площади всего изображения; уменьшение числа пересечений дуг; отсутствие слишком длинных дуг; уменьшение числа сгибов дуг.

Одновременная оптимизация изображения по всем перечисленным критериям является невозможной, т.к. критерии зачастую оказываются противоречащими друг другу. Например, уменьшение числа сгибов дуг может нарушать равномерность распределения вершин и приводить к увеличению общей площади изображения.

4. РАСПРЕДЕЛЕНИЕ ВЕРШИН ПО УРОВНЯМ

Задачей данного этапа является присваивание каждой вершине ее конечной вертикальной y-координаты. Для этого исходный ациклический граф $G = (V, E)$ должен быть приведен к поуровневому представлению, которое есть разбиение V на подмножества $L_1, L_2, \ldots, L_h$, так что для каждой дуги $(u,v) \in E$, где $u \in L_i$ и $v \in L_j$, верно то, что $i < j$. Подразумевается, что все вершины одного уровня будут расположены на одной горизонтальной прямой. Высотой представления называется количество уровней h, а шириной w_i – наибольшее число вершин на одном уровне L_i. Зазором дуги (u,v), где $u \in L_i$ и $v \in L_j$, называется разность $j - i$. Представление называется сжатым, если не существует дуги с зазором, большим единицы. После разбиения вершин по уровням всем вершинам одного уровня присваивается одна и та же вертикальная y-координата, т.е. $v_y = h - i$ для всех вершин v с уровня L_i.

Существование поуровневого представления для любого ациклического графа очевидно. Однако не для всякого ациклического графа существует его сжатое представление – это уже имеет место для графа, состоящего всего из трех вершин и трех дуг. Необходимость же построения сжатого разбиения диктуется следующими двумя причинами. Во-первых, сжатое разбиение минимизирует горизонтальную длину дуг и количество их сгибов. При сжатом разбиении дуги связывают вершины только соседних уровней и могут быть изображены прямолинейными отрезками. Во-вторых, сжатое разбиение существенно упрощает задачу минимизации пересечений дуг графа, которая решается на втором этапе алгоритма. Сжатое разбиение сводит эту глобальную задачу к задаче минимизации пересечений для двух соседних уровней.

Для построения сжатого разбиения произвольного ациклического графа применяется техника вставки фиктивных вершин. Фиктивные вершины добавляются в граф вдоль длинных дуг, т.е. дуг с зазором, большим единицы. Каждая дуга (u,v), где $u \in L_i$ и $v \in L_j$, с зазором $k = j - i > 1$ заменяется на $k - 1$ фиктивных вершин $u_1, \ldots, u_{k-1}$, таких, что $u_l \in L_{i+l}$ для всех l, и k дуг (u,u_1), (u_{k-1},v) и (u_l,u_{l+1}), где $1 \le l < k - 1$. Такое построение приводит к тому, что в графе остаются только дуги, соединяющие вершины соседних уровней, и задача минимизации пересечения дуг сводится к соответствующей задаче для двухуровневого случая. Данное построение заодно решает и задачу проведения дуг в конечном изображении. Дуга изображается в виде сплайна, концами которого являются сами вершины, а опорными точками (точками сгиба) – вставленные на первом этапе фиктивные вершины. Таким образом, исчезает возможность пересечения дуг по внутренним точкам с вершинами и портами.

Выполнение данного этапа связано с обработкой графа $G = (V, E)$, представляющего некоторый фрагмент F, в котором V состоит из портов фрагмента F и вершин, непосредственно содержащихся в F. Пусть t – длина самого длинного пути по G. Тогда будет построено поуровневое представление $L_1, L_2, \ldots, L_{t+1}$, в котором L_1 – все входы F, а L_{t+1} – все выходы F.

