УДК 004.41/.42

М.Г. Адигеев, канд.техн.наук, Южный федеральный университет,
E-mail: madi@math.sfedu.ru

Экономное отображение вычислений на топологии типа «решетка»

Аннотация

Статья посвящена проблеме эффективного отображения исполняемой программы в структуру вычислительной системы с программируемой структурой, имеющей топологию двумерной решётки [6]. Описан алгоритм отображения вычислительных операций в узлы решетки, гарантирующий для потоковых (бесконтурных) графов вычислений получение конвейерной реализации с минимальным возможным интервалом инициализации итерации. Работа выполнена в Южном федеральном университете в рамках темы № 02.740.11.0208.

Введение

Одним из перспективных направлений развития выскопроизводительных вычислений является применение однородных вычислительных сред (ОВС) с программируемой структурой. Такая вычислительная система состоит из множества процессорных элементов (ПЭ), соединённых коммуникационными линиями в единую сеть [6,7,8,9,11]. Результат выполнения каждой операции передаётся из того ПЭ, в котором он был получен, непосредственно в те ПЭ, которые будут его использовать для дальнейших вычислений. Такой принцип работы («структурная реализация вычислений», [5]) позволяет значительно ускорить вычисления, в том числе за счёт сокращения времени на запись и чтение

результатов вычислений в оперативную память. Однако достижение максимальной скорости вычислений возможно только при оптимальном отображении графа вычислений на топологию вычислительной системы [6,7,11,9,5]. Поскольку коммуникационная система ОВС часто имеет топологию двумерной решётки, актуальной является задача разработки методов эффективного отображения графа вычислений на такую топологию. Подобные задачи рассматривались в [6] и [9,11]. В данной работе предложен новый метод, позволяющий *автоматически* строить как отображение бесконтурного графа вычислений на решётку процессорных элементов, так и расписание вычислений, гарантирующее работу вычислительного конвейера с минимальным возможным интервалом инициализации итераций. Предложенный метод реализован в Диалоговом высокоуровневом оптимизирующем распараллеливателе (ДВОР) [3]. Работа выполнена в Южном федеральном университете в рамках темы № 02.740.11.0208.

1. Постановка задачи

Рассматривается задача отображения графа вычислений на граф ОВС, имеющей топологию двумерной решётки. Предполагается, что коммуникационная подсистема ОВС обладает следующими характеристиками:

A1. Соединения между соседними узлами решётки поддерживают полнодуплексный режим, то есть между соседними узлами возможна одновременная передача данных в обе стороны.

A2. При отображении фрагментов графа вычислений возможно временное разделение пересылок данных, т.е. можно выполнять один шаг конвейера за несколько тактов работы (вычисления + пересылки данных) вычислительной системы. При этом возможна пересылка

различных данных по одному и тому же ребру в одном и том же направлении, но в разное время (на разных тактах).

А3. Каждый узел решётки содержит память, достаточную для хранения промежуточных данных.

Этим условиям удовлетворяют как уже имеющиеся архитектуры (см., например [10]), так и разрабатываемые (например, [6]).

На описанной ОВС рассматривается выполнение вычислений в конвейерном режиме, то есть прохождение потока обрабатываемых данных через ОВС и выполнение над каждым элементом потока одинакового набора операций. Этот набор операций описывается *графом вычислений*, который состоит из множества операций (вершины графа вычислений) и дуг, описывающих зависимости между операциями. Дуга соединяет операции O1 и O2, если результат операции O1 является входным аргументом для O2.

Для реализации заданного графа вычислений на ОВС, необходимо отобразить множество операций во множество узлов решётки (процессорные элементы, ПЭ), задать пути передачи данных между ПЭ и расписание работы каждого ПЭ (какие вычислительные операции и пересылки данных ПЭ выполняет на каждом такте работы).

Описываемый метод отображения может быть применён к графам вычислений, удовлетворяющим следующим условиям:

Г1. Граф не содержит контуров, то есть ориентированных циклов.

Г2. Степени захода не превышают 2.

Г3. На выходе у каждой операции — только одно значение. То есть если из вершины на графе вычислений выходят несколько дуг, по ним передаётся одно и то же значение.

Условие Г1 описывает графы потоковых вычислений, аналогичных вычислениям в систолических массивах, но реализованных путём отображения программы на универсальную топологию (решётка), а не аппаратно.

Остальные условия не накладывают существенных ограничений, поскольку им удовлетворяют графы большинства программ.

Заметим также, что предлагаемый метод может быть обобщён на случай графов, не удовлетворяющих условиям Г1–Г3, однако в этом случае не гарантируется минимальное значение интервала инициализации итераций (этот термин определён ниже).

Таким образом, на входе у процедуры автоматического отображения вычислений на архитектуру ОВС имеются следующие объекты:

1. Ориентированный граф G(V,E). Вершины графа помечены именами переменных и операций.

2. Размеры двумерной решётки: nMeshWidth, nMeshHight. Поскольку рассматриваются системы с полнодуплексной связью, каждое ребро решётки можно трактовать как пару противоположно ориентированных однонаправленных *дуг*.

Для того чтобы избежать путаницы, в рамках данной статьи вершины решётки ОВС будем называть *узлами*, вершины графа вычислений — *операциями*. Узлы решётки будем задавать двумя координатами (номер строки и номер колонки).

