ЛЕКЦИЯ 7

Развертки и граф-машина

Содержание: *строгие и обобщенные развертки*, *развертки и параллелизм в алгоритмах*, *компьютерная интерпретация, граф-машина*, *теорема о гомоморфной свертке графа*, *параллельная структура*, *макро- и микропараллелизм*, *расщепляющие развертки*, *полумодуль обобщенных разверток*, *направленные графы*, *линейные развертки*, *расщепление алгоритма на фрагменты*, *рекуррентные соотношения*, *регулярные графы*.

Как уже отмечалось ранее, при заданном алгоритме и входных данных граф алгоритма определяется однозначно и представляет информационное ядро алгоритма. Такая его интерпретация связана с тем, что этот граф явно показывает, какая операция алгоритма с какой связана *информационно*. Всестороннее изучение информационных отношений в процессах реализации алгоритмов или, другими словами, информационной структуры алгоритмов является исключительно важной задачей. В частности, одной из важнейших информационных задач является нахождение всех возможных реализаций алгоритма на вычислительных системах параллельной архитектуры. Ранее было показано, что она эквивалентна описанию всех параллельных форм графа алгоритма. Напомним, что каждая параллельная форма позволяет разбить операции алгоритма на группы. При этом группы операций можно выполнять одна за другой последовательно, а все операции внутри каждой группы – параллельно.

Пока нет никаких оснований, мешающих рассматривать граф алгоритма как произвольный ориентированный ациклический граф. Без ограничения общности можно считать, что он размещен в некотором арифметическом пространстве X подходящей размерности n. Рассмотрим вещественный функционал $f(x)$, определенный на точках-вершинах x графа алгоритма G. Предположим, что дуга идет из точки u в точку v. Будем говорить, что функционал $f(x)$ возрастает (не убывает) вдоль этой дуги, если $f(v) > f(u)$ ($f(v) \geq f(u)$). Назовем функционал $f(x)$ *строгой* (*обобщенной*) *разверткой графа* G, если он строго возрастает (не убывает) вдоль всех дуг графа.

Степень важности разверток для исследования структуры алгоритмов через их графы определяется свойствами разверток. Пусть известна какая-нибудь *строгая* развертка $f(x)$. Каждая вершина графа находится на одной и только на одной поверхности уровня $f(x)=c$ развертки $f(x)$. Разобьем все вершины графа на группы по принадлежности поверхностям уровней и перенумеруем группы в порядке роста константы c. Ясно, что группы операций можно выполнять *последовательно* в том же порядке. На любой поверхности уровня никакие точки-вершины не могут быть связаны ни дугами графа алгоритма, ни его

путями. Это означает, что соответствующие таким вершинам операции можно выполнять *параллельно*. Другими словами, знание любой строгой развертки позволяет через ее поверхности уровней построить параллельную форму графа или, что то же самое, параллельную форму алгоритма. Верно и обратное: любой параллельной форме можно сопоставить вполне определенную строгую развертку. Для ее построения необходимо положить значение развертки в каждой вершине x равным номеру того яруса параллельной формы, в котором располагается эта вершина x.

Таким образом, между строгими развертками и параллельными формами алгоритма установлено взаимное соответствие. Граф алгоритма и развертки являются математическими объектами. Следовательно, на основе их использования можно создать математический аппарат для изучения параллелизма в алгоритмах. Эффективность изучения во многом будет зависеть от того, насколько в подходящем для исследований виде удастся представить граф алгоритма и в каком классе функционалов придется искать развертки. Вполне возможно, что в желаемом классе не окажется ни одной строгой развертки. И тогда окажутся важными обобщенные развертки, по крайней мере, как естественное *замыкание* множества строгих разверток.

Прежде чем переходить к математическим исследованиям, полезно рассмотреть компьютерную интерпретацию графа алгоритма и его разверток. Она поможет в дальнейшем лучшему пониманию получаемых результатов. Перенумеруем каким-либо образом все вершины графа. Развертки определены на конечном числе точек. Поэтому строгую или обобщенную развертку можно также задать вектором, в котором размерность равна числу вершин графа алгоритма, номер координаты совпадает с номером вершины, а значение каждой координаты есть значение развертки в соответствующей точке. Рассмотрим какую-нибудь реализацию какой-нибудь схемы алгоритма на каком-нибудь реальном параллельном или последовательном компьютере. Каковы бы не были архитектура компьютера, времена выполнения операций и времена передачи данных, реализация алгоритма *однозначно* определяет временны́е моменты окончания всех его операций. Сохранив соответствие между номерами координат и номерами вершин графа алгоритма, составим из этих моментов вектор. Очевидно, что он представляет строгую развертку. Следовательно, множество строгих разверток графа алгоритма содержит в качестве своего подмножества все реальные реализации самого алгоритма.