Сначала из портов фрагмента строятся L_1 и L_{t+1} и эти порты вместе с инцидентными дугами удаляются из G. Процесс построения продолжается по шагам, на каждом из которых одна из вершин, не имеющая заходящих дуг в текущем состоянии G, включается в множество L_{i+1}, где i – максимальный номер множества L_i, содержащего ее

предшественника в исходном графе G, с одновременным удалением этой вершины из G вместе со всеми исходящими из нее дугами. Причем, среди вершин, не имеющих заходящих дуг в текущем состоянии G, выбирается и включается в соответствующее множество та вершина, у которой было наименьшее число входящих дуг и наибольшее число исходящих дуг в исходном графе G (эти числа предварительно подсчитываются для всех вершин исходного графа). Этот процесс продолжается пока текущий граф G не станет пустым. После этого приводится построенное представление графа G к сжатому виду, используя технику фиктивных вершин.

5. ВЫБОР ПОРЯДКА ВЕРШИН НА УРОВНЕ

Задачей данного этапа является нахождение порядка вершин на каждом уровне, с целью минимизации количества пересечений дуг.

Следует отметить, что количество пересечений дуг в поуровневом представлении графа не зависит от конечных горизонтальных координат вершин, а зависит только от их относительного положения внутри каждого уровня (их порядкового номера на данном уровне). Таким образом, задача данного этапа является не геометрической, а всего лишь комбинаторной. Однако эта задача является NP-полной уже для графа, имеющего всего лишь два уровня.

Выполнение данного этапа для заданного фрагмента осуществляется следующим образом:

1. Если у фрагмента есть входные и/или выходные порты, размещенные на уровне L_1 и/или L_h соответственно, то следует приписать этим портам соответствующие им порядковые номера.

2. Рассматриваем вершины уровня L_1 в их упорядоченности, если у фрагмента нет портов, или L_2, если порты у него есть, и осуществляем обход в глубину содержимого фрагмента, начиная с этих вершин, с использованием стека.

3. Если стек пуст, то либо продолжаем шаг 2, либо данный этап завершен. В противном случае рассматриваем вершину, находящуюся на верху стека, но не удаляем ее из стека. Если не получивших номеров преемников у рассматриваемой вершины нет, то присваиваем вершине текущий номер порядка, удаляем ее из стека и переходим на шаг 3. Если же такие преемники у вершины есть, то добавляем одну из них в стек и переходим на шаг 3; при выборе преемника для размещения в стек учитываем следующее:

• Если все преемники данной вершины соединены дугами, исходящими из разных портов, то порядок включения этих вершин не противоречит порядку портов.

• Если среди преемников есть вершины, связанные с вершинами, которые уже получили порядковые номера, то они включаются раньше других.

• Если среди преемников есть фиктивные вершины, то они выбираются в таком порядке, чтобы они разместились на уровне посередине, а реальные вершины на краях уровня.

6. ОПРЕДЕЛЕНИЕ КООРДИНАТ ВЕРШИН НА УРОВНЕ

После определения порядка вершин на уровне необходимо определить их реальные координаты. Дуги графа изображаются в виде кривых с опорными точками, находящимися в фиктивных вершинах, поэтому задача определения окончательных координат всех вершин одновременно является и задачей проведения дуг. Если же дуги графа имеют какую-либо другую форму и/или способ проведения, то соответствующие критерии должны быть рассмотрены при решении задачи об определении координат вершин.

На входе данного этапа имеем разбиение вершин по уровням $L_1, L_2, \ldots, L_h$, вершины на каждом уровне имеют порядок от 1 до w_i, где i изменяется от 1 до h. Причем самое большое количество вершин находится на уровне L_1 (или на уровне L_{h-1}, если выходных портов нет).

Начиная с последнего уровня, используя метод барицентров в сочетании с ограничениями на порядок вершин, полученными на предыдущем шаге, определяем x-координаты вершин на предыдущем уровне.

Вершины последнего уровня распределяются равномерно на некотором отрезке горизонтальной прямой, выделенной под вершины этого уровня. Затем для каждого следующего уровня координаты его вершин последовательно определяются как среднее арифметическое координат их соседей из уже обработанных уровней. При этом, не допускается нарушение изначального порядка вершин на уровне.