Выходные данные процедуры— ***структурная реализация*** (СР) вычислений на решётке, включающая в себя следующие компоненты:

- ***Укладка*** (embedding) графа вычислений в граф решётки. Задаётся в виде пары:

- отображение множества вершин графа вычислений (VG) во множество вершин решётки (VM), и
- отображение каждой дуги графа вычислений (элемент в EG) в путь на решётке (подмножество в EM). Такой путь будем называть ***маршрутом***.

- ***Расписание*** (schedule) выполнения операций вычисления и пересылки. Расписание также задаётся в виде двух отображений:
  - Для каждого узла решётки и каждого такта определяется вычислительная операция и номер итерации. Если отображение операций в узлы решётки инъективно (тогда и вся укладка называется инъективной), то достаточно указать номер итерации. Если данный узел на данном такте не выполняет никакой операции, то результатом отображения является специальное значение (void).
  - Для каждой дуги решётки и для каждого такта определяется пересылка (дуга на графе вычислений) и номер итерации.

Качество структурной реализации программы определяется двумя основными показателями:

- Скорость работы. Наиболее критичными с точки зрения общего времени работы параллельной программы является, безусловно, время работы циклических участков (циклы while и for). При конвейерном способе выполнения параллельной программы время работы цикла определяется *интервалом инициализации итерации* (ИИИ). В [11] аналогичный показатель назван *конвейерной временной сложностью*. Величина ИИИ определяется как свойствами графа вычислений для рассматриваемого фрагмента программы, так и особенностями структурной реализации этого фрагмента на конкретной вычислительной архитектуре. Очевидно, что ИИИ исходного графа

вычислений является минимальным возможным значением для любой структурной реализации. Конкретную структурную реализацию фрагмента программы будем называть *экономной*, если ИИИ для полученного конвейера равен интервалу инициализации итераций для конвейера, соответствующего исходному графу.

- Площадь фрагмента решётки, занятой укладкой. Будем для краткости называть этот показатель *площадью укладки*. Он определяет объём ресурсов системы, задействованных для выполнения вычислений.

В данной работе показано, что для любого фрагмента программы с бесконтурным графом вычислений возможна эффективная структурная реализация, и приведён алгоритм её построения. Дополнительно рассмотрены усовершенствования этого алгоритма, направленные на уменьшение площади укладки. В силу ограниченности объёма статьи, теоретические результаты приведены без строгого доказательства.

Очевидно, что экономная реализация (с минимальным возможным ИИИ) возможна только в случае, если при укладке операций в узлы решётки разные операции отображаются в разные узлы. Поэтому в дальнейшем будем рассматривать только такие укладки, и будем называть их *инъективными* укладками.

2. Построение структурной реализации

2.1. Прямоугольная укладка

Ядром полученных в данной статье результатов является описываемый в данном разделе алгоритм укладки ориентированных бесконтурных графов в двумерную решётку. Данный алгоритм по заданному графу строит для него укладку особого вида — *прямоугольную* укладку. В данном разделе мы рассмотрим построение прямоугольной укладки, а в следующем

разделе покажем, что для любого бесконтурного графа можно построить прямоугольную укладку, допускающую экономную структурную реализацию.

Итак, рассмотрим следующий алгоритм. Входом для него служит ориентированный бесконтурный граф G(V,E). Вершины графа помечены именами переменных и операций.

Для описания и обоснования алгоритма нам потребуются некоторые понятия из теории графов. Определение ярусного представления, его ширины и высоты взяты из [1], понятия «моноширинное ЯП» и «короткая дуга» введены в данной статье в качестве технических терминов, необходимых для формулировки алгоритма.

Определение. ***Ярусным представлением графа*** называется разбиение вершин графа на непересекающиеся подмножества (*ярусы*) V_i (i=1,…,L), при котором концевые вершины каждой дуги лежат в разных ярусах. Будем считать, что ярусы пронумерованы в «топологическом» порядке, т.е. для каждой дуги номер яруса начальной вершины меньше, чем номер яруса конечной вершины.

Определение. ***Шириной ярусного представления*** называется максимальная мощность яруса.

Определение. ***Высотой ярусного представления*** называется количество ярусов.

Определение. ***Моноширинным ярусным представлением*** будем называть ярусное представление, в котором мощности всех ярусов одинаковы.

Определение. ***Короткая дуга***: дуга, вершины которой лежат в соседних ярусах. Дуги, не удовлетворяющие такому условию, будем называть ***длинными***.

Алгоритм построения прямоугольной укладки состоит из следующих шагов.

1. Для заданного графа получить моноширинное ярусное представление (ЯП) без длинных дуг. Для этого необходимо сначала построить произвольное ярусное представление графа, а затем выровнять ширину ярусов, добавив новые вершины двух видов: «Т» (транзитная вершина, которая не выполняет операцию, а просто пересылает полученные данные дальше) и «П» (пустая операция, ничего не выполняет, добавляется для удобства дальнейшей работы). Детальный алгоритм приведен в приложении (Алгоритм `ПостроитьЯП`).
2. Пусть ЯП задаётся графом G*, имеет ширину nGraphWidth и высоту nGraphHight.