Поместим в каждую вершину графа алгоритма функциональное устройство, имеющее возможность выполнять соответствующую операцию. Пусть дуги графа представляют линии связи, обеспечивающие передачу информации от одного устройства к другому. Будем теперь рассматривать эту конструкцию как абстрактную специализированную вычислительную систему. Предположим, что после запуска системы все ее функциональные устройства начинают работать под собственным управлением, соблюдая следующие простые правила. Именно, входные данные по мере необходимости доступны

потребляющим их устройствам без каких-либо задержек; кроме неотрицательности не накладываются никакие ограничения на времена выполнения операций и времена передачи данных по линиям связи; каждое функциональное устройство может начинать выполнение операции в любой момент после того, как будут готовы для использования все ее аргументы. Назовем построенную систему *граф-машиной* и будем считать, что режимы ее функционирования описываются множеством разверток графа алгоритма.

Выше отмечалось, что среди этих режимов заведомо присутствуют такие, которые отражают любые *реальные* реализации алгоритма. Но очевидно, что имеются и другие режимы функционирования граф-машины, которые следует отнести к каким-то *гипотетическим* реализациям на гипотетических компьютерах. Возможно, наличие именно этих режимов позволит находить более эффективные схемы реализации конкретных алгоритмов и, следовательно, разрабатывать для них вычислительные системы более подходящей архитектуры.

Конечно, не стоит рассматривать граф-машину как прямой прообраз некоторой реальной вычислительной системы. В этом отношении она имеет немало недостатков. В ней очень много функциональных устройств и линий связи, каждое устройство и каждая линия связи срабатывают только по одному разу, совсем не используется память и т.д. Более того, несмотря на большое число устройств, граф-машина имеет возможность реализовывать только один алгоритм. Однако граф-машина и не предназначена для того, чтобы быть непосредственным прообразом реальной универсальной системы. Имеются две основные области ее использования. Во-первых, граф-машина является хорошим инструментом для изучения любых существующих и даже еще не существующих реализаций *конкретного* алгоритма. И, во-вторых, с помощью некоторых специальных преобразований именно из граф-машины можно построить математические модели многих типов вычислительных систем. Среди них имеются и такие, которые реализуют алгоритм за минимально возможное время, но обладают лучшими "техническими" характеристиками.

Несколько слов об этих преобразованиях. В их основе лежит гомоморфная свертка граф-машины в граф некоторой вычислительной системы со многими функциональными устройствами. Рассмотрим произвольный ориентированный граф G с множеством вершин V и множеством дуг E. Сейчас граф может не быть ациклическим и может содержать петли. Выберем в V любые две вершины u, v и сольем их в одну вершину z. Новое множество вершин обозначим V'. Перенесем на V' без изменения те дуги из G, для которых концевые вершины не совпадают ни с u, ни с v. Если же какая-то из концевых вершин совпадает с u или v, то такие дуги перенесем с заменой этих вершин на z. И, наконец, в новом графе все петли, относящиеся к одной вершине, заменим одной петлей. Также заменим одной дугой все кратные дуги одной ориентации. Множество дуг на V' обозначим E'. Граф с множеством вершин V' и множеством дуг E' обозначим G'. Преобразование графа G в граф

G' называется простым гомоморфизмом, а многократное преобразование простого гомоморфизма называется *гомоморфной сверткой графа*. При гомоморфной свертке графа G множество его вершин распадается на непересекающиеся подмножества. Каждое из подмножеств состоит из тех и только тех вершин, которые в конечном счете сливаются в одну вершину. Ясно, что любой ориентированный граф всегда можно гомоморфно свернуть в граф, состоящий из одной вершины и одной петли. Пример операций простого гомоморфизма приведен на рис. 7.1. Сливаемые вершины обозначены на нем "звездочками".

Рис. 7.1. Операции простого гомоморфизма.

Простым и конструктивным приемом осуществления гомоморфной свертки является операция проектирования. Если граф расположен в пространстве X, то спроектируем его вдоль любой прямой на перпендикулярную гиперплоскость. Пусть при этом какие-то вершины спроектируются в одну точку. Если в ту же точку спроектируются некоторые вектор-дуги, то поставим около точки петлю. Очевидно, что подобная операция есть гомоморфная свертка графа G. Чем больше точек-вершин графа расположено по направлению проектирования, тем больше вершин спроектируются в одну точку. Ничто не мешает повторять операцию проектирования многократно, пока не получится граф нужного строения. После числа шагов, равного размерности пространства, всегда в проекции получится одна точка. Если в графе G была хотя бы одна дуга, то около точки будет петля.

