Визуализация графовых представлений потоковых программ

В.Н. Касьянов[1,2], Т.А. Золотухин[1,2]
kvn@iis.nsk.su|tzolotuhin@gmail.com
[1] Институт систем информатики, Новосибирск, Россия;
[2] Новосибирский государственный университет, Новосибирск, Россия

В докладе рассматривается задача визуализации графовых представлений потоковых программ. Описывается эффективный алгоритм построения наглядных изображений иерархических графов с портами, а также его эффективная реализация в рамках системы Visual Graph.

Ключевые слова: *графовые представления потоковых программ, графы с портами, визуализация графов, иерархические графы, системы визуализации графов.*

Visualization of graph presentations of data-flow programs

V.N. Kasyanov[1,2], T.A. Zolotuhin[1,2]
kvn@iis.nsk.su|tzolotuhin@gmail.com
[1]Institute of Informatics Systems, Novosibirsk, Russia;
[2]Novosibirsk State University, Novosibirsk, Russia

In this paper, the problem of visualization of graph representations of data-flow programs is considered. An effective algorithm for constructing visual images of hierarchical graphs with ports is described, as well as its effective implementation within the framework of the Visual Graph system.

Keywords: *graph presentations of data-flow programs, graphs with ports, visualization of graph, hierarchical graphs, graph visualization systems.*

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 потоковый язык Cloud Sisal [8], 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 языков программирования с неявным параллелизмом гарантирует детерминированные результаты для параллельной и последовательной реализаций — то, что невозможно гарантировать для традиционных языков, подобных языку Фортран. Более того, неявный параллелизм языка снимает необходимость переписывания исходного кода при переносе его с одного компьютера на другой. Гарантировано, что программа с неявным параллелизмом, правильно исполняющаяся на персональном компьютере, будет также давать правильные результаты на высокоскоростном параллельном или распределенном вычислителе.

В докладе рассматривается 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. [9]

Цель создания системы – дать возможность широкому кругу лиц, находящихся в удаленных населенных пунктах или в местах с недостаточными вычислительными средствами, но имеющих выход в Интернет, дистанционно и без установки дополнительного программного обеспечения на своих недорогих вычислительных устройствах в визуальном стиле создавать и отлаживать переносимые параллельные программы на языке Cloud Sisal, а также затем дистанционно (в облаке) осуществлять эффективное решение своих задач, исполняя на некоторых супервычислителях, доступным им по сети, созданные и отлаженные переносимые Cloud-Sisal-программы, предварительно адаптировав их под используемые супервычислители с помощью облачного оптимизирующего кросс-компилятора, предоставляемого системой.

На рынке широко представлены наукоемкие программные продукты, использующие методы визуализации информации на основе графов. 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 Higres, aiSee, yEd и Cytoscape [6].

2. Графовое представление Cloud Sisal программ

Система CPPS использует графовое представление Cloud Sisal программ, которое ориентировано на их семантическую и визуальную обработку и основано на атрибутированных иерархических графах [4]. Это представление, получившее название IR (Intermediate

The publication of the conference proceedings was supported by the RFBR, grant No. 18-07-20045\18

Representation), в отличие управляющего графа [5], обычно используемого в оптимизирующих компиляторах для императивных языков (таких как C или Фортран), выражает не поток управления, а поток данных в программе.

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 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) любого типа, 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.

3. Изображения IR графов

Предполагается, что IR представления Cloud Sisal программ демонстрируются пользователям системы наряду с их текстовыми представлениями и используются пользователями для целей визуальной отладки Cloud Sisal программ и управляемой их оптимизации. Возникает необходимость автоматического построения наглядного изображения IR графов.

Предполагается следующие правила изображения IR графов:

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

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

3. Составные вершины изображаются в виде прямоугольников с прямоугольным выступом сверху слева для указания типа составной вершины, а также с кругами сверху справа и снизу для изображения входов и выходов этой вершины в их упорядоченности слева направо. Каждый круг, изображающий некоторый полюс фрагмента, состоит полукругового выступа наружу и полукругового 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 по своим внутренним точкам.

Основные трудности решения построения наглядного изображения IR графов связаны с тем, что в отличие от стандартной задачи укладки графа на плоскости [2-7] в IR графе вершины графа соединяются дугами через свои порты и могут иметь разный размер, который определяется характеристиками изображения тех графов, которые вложены в данную вершину.

4. Общая схема алгоритма

При описании алгоритма предполагается, что исходный граф является дэгом, т. е. ациклическим ориентированным графом. Если исходный граф является неориентированным и/или содержит циклы, то он сначала может быть приведен к дэгу путем задания и/или смены ориентации у части его ребер, а затем по построенной укладке дэга обратным преобразованием, примененным к изображениям дуг, получаем укладку исходного графа [3, 5].