Если на каком-то шаге происходит наслоение, т.к. двум вершинам присвоится одна x-координата, то следует немного раздвинуть последний уровень. Например, предположим, что изначально вершины на этом уровне имеют x-координаты 0, 1, 2, 3, 4 и т.д. При следующем шаге они будут иметь x-координаты 0, 2, 4, 6, 8 и т.д. А если снова не получится уложить граф, то можно еще кратно увеличить основание: 0, 3, 6, 9 и т.д.

7. РЕАЛИЗАЦИЯ АЛГОРИТМА

При описании алгоритма предполагалось, что исходный граф является ациклическим ориентированным графом. Однако его реализация в рам-

ках системы Visual Graph [13] выполнена для более общего случая и включает также шаги предварительной и окончательной обработки графа, позволяющие системе работать не только с дэгами. Суть и цель этих преобразований заключается в обратимом преобразовании структуры исходного графа, так чтобы после размещения дэга, возможно было безболезненно для качества получаемого изображения восстановить структуру исходного графа и, тем самым, построить его изображение.

Если исходный граф содержит неориентированные ребра, то в качестве предварительного шага реализовано задание им направления в соответствии с обходом графа в глубину с удалением ориентации при окончательной обработке.

Если исходный граф содержит циклы, то происходит изменение ориентации у части дуг, входящих в цикл, а затем в построенной укладке дэга обратным преобразованием строится изображение исходного графа.

Поскольку задача нахождения минимального числа дуг, изменение ориентации которых переводит граф в дэг, является NP-полной, нами используется простой метод нахождения таких дуг: в процессе обхода в глубину выделяются все обратные дуги, у которых и происходит смена ориентации. И хотя у произвольного графа $G = (V, E)$ обратных дуг может быть достаточно много (порядка $|E| - |V| - 1$), для графов программ, на которые ориентирована система, это не имеет значения, поскольку у таких графов (в частности, для управляющих графов программ) есть четко выраженное направление и, как правило, если и есть, то незначительное количество обратных дуг, образующих циклы.

Если граф содержит вершины с более чем одним предшественником, то у каждой из них оставляем только одного по следующему правилу. Если среди предшественников вершины есть только одна фиктивная или только одна не фиктивная вершина, оставляем ее одну. Иначе ищем общего предшественника для предшественников данной вершины (в случае необходимости создаем фиктивного предшественника) и прокладываем путь от него до нашей вершины.

Предполагается, что система Visual Graph взаимодействует с интерпретатором и кросс-компилятором системы CPPS через представление IR-графа Cloud-Sisal-программы на языке GraphML [14], который позволяет задавать атрибутированные иерархические графы с портами произвольного вида.

8. МОДЕЛЬ ИЗМЕНЕНИЙ IR-ГРАФА

Визуальное изображение IR-графа G строится в зависимости от структуры графа, а также атрибутов и их значений. При наличии такой зависимости отражать ход вычислений на графе можно с помощью изменения его структуры или изменения значений атрибутов его элементов (вершин, дуг и портов). В этом случае можно говорить о динамическом графе DG [15], где DG это последовательность графов G_i, где каждый граф G_{i+1} получается из G_i через преобразование $G_{i+1} = \phi(G_i)$, где ϕ осуществляет некоторую теоретико-графовую операцию, например, удаление или добавление ребра или дуги, а также изменение значения некоторого атрибута вершины, дуги или порта.

Каждое изменение в графе DG, соответствующее преобразованию ϕ, можно описать как событие изменения [15] в графовой модели G. Определим каждое такое событие как кортеж (e, n, v_1, v_2, t, c), где e – это элемент графа G, n – это имя атрибута элемента e, v_1 – это предыдущее значение атрибута с именем n, v_2 – это новое значение атрибута с именем n, t – это момент времени, когда произошло изменение значения атрибута с именем n, а c – это кортеж контекстной информации, представляющий собой пару (a, b), где a обозначает признак либо входа либо выхода из контекста, b – это контекстная информация, например, множество вершин, множество дуг, или целый подграф исходного графа. Рассматриваются следующие события изменений:

1. Добавление вершины в граф. Например, это может соответствовать добавлению функции или выражения в текст программы или добавлению вершины во вспомогательные структуры данных, например стек или очередь.