   Если размеры ЯП превышают размеры решётки, т.е. `min(nGraphWidth,nGraphHight) > min(nMeshWidth,nMeshHight)` или

   `max(nGraphWidth,nGraphHight) > max(nMeshWidth,nMeshHight),`

   то сообщить о невозможности уложить граф в решётку и прекратить работу.
3. Выделить в решётке прямоугольный фрагмент размера `nGraphWidth × nGraphHight`. Построить укладку моноширинного ярусного представления G* в выделенный фрагмент решётки: каждый ярус ЯП отображается в соответствующую колонку прямоугольника, т.е. вершины *i*-го яруса отображаются в узлы *i*-й колонки прямоугольника. Размещение вершин одного яруса графа G* в узлах одной колонки решётки на данном этапе считаем произвольным. Вопросы оптимизации размещения рассматриваются в Приложении 2.
4. Построить маршруты пересылки данных. Для этого необходимо для каждой дуги (*u*,*v*) графа G* указать путь на решётке, по которому результат операции *u* будет передаваться на вход операции *v*. Поскольку граф G* не содержит длинных дуг, операции *u* и *v* в

результате прямоугольной укладки оказались размещены в соседних колонках. Допустим для определённости, что операция *u* была размещена в узле (k,i), а операция v — в узле (m,i+1). Маршрут от *u* к *v* можно построить двумя способами:

a. Сначала по строке к узлу (k, i+1), а затем по столбцу от (k,i+1) к (m,i+1).

b. Сначала по столбцу к (m,i), потом по строке к (m,i+1).

Каждый маршрут будем строить по правилу (a). См. пример на Рисунок 1. Это правило важно для обоснования теоремы, рассматриваемой в следующем разделе.

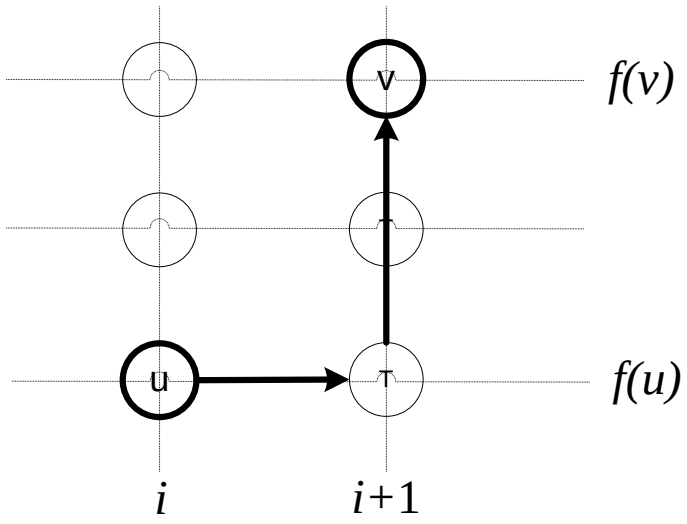


Рисунок 1

В результате описанных выше шагов получится укладка графа G* в решётку. Укладка исходного графа G может быть получена из укладки G*: вершины G образуют подмножество вершин G*, а маршруты, соответствующие длинным дугам графа G, очевидным образом получаются последовательным соединением соответствующих маршрутов для дуг графа G*.

Однако для полученной укладки может оказаться, что несколько маршрутов проходят по одному и тому же ребру решётки. Поэтому при составлении расписания пересылок данных для полученной укладки может возникнуть необходимость разделить пересылки по таким маршрутам во времени: сначала пересылать данные по одному маршруту, потом — по другому. Это приведёт к построению неэкономной реализации. В следующем разделе показано, что полученную укладку всегда можно преобразовать таким образом, что разные маршруты не проходят по одному и тому же ребру. Математически это свойство укладки можно охарактеризовать с помощью понятия «уплотнение».

Определение. Для заданной укладки графа G в решётку *уплотнением* (*congestion*) дуги e на решётке называется количество рёбер графа вычислений, образы которых на решётке проходят по дуге e. Максимальное уплотнение дуги называется уплотнением укладки.

Покажем, что в рамках описанной модели для любого графа вычислений можно построить укладку с уплотнением, равным 1, в решётку подходящего размера.

2.2. Укладка с единичным уплотнением

При построении укладки в соответствии с описанным выше алгоритмом, возможно «наложение» дуг, т.е. ситуация, когда два или более маршрута проходят по одному ребру решётки. Это в некоторых случаях может приводить к конфликту, который заключается в использовании одного канала для пересылки разных данных. Этот конфликт можно устранить при составлении расписания, разделив конфликтующие пересылки во времени. Но такое разделение приведёт к увеличению ***интервала***

инициализации итераций, что означает ухудшению качества распараллеливания.

Покажем, что если не учитывать размеры решётки, для бесконтурного графа всегда можно построить укладку с единичным уплотнением. Для этого, однако, может понадобиться увеличить размеры прямоугольной области, в которую укладывается граф.

Проведём анализ ситуаций, возникающих при укладке графа вычислений в решётку.

1. По одному ребру решётки проходят два маршрута, но в разных направлениях. См. Рисунок 2.

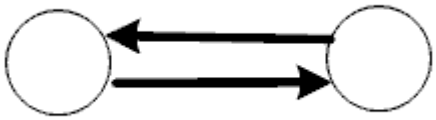


Рисунок 2

Поскольку в данной работе рассматривается случай полнодуплексных соединений между узлами решётки, эта ситуация не является конфликтной. Поэтому далее рассмотрим ситуации, когда маршруты проходят по ребру решётки в одном и том же направлении.