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.

Само построение изображения одного текущего фрагмента базируется на методике так называемого поуровневого подхода к построению укладки ациклического ориентированного графа, которая была предложена К. Сугиямой (K. Sugiyama), и состоит из следующих трех основных этапов [3, 5]:

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 шага, реализация каждого 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.

Работа опубликована при финансовой поддержке РФФИ, грант 18-07-20045\18

5. Распределение вершин по уровням

Задачей данного шага 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 $i - j$. 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. $v_y = i$ для всех вершин с уровня L_i.

Существование поуровневого представления для любого ациклического графа очевидно. Однако не для всякого ациклического графа существует его сжатое представление — это может быть продемонстрировано уже для графа, состоящего всего из трех вершин (см. Рис. 1). 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 шаге алгоритма. Сжатое разбиение сводит эту глобальную задачу к задаче минимизации пересечений для двух соседних уровней.

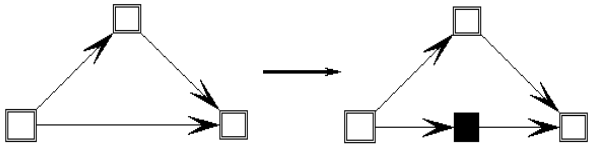

Рис. 1. Пример укладки графа, в которой вершины одного уровня расположены на одной вертикальной линии. Заштрихованная вершина является фиктивной.

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. Для каждой такой дуги (u, v), $u \in L_i$ и $v \in L_j$, с зазором $k = i - j > 1$ добавляется $k - 1$ вершин $u_1, \ldots u_{k-1}$, таких, что $u_m \in L_{i-m}$, и дуг (u_l, u_{l+1}) для всех $l < k - 1$. 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 другими опорными точками (точками сгиба) — вставленные на первом этапе фиктивные вершины. Таким образом, исчезает возможность пересечения дуг с вершинами.

Выполнение данного этапа связано с обработкой графа $G = (V, E)$, представляющего некоторый фрагмент F, в котором V состоит из полюсов 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 полюсов фрагмента строятся L_1 и L_{t+1} и эти полюса 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 проводится приведение построенного представления графа G к сжатому виду, используя технику фиктивных вершин.

6. 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 графе 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.

7. Определение координат вершин на уровне

После определения порядка вершин на уровне необходимо определить их реальные координаты. Дуги

The publication of the conference proceedings was supported by the RFBR, grant No. 18-07-20045\18

графа изображаются в виде ломанных с точками излома, 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, … , 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-координату 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 поставленных 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.

8. Реализация алгоритма

При описании алгоритма предполагалось, что исходный граф является ациклическим ориентированным графом. Однако его реализация в рамках системы Visual Graph [6] 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 [1], который позволяет специфицировать произвольного вида атрибутированные иерархические графы с полюсами.

9. Заключение

Рассмотренный в докладе алгоритм укладки графов с портами имеет квадратичную временную сложность и позволяет строить наглядные изображения графовых представлений потоковых 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).

10. Благодарности

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

11. Литература

[1] Di Battista G., Eades P., Tamassia R., Tollis I. G. Graph Drawing: Algorithms for Vizualization of Graphs. - Prentice Hall, 1999.

[2] Drawing Graphs. Methods and Models. - Berlin: Springer, 2001.

[3] Sugiyama K. Graph drawing and applications. For software and knowledge engineers. - World Scientific. 2002.

[4] Касьянов В.Н. Иерархические графы и графовые модели: вопросы визуальной обработки // Проблемы систем информатики и программирования. - Новосибирск, 1999. - С. 7-32.

[5] Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. - СПб.: БХВ-Петербург, 2003.

[6] Касьянов В.Н., Золотухин Т.А. Visual Graph — система для визуализации сложно структурированной информации большого объема на основе графовых моделей // Научная визуализация. – 2015. - Том. 7, N 4. - С. 44 – 59.

[7] Касьянов В. Н., Касьянова Е. В. Визуализация информации на основе графовых моделей // Научная визуализация – 2014. – Т. 6, N 1. – С. 31 - 50.

[8] Касьянов В. Н., Касьянова Е. В. Язык программирования Cloud Sisal. – Новосибирск, 2018. – 42 с. – (Препринт/ РАН, Сиб. отд-ние, ИСИ; N181).

[9] 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.

Работа опубликована при финансовой поддержке РФФИ, грант 18-07-20045\18