2. Добавление порта к вершине. Например, это может соответствовать добавлению аргумента в список аргументов функции или добавлению дуги во вспомогательные структуры данных.

3. Добавление дуги. Например, это может соответствовать использованию результатов функции в выражении.

4. Удаление вершины.

5. Удаление порта.

6. Удаление дуги.

7. Изменение значения атрибута вершины, порта или дуги.

Кроме них рассматриваются также следующие события:

1. Вычисление предиката на вершине.

2. Вычисление предиката на порту.

3. Вычисление предиката на дуге.

4. Перебор смежных вершин с предикатом или без.

5. Перебор смежных дуг с предикатом или без.

6. Получение предка по ориентированной дуге.

7. Получение смежных дуг с предикатом или без.

8. Получение смежных вершин с предикатом или без.

9. Получение элемента из множества по предикату.

10. Чтение значения атрибута вершины, дуги или порта.

11. Вход или выход из контекста.

Модель обрабатывает полученный поток событий изменений, сверяясь со своим внутренним состоянием, и преобразовывая каждое изменение в серию визуальных эффектов. Простой визуальный эффект можно описать как непрерывную функцию $F : [0,1] \to S$, где S – это множество троек (l, w, c), описывающих визуальное состояние геометрических линий, где l – это кортеж точек, задающий форму линии, а w и c – это толщина и цвет линии.

Составные визуальные эффекты представляют собой последовательности простых эффектов, где каждый следующий эффект запускается после предыдущего. Таким образом составной эффект можно описать как функцию:

$$F^c(t) = \begin{cases} F_1\left(t \times \frac{1}{|d_1|}\right), & t \in d_1 \\ F_2\left(t \times \frac{1}{|d_2|}\right), & t \in d_2, \\ \ldots \\ F_n\left(t \times \frac{1}{|d_n|}\right), & t \in d_n \end{cases} \qquad (8.1)$$

где $d_1, d_2, \ldots, d_n$ – это некоторое разбиение отрезка [0, 1] на последовательность попарно непересекающихся интервалов.

Таким образом, если требуется отображать изменение значения атрибута N некоторой вершины от значения A до значения B в течение, например, двух секунд, то для этого можно использовать функцию $F_N(t)$, где $F_N(0) = A$, а $F_N(1) = B$, причем для получения промежуточных значений использовать выражение $F_N\left(\frac{x}{2000}\right)$, где x меняется от 0 до 2000 миллисекунд.

При отображении изменений значений атрибутов элементов графовой модели, для которого используются функции визуальных эффектов, также полезно отображать контекстную информацию, соответствующую контексту, где происходит изменение значения атрибута. Так, например, при перечислении смежных вершин для заданной вершины происходит последовательный перебор элементов из множества смежных вершин, причем для каждой отдельно итерации перебора, множество смежных вершин является контекстной информацией.

Для отображения контекстной информации также применимы визуальные эффекты с тем замечанием, что контекстная информация отображается графически в течение всего времени жизни контекста. Например, множество смежных вершин графически выделено все время, пока производится перебор смежных вершин.

Другими примерами контекстной информации могут служить отображение смежной вершины из множества в рамках итерации цикла, который осуществляет перебор, или отображение подграфа, на котором производится вычисление внутри подпрограммы, если программа использует подпрограммы для осуществления вычислений.