2. Два или более маршрута проходят по одному и тому же ребру с одного яруса на другой.

Данная ситуация не создаёт конфликта, поскольку при построении укладки ранее было принято правило: маршрут сначала переходит на нужный ярус, и уже на нужном ярусе проходит к нужной вершине. См. Рисунок 1. Действительно, в этом случае по одному ребру решётки между ярусами проходят только маршруты, выходящие из одного и того же узла. А в

соответствии с условием Г3 это означает, что по данным маршрутам передаётся одно и то же значение. Поэтому конфликт отсутствует: *одно и то же* значение передаётся на следующий ярус, и уже там передаётся по разным маршрутам к нужным вершинам.

3. Два или более маршрута проходят по одному и тому же ребру решётки между узлами одного яруса.

В этом случае для того, чтобы избежать конфликта, имеется два варианта:

1) Разместить операции очередного яруса в узлах решётки таким образом, чтобы маршруты проходили по одному и тому же ребру только в разных направлениях — в этом случае конфликт не возникает. Детальный анализ приведён в разделе «Оптимизация укладки с помощью выбора размещения вершин внутри очередного яруса» из Приложения 2.
2) Вынести некоторые из конфликтующих маршрутов на новый ярус.

Вариант (1) позволяет избежать увеличения площади укладки. Но вариант (1) возможен не всегда (см. анализ в Приложении 2). Вариант (2) возможен в любом случае, но приводит к увеличению площади укладки.

Пример вынесения части маршрутов на новый ярус приведён на рисунке 3. На рис. 3а изображены требуемые маршруты. Стрелками одного цвета изображены маршруты, поставляющие данные для одной и той же операции. Поскольку на участке между узлами 3 и 4 по одному ребру проходят три маршрута, возникает конфликт. Его можно устранить, перенеся один из маршрутов (например, красный), на новый ярус, и перенеся конечный пункт жёлтых маршрутов в узел 4. Результат показан на рис. 3б.

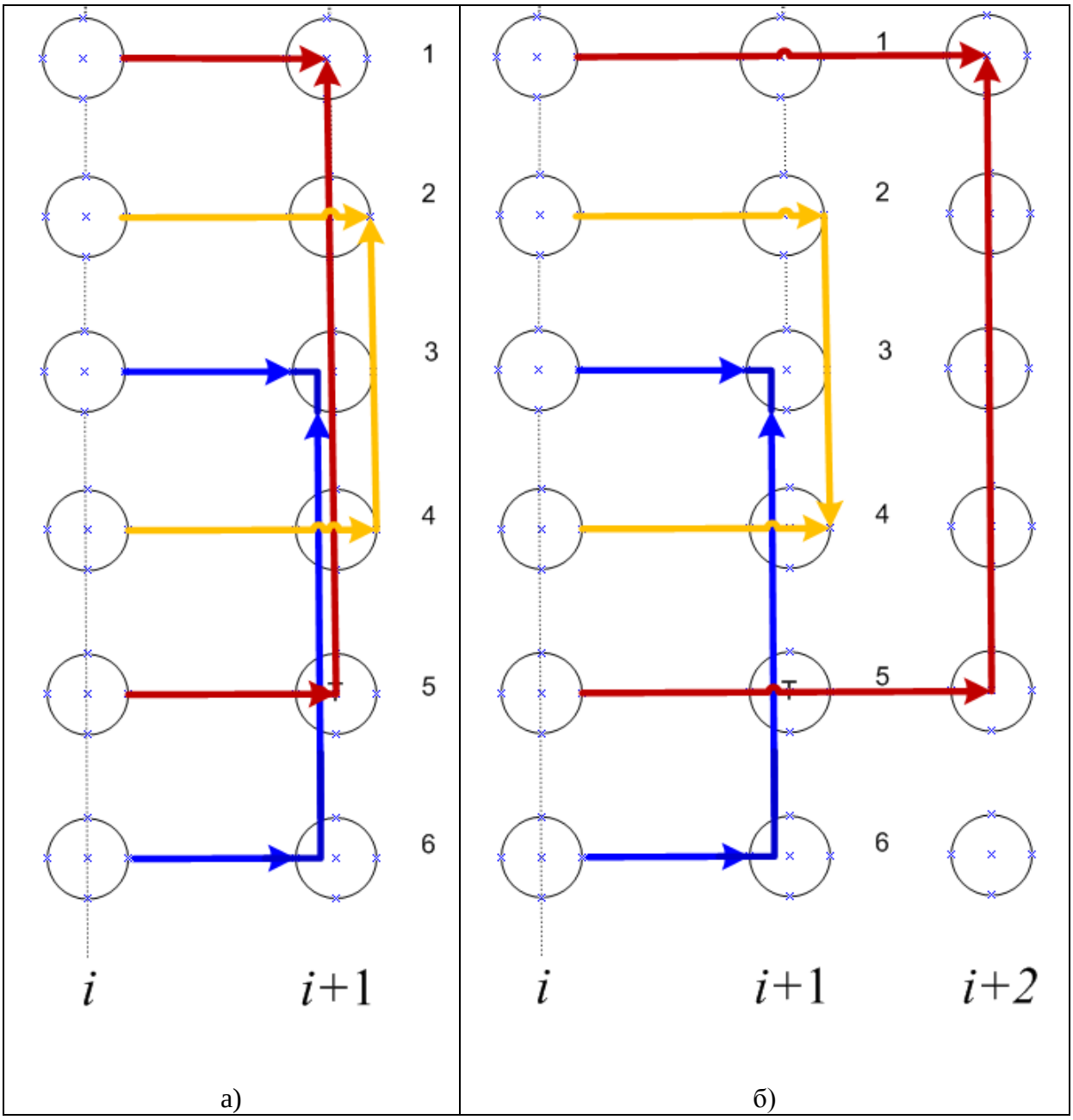


Рисунок 3

Результатом описанных построений и анализа является следующая теорема:

Теорема 1. Для любого бесконтурного графа вычислений существует инъективная укладка с единичным уплотнением в решётку подходящих размеров.

2.3. Построение расписания

Для реализации графа вычислений на решётке, необходимо кроме укладки графа (отображения операций и пересылок в узлы и дуги решётки) построить расписание выполнения вычислительных операций и пересылок.

Теорема 2. Если для графа вычислений имеется инъективная укладка с единичным уплотнением, то для этого графа существует экономная структурная реализация.

Доказательство.

Утверждение теоремы 2 следует из того, что для любого бесконтурного графа вычислений можно построить расписание с ИИИ, равным максимальной величине задержек операций (см. [13]). А при инъективной укладке с единичным задержки операций для графа на решётке такие же, как и для исходного графа, поскольку отсутствуют конфликты при выполнении операций и при пересылках.□

Заключение

В данной статье рассмотрена задача отображения фрагмента программы на вычислительную систему с топологией двумерной решетки. Приведен алгоритм построения укладки, допускающей построение экономной (без увеличения интервала инициализации итераций) реализации вычислений. Этот алгоритм лёг в основу одной из процедур отображения компилируемой программы на архитектуру вычислительной системы в составе Диалогового высокоуровневого оптимизирующего распараллеливателя (ДВОР) [3].

Полученные результаты могут быть развиты в дальнейших исследованиях. Например, полезно было бы получить оценку площади, занимаемой на

решётке укладкой, которую строит описанный алгоритм. Отдельного исследования заслуживают вопросы оптимизации укладки по площади и глубине, а также оптимальные алгоритмы переразмещения данных после выполнения фрагмента программы для выполнения следующего фрагмента.

Приложение 1. Алгоритм построения моноширинного ярусного представления без длинных дуг

Алгоритм `ПостроитьЯП`

1. Построить ярусное представление графа. Например, с помощью итерационного процесса:
    1) Найти все источники, т.е. вершины со степенью захода 0.
    2) Поместить их в новый ярус.
    3) Удалить все источники из графа.
    4) Есть граф не пуст, то перейти к п.1
2. Каждую длинную дугу представить в виде цепи из новых вершин, имеющих пометку «Т». Допустим, дуга ведёт из яруса с номером i к ярусу с номером j, $j>i+1$. Преобразуем эту дугу в путь длины $(j–i)$, добавив $(j–i–1)$ новую вершину. Все вершины пути размещаются в ярусах последовательно. В результате одна длинная дуга преобразуется в несколько коротких.
3. Вычислить ширину полученного ярусного представления. Дополнить каждый ярус до данной величины (т.е. чтобы мощность всех ярусов была одинакова), добавив необходимое количество новых вершин с пометками «П».

Пример применения алгоритма показан на Рисунок 4.

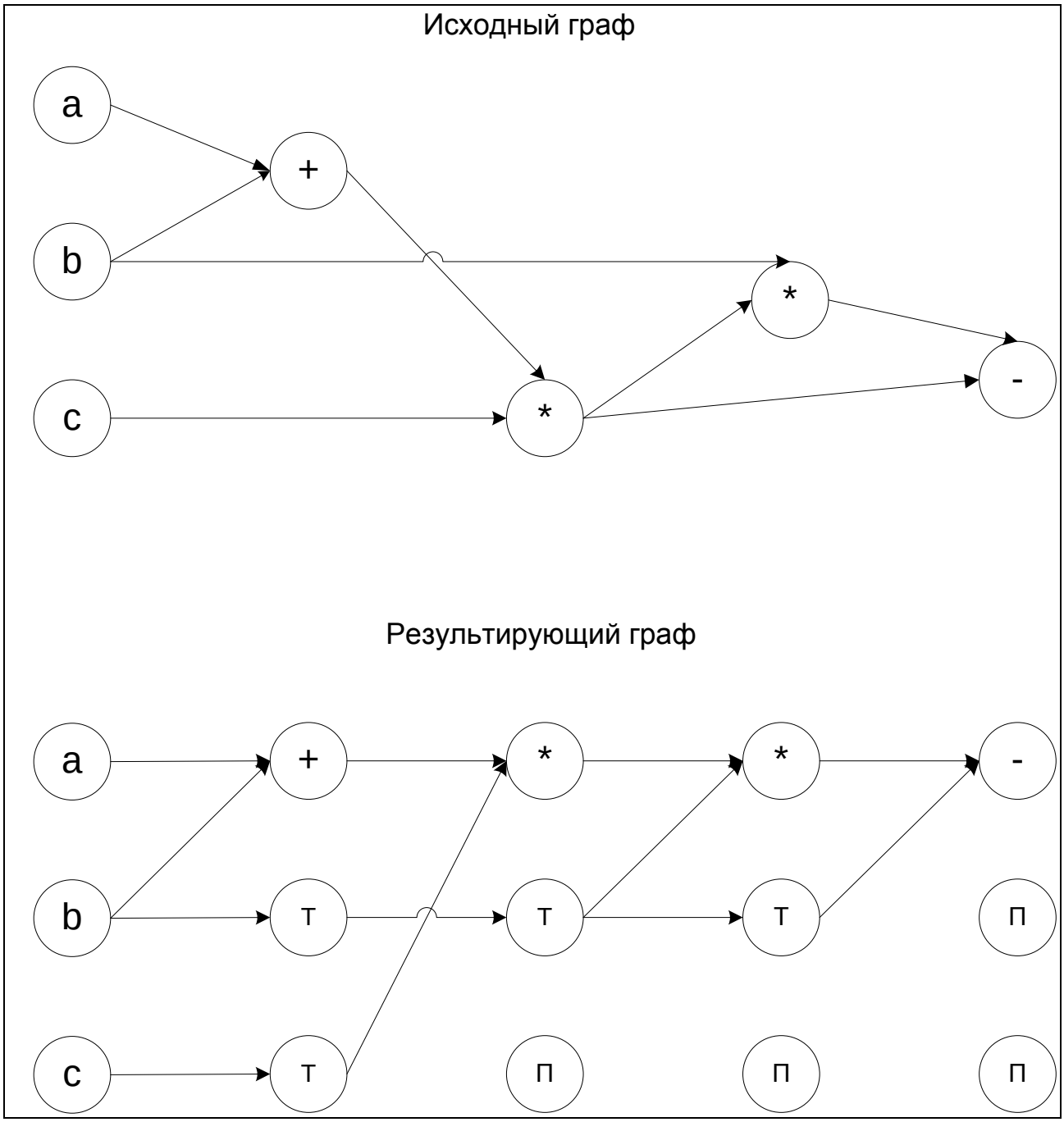
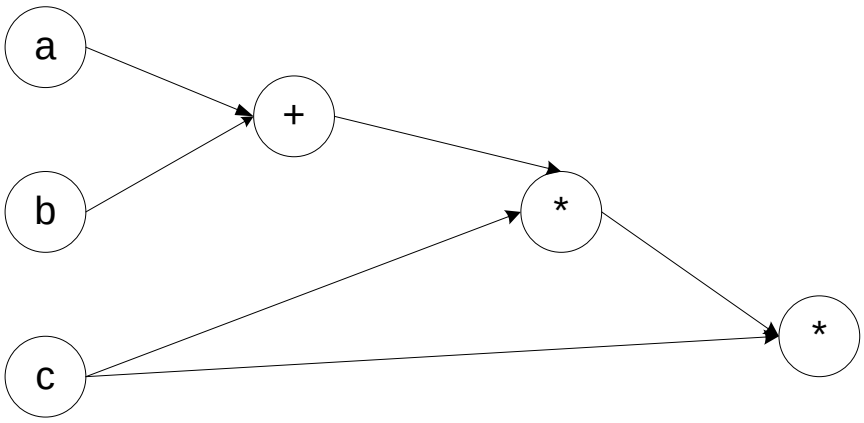


Рисунок 4

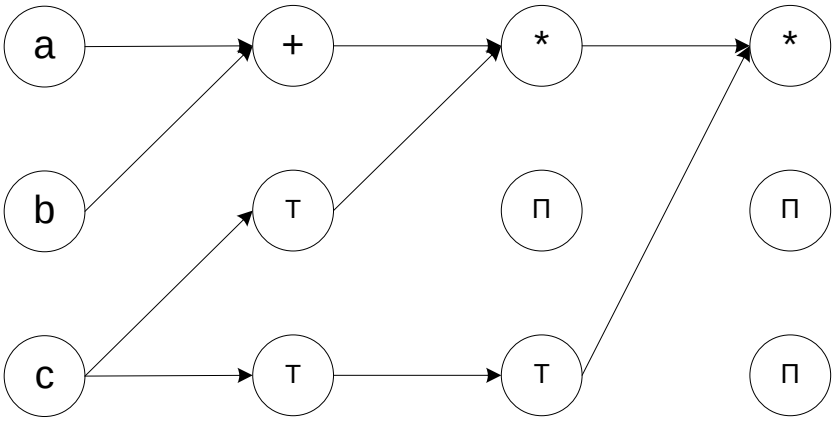
В силу принятого ранее (см. постановку задачи в документе «Укладка графа вычислений в двумерную решётку») упрощающего предположения Г3 можно усовершенствовать шаг 2 данного алгоритма следующим образом: если из вершины выходят несколько дуг, одна (или несколько) из которых длинные, то можно совместить начальные фрагменты путей, в

которые преобразуются длинные дуги, как показано на Рисунок 5. Это позволит избежать ненужного увеличения ширины ЯП.