9. РЕАЛИЗАЦИЯ ДИНАМИЧЕСКОЙ МОДЕЛИ

Реализация предложенной модели визуализации предназначена для использования в качестве компонента подсистемы отладки в системе CPPS, который интегрируется с компонентом, интерпретирующем отлаживаемую Cloud Sisal программу, и использует его в качестве одного из источников событий, отражающих изменения в графовой модели внутреннего представления Cloud Sisal программы. Предложенная модель позволяет отображать поток событий, отражающих изменения графовой модели, в множество взаимозависимых анимаций над визуальными образами элементов графа, а также осуществлять приостановку анимаций, и запуск анимаций в обратном направлении. Данные возможности достигаются с использованием модели изменения значения атрибута, реализованной следующим образом. При визуализации изменения значения v_1 некоторого атрибута на значение v_2 существует три возможных состояния: а) значение равно v_1, б) значение изменяется, в) значение равно v_2. При возникновении команды “установить значение v_2”, когда значение равно v_1, модель переходит в состояние “состояние изменяется” и создается команда “начать анимацию”, отражающую изменение значения. Когда анимация завершается, происходит изменение состояния модели из “значение изменяется” в “значение равно v_2”. Аналогичным образом происходит изменение значения в обратном направлении из v_2 в v_1, если это требуется при визуализации, или если визуализация переключена в обратном направлении.

Например, для дуги IR-графа, назначение которой передавать значения из выходного порта во входной порт, значение v_1 означает ожидание появления значения в выходном порту, а промежуточное состояние означает передачу значения, а значение v_2 означает значение передано во входной порт. Соответственно для порта значение v_1 означает отсутствие значения в порту, промежуточное состояние означает заполнение порта, чтобы процесс заполнения мог быть замечен

пользователем, а значение v_2 означает заполненность порта. Аналогично для вершины значение v_1 означает ожидание заполнения всех входных портов, чтобы начать вычисление соответствующей функции, промежуточное состояние означает процесс вычисления соответствующей функции, а значение v_2 означает завершение вычисления и заполнение всех выходных портов вершины вычисленными значениями. Таким образом модель изменения значений атрибутов применима к вершинам, портам и дугам.

Так, например, для дуги команда "установить значение v_2^{edge}" соответствует состоянию "значение равно v_2^{port}" выходного порта, инцидентного дуге, а анимация передачи значения может отображаться с помощью утолщенной серой линии, расположенной поверх тонкой темной линии, изображающей соответствующую дугу.

При графическом отображении графов обычно требуется тем или иным способом решить задачу определения размеров вершин графа и укладки графа на плоскость. В случае, если при графическом отображении изменений в графовой модели изменения связаны с добавлением вершин, портов или дуг, то задачу укладки приходится решать на лету. Существенной трудностью здесь является необходимость как можно полного сохранения относительного расположения элементов графа до и после изменения. С другой стороны, та динамика, которая может возникнуть в процессе отладки потоковой программы, не связана с существенными изменениями структуры IR-графа программы. Поэтому нами выбран следующий метод, который существенно упрощает указанную проблему и в нашем случае вполне применим.

Графовая модель внутреннего представления модифицируется добавлением служебного атрибута ко всем элементам графа, обозначающим "видимость". Если значение атрибута задано, то вершина должна быть отображена графически, если не задано, то соответственно вершина должна быть невидима. Также меняется логика вычислений над множествами вершин, портов и дуг графа так, что, если значение атрибута видимости задано, то вершина при операциях перечисления элементов в множестве должна быть доступна для перечисления, а если значение не задано, соответственно, вершины должна быть недоступна для перечисления. Также меняется алгоритм укладки графой модели на плоскость, и вычисление будет проходить по следующей схеме:

1. Потребуется взять всю последовательность графов из динамического графа *DG*, объединить множества вершин всех элементов последовательности, а также множества портов для каждой вершины и множества дуг.

2. Применить алгоритм укладки.

3. Пометить все элементы первого графа из последовательности *DG* как видимые с помощью атрибута "видимости".

Соответственно, при отображении событий добавления или удаления вершин, вместо удаления проставляется или очищается значение атрибута видимости.

10. ЗАКЛЮЧЕНИЕ

В статье были рассмотрены методы и алгоритмы визуализации графовых представлений программ на функциональном языке Cloud Sisal в рамках облачной системы параллельного программирования CPPS.