Исходный граф



Результирующий граф при использовании неусовершенствованного алгоритма



Результирующий граф при использовании усовершенствованного алгоритма

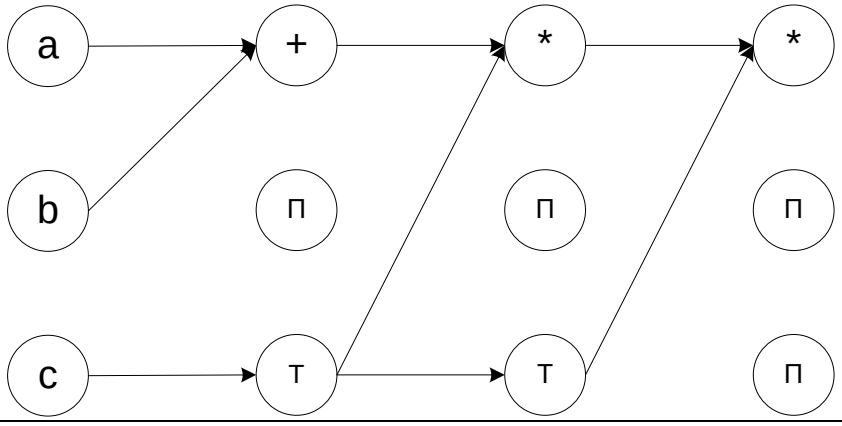


Рисунок 5

Приложение 2. Оптимизация прямоугольной укладки

Оптимизация укладки с помощью выбора размещения вершин внутри очередного яруса

Описанные в основном тексте статьи алгоритмы позволяют построить укладку, которая гарантированно является инъективной и имеет удлинение, равное 1. Но важным показателем качества укладки является также количество ярусов (глубина). В данном параграфе рассмотрим алгоритм оптимизации укладок по глубине.

На входе у алгоритма – построенная описанным выше способом укладка.

Для каждой вершины u из i-го (ранее размещённого) яруса через $f(u)$ и $g(u)$ обозначим номера вершин $(i+1)$-го (текущего) яруса, к которым необходимо передать данные из узла u. При этом считаем, что $f(u) \le g(u)$. Если от узла u необходимо передать данные только к одному узлу следующего яруса, то $f(u) = g(u)$. Если от узла u необходимо передать данные к более чем двум узлам следующего яруса, то через $f(u)$ обозначим наименьший номер такого узла, а через $g(u)$ — наибольший.

Тогда передача данных из i-го яруса в $(i+1)$-й задаётся множеством *сегментов* $[f(u),g(u)]$.

Рассмотрим следующую ситуацию: операции ярусов с номерами от 1 по i уже размещены в узлах решётки, и перед нами стоит задача разместить операции очередного, $(i+1)$-го яруса. Постараемся выполнить это размещение таким образом, чтобы избежать образования сдвоенных дуг.

Очевидно, что при этом необходимо учесть два фактора:

- Размещение операции предыдущих ярусов. Поскольку рассматривается ярусное представление без длинных дуг, достаточно учесть только размещение операций предыдущего, i-го яруса.
- Пересылки данных с i-го яруса на $(i+1)$-й.

Поскольку операции i-го уже размещены, каждую из них можно задать номером узла в ярусе (колонке) решётки.

После построения ярусного представления (п.1 алгоритма из документа «Алгоритм прямоугольной укладки») можно выполнить преобразования этого ЯП, которые оптимизируют его по ряду показателей (удлинение дуг, уплотнение дуг).

Вершины одного яруса размещаются в узлах колонки таким образом, чтобы минимизировать максимальное время пересылки данных. Требуемые пересылки определяются индексами узлов, в которых размещены начальная и конечная вершина каждой дуги (при этом начальная вершина размещается в узле предыдущей колонки, а конечная — в узле текущей колонки). Например, рассмотрим дугу, ведущую из вершины u яруса с номером i в вершину v яруса с номером (i+1). И допустим, что отображение вершин в узлы задаётся функцией f, то есть вершина u размещается в узле с номером $f(u)$ колонки i, а вершина v — в узле с номером $f(v)$ колонки i+1. В этом случае время пересылки данных для дуги (u,v) характеризуется величиной $|f(u)–f(v)|$.

В качестве критерия оптимальности укладки (помимо ИИИ) можно выбрать максимальное время пересылки данных по всем дугам, то есть величину $\max\{|f(u)–f(v)|: (u,v)\in E\}$ («стоимость» размещения). Для минимизации этой величины может быть применён один из следующих подходов:

1. Поиск глобального оптимума: алгоритм перебирает все возможные размещения вершин графа в узлах прямоугольника, для каждого размещения вычислять его «стоимость», и отбирать размещения с минимальной стоимостью. Для ЯП ширины W с L ярусами количество возможных размещений равно $(W!)^L$.

2. Поиск локального оптимума на основе следующих принципов:
    - последовательно размещать ярусы,
    - для каждого яруса находить такое размещение вершин этого яруса в узлах соответствующей колонки, при котором минимизируется $\max|f(u)-f(v)|$ по всем дугам, ведущим с предыдущего яруса в текущий,
    - при это считать, что размещение вершин предыдущих ярусов фиксировано.

    То для каждого из L ярусов требуется перебрать W! вариантов, всего будет L·W! комбинаций.

    В этом случае при размещении каждого яруса придётся решать «минисумный» вариант классической NP-трудной задачи «Одномерное размещение блоков» [3, стр.306].
3. Аналогично предыдущему варианту, но для каждого яруса искать не оптимальное размещение, в приближённое к оптимальному, пользуясь эвристикой. Аналогичный подход описан в [1].