Впервые рассмотрена и решена задача построения наглядных изображений представления Cloud-Sisal-программ в виде IR-графов. Описана эффективная реализация в рамках системы Visual Graph алгоритма укладки на плоскости произвольного атрибутированного иерархического графа с портами. Впервые рассмотрена и решена задача визуализации атрибутированных динамических графов с фрагментами и портами. Описаны методы наглядного визуального представления IR-графов и их динамических изменений, связанных с отладкой Cloud Sisal программ.

Предложенный алгоритм укладки графов с портами и фрагментами имеет квадратичную временную сложность и позволяет строить наглядные изображения графовых представлений функциональных Cloud-Sisal-программ. Реализация алгоритма в рамках системы Visual Graph также достаточно эффективна, поскольку она позволяет на обычном персональном компьютере за реальное время (без видимых задержек) визуализировать произвольного вида атрибутированные иерархические графы с портами, содержащие порядка 10000 элементов (вершин и дуг).

11. БЛАГОДАРНОСТИ

Исследование выполнено за счет гранта Российского научного фонда (проект 18-11-00118).

СПИСОК ЛИТЕРАТУРЫ

1. *Касьянов В.Н., Касьянова Е.В.* Язык программирования Cloud Sisal. Препринт ИСИ СО РАН. Новосибирск. 2018. № 181. 48 с.
2. *Касьянов В.Н., Касьянова Е.В.* Методы и система облачного параллельного программирования // Проблемы оптимизации сложных систем: Материалы XIV Международной Азиатской школы-семинара. Алматы. 2018. Часть 1. С. 298–307.
3. *Kasyanov V.N.* Methods and tools for structural information visualization // WSEAS Transactions on Systems. 2013. V. 12. № 7. P. 349–359.

4. *Касьянов В.Н.* Применение графов в программировании // Программирование. 2001. № 3. С. 51–70.
5. *Евстигнеев В.А., Касьянов В.Н.* Оптимизирующие преобразования в распараллеливающих компиляторах // Программирование. 1996. № 6. С. 12–26.
6. *Касьянов В.Н., Евстигнеев В.А.* Графы в программировании: обработка, визуализация и применение. СПб.: БХВ-Петербург, 2003. 1104 с.
7. *Herman I., Melancon G., Marshall M.S.* Graph visualization and navigation in information visualization: a survey // IEEE Transactions on Visualization and Computer Graphics. 2000. V. 6. P. 24–43.
8. *Касьянов В.Н., Касьянова Е.В.* Визуализация информации на основе графовых моделей // Научная визуализация. 2014. Т. 6. № 1. С. 31–50.
9. *Di Battista G., Eades P., Tamassia R., Tollis I.G.* Graph Drawing: Algorithms for Vizualization of Graphs. Prentice Hall, 1999. 397 p.
10. *Kaufmann M., Wagner D. (editors)* Drawing Graphs. Methods and Models. Berlin: Springer, 2001. 312 p. (Lecture Notes in Computer Science, V. 2025)
11. *Sugiyama K.* Graph Drawing and Applications. For Software and Knowledge Engineers. World Scientific. 2002. 215 p.
12. *Demetrescu C., Finocchi I., Stasko J.T.* Specifying algorithm visualizations: interesting events or state mapping? // Lecture Notes in Computer Science (LNCS). 2002. V. 2269. P. 16–30.
13. *Касьянов В.Н., Золотухин Т.А.* Visual Graph – система для визуализации сложно структурированной информации большого объема на основе графовых моделей // Научная визуализация. 2015. Т. 7. № 4. С. 44–59.
14. *Brandes U., Eiglsperger M., Herman I., Himsolt M., Marshall M.* Graph ML progress report: structural layer proposal // Lecture Notes in Computer Science (LNCS). 2002. V. 2265. P. 501–512.
15. *Zaki A.* Comprehensive survey on dynamic graph models // International Journal of Advanced Computer Science and Applications. 2016. V. 7. № 2. P. 573–582.