В основу программной реализации алгоритма в ДВОР был положен вариант (2), поскольку для большинства графов, встречающихся на практике, ширина яруса (W) является небольшой величиной (до 10).

Оптимизирующие преобразования укладки

Ниже описаны преобразования укладки графа вычисления в решетку, которые позволяют получить более оптимальную (с меньшей глубиной) укладу. Данные преобразования являются эвристическими, то есть не гарантируют улучшения начальной укладки, поэтому при реализации должна выполняться проверка целесообразности применения того или иного преобразования.

Совмещение вершин

Суть преобразования: если две смежные вершины графа вычислений соответствуют операциям, которые могут быть выполнены одновременно, то такие две вершины следует «склеить» в одну. При совмещении вершин все дуги, которые входили в старые вершины (исходили из старых вершин), будут входить в новую (исходить из новой). Две операции можно выполнять одновременно в том случае, если они относятся к разным типам. Типы операций:

- Преобразование данных (вычислительная операция).
- Чтение данных (вершина, помеченная именем переменной, которая в рассматриваемом фрагменте программы только читается) или запись данных (вершина, в которой значение переменной изменяется, но в пределах укладываемого фрагмента не используется).
- Транзит, т.е. передача значения.

Подтягивание вершин

Преобразование заключается в укорачивании длинных дуг.

Пусть ширина построенного ярусного представления равна nWidth.

Тогда следует перебрать все вершины графа, и для каждой вершины v, из которой выходят только длинные дуги, выполнить операцию «подтягивание вершины»:

1. Пусть вершина v находится на ярусе с номером i и из неё исходят только длинные дуги. Если мощность $(i+1)$-го яруса меньше, чем nWidth, то переместить вершину v в ярус $(i+1)$.
2. Если все дуги, исходящие из v, всё ещё являются длинными, то продолжить (вернуться к п.1). Иначе операция завершена.

В результате выполнения этой операции некоторые короткие дуги могут стать длинными. Поэтому «подтягивание вершин» следует выполнять до

тех пор, пока есть хотя бы одна вершина, для которой эта операция может быть выполнена.

Совмещение ярусов

В том случае если мощность яруса с номером *i* меньше максимальной (nWidth), возможно совмещение этого яруса с соседним — (*i*–1)-м или (*i*+1)-м. То есть все вершины яруса (или часть вершин, в зависимости от количества «свободных мест» в соседнем ярусе) перемещаются в соседний ярус. В результате полученное ЯП может перестать быть «ярусным» в смысле строгого определения — если появятся дуги, соединяющие вершины из одного и того же яруса. Но полученное «квазиярусное» представление всё равно можно будет уложить в решётку рассмотренным ранее алгоритмом.

Литература

1. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. - Вильямс, Москва - Санкт-Петербург – Киев, 2000.
2. В.В.Воеводин, Вл.В.Воеводин. Параллельные вычисления. -СПб.: БХВ-Петербург, 2002.- 608 с.
3. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. — М.: Мир, 1982.
4. Диалоговый высокоуровневый оптимизирующий распараллеливатель программ: http://ops.rsu.ru/about_DVOR.shtml
5. Каляев А.В. Программирование виртуальных архитектур и организация структурно-процедурных вычислений в многопроцессорных системах с массовым параллелизмом. // Труды Первой Всероссийской научной конференции «Методы и средства обработки информации» (МСО’2003), 1-3 октября 2003 г., Москва — с. 37–46.

6. Корнеев В.В. Архитектура вычислительных систем с программируемой структурой. — Новосибирск: Наука, 1985 г., 166 с.
7. Корнеев В.В. Программная настраиваемость аппаратной структуры. // Открытые системы, 2007. – № 10, с. 12–16.
8. Корнеев В.В. Следующее поколение суперкомпьютеров. // Открытые системы, 2008. – № 8, с. 14–19.
9. Лукин Н.А. Основы теории проектирования архитектур функционально-ориентированных процессоров для систем реального времени // Высокопроизводительные вычислительные системы // Материалы Пятой Международной научной молодежной школы и Пятой Международной молодежной научно-технической конференции, 31 августа - 6 сентября 2008, Таганрог - Таганрог: Изд-во ТТИ ЮФУ, 2008. - с. 115 - 166.
10. Процессоры серии Tile64: http://www.tilera.com/products/processors/TILE64
11. Тришин В.Н., Лукин Н.А. Реализация дискретного преобразования Фурье на однородной вычислительной среде. // Труды Пятой Международной конференции «Параллельные вычисления и задачи управления» (PACO’2010), 26 – 28 октября 2010, Москва. — с. 1015 – 1023.
12. Штейнберг Б.Я. Математические методы распараллеливания рекуррентных циклов для суперкомпьютеров с параллельной памятью.// Ростов-на-Дону, Издательство Ростовского университета, 2004 г., 192 с.
13. Штейнберг Р.Б. Вычисление задержки в стартах конвейеров для суперкомпьютеров со структурно процедурной организацией вычислений // Искусственный интеллект. Научно-теоретический журнал. Институт проблем искусственного интеллекта НАНУ. Украина, Донецк, ДонДИШИ, “Наука и Освита”, № 4, 2003, с. 105-112.