Гомоморфная свертка имеет очень прозрачный "компьютерный" смысл. Если граф G представляет граф-машину, то, выбирая вершины u, v, мы определяем две операции алгоритма и два ФУ, которые эти операции реализуют. Сливая вершины u, v, мы связываем с вершиной z не одну, а пару операций. ФУ, соотответствующее вершине z, должно иметь возможность выполнить обе операции последовательно. После многократного применения операции простого гомоморфизма полученный граф можно рассматривать как граф новой модели вычислительной системы. ФУ, связанное с любой его вершиной, обязано *последовательно* выполнять все операции алгоритма, связанные со всеми вершинами-прообразами. Дуги по-прежнему символизируют направленные передачи информации. Наличие петли около вершины говорит о том, что соответствующее ФУ будет срабатывать многократно.

Имеется одно принципиальное отличие граф-машины от вычислительной системы, полученной при гомоморфной свертке. Граф-машина не имеет память. Роль ее ячеек успешно выполняют сами ФУ в силу того, что каждое из них срабатывает только один раз. При многократном срабатывании ФУ для сохранения результатов предшествующих срабатываний уже нужна память. Зная граф алгоритма и временной режим срабатываний ФУ новой системы, можно подсчитать величину требуемой памяти и даже изучить процесс ее использования.

В общем случае на вычислительной системе, полученной после гомоморфной свертки, нельзя реализовать все временные режимы, допустимые для граф-машины. Тем не менее, имеет место важная

Теорема о гомоморфной свертке. Пусть при гомоморфной свертке граф-машины сливаются лишь вершины, связанные едиными путями. Тогда на вычислительной системе с полученным графом можно реализовать то же множество временны́х режимов, что и на граф-машине.

Достаточная разнесенность во времени моментов включения ФУ построенной системы гарантируется здесь тем, что сливаемые вершины находятся на одном пути. Поэтому соответствующие им операции как обязаны были раньше, так и имеют возможность теперь выполняться последовательно друг за другом. Подчеркнем также, что совсем не обязательно, чтобы образ сливаемых вершин имел в качестве своих прообразов все вершины, находящиеся на одном пути. Важно лишь, чтобы прообразы были связаны одним путем. Это обстоятельство имеет существенное значение, так как чаще всего объединяются вершины, соответствующие однотипным операциям, а они обычно в вычислениях перемешиваются с операциями других типов. Что же касается установления соответствия между вершинами графа алгоритма и срабатываниями ФУ, помещенными в вершины графа вычислительной системы, полученной после гомоморфной свертки, то теперь оно очень простое. Именно, если из двух вершин графа алгоритма одна достижима из другой, то из двух соответствующих срабатываний ФУ ей соответствует более позднее.

Таким образом, разбивая вершины графа алгоритма на подмножества, лежащие на одном пути, и объединяя их с помощью операций простого гомоморфизма, мы получаем конструктивный способ построения математических моделей вычислительных систем. Естественно, что таких систем может быть много, и о ни, вообще говоря, не одинаковы с точки зрения состава ФУ, их загруженности, размера присоединенной памяти, сложности коммуникационной сети и т. п. Но все эти системы по своим основным параметрам, кроме размера памяти, лучше, чем граф-машина. Они содержат меньшее число ФУ, загруженность каждого ФУ больше, число линий связи́ между ФУ меньше и при этом часто реализуется весь спектр временны́х режимов, включая наискорейшие. Снова можно ставить задачу оптимизации, пытаясь разбить вершины графа алгоритма на наименьшее число

подмножеств, лежащих на одном пути. И снова возникает противоречивая ситуация: уменьшение числа ФУ может привести к усложнению коммуникационной сети и увеличению объема памяти. Описанная свертка граф-машины была с успехом использована при построении математических моделей систолических массивов [1].

Но вернемся к исследованию параллелизма с помощью разверток. Изучение параллельной структуры алгоритмов связано с отысканием множеств операций, которые можно выполнять независимо друг от друга. В терминах, связанных с графом алгоритма, это эквивалентно нахождению множеств его вершин, не связанных между собой ни дугами, ни путями графа. Рассмотрим непересекающиеся множества $M_1,\ldots,$ Mp вершин графа алгоритма G. Назовем эти множества параллельными по графу G или просто параллельными, если любой путь, связывающий две вершины одного множества, целиком лежит в этом множестве и никакие две вершины из разных множеств не связаны ни дугой, ни путем графа G. Под параллельной структурой алгоритма или графа будем понимать совокупность сведений о параллельных множествах. Сюда же будем относить и сведения о тех преобразованиях, целью которых является либо выявление, либо изменение параллельных множеств. Говоря о параллелизме, обычно обсуждают два его вида: макропараллелизм и микропараллелизм. Макропараллелизм связан с ситуацией, когда все или хотя бы часть из параллельных множеств содержат много точек. Микропараллелизм имеет дело с теми случаями, при которых в каждом из параллельных множеств находится всего лишь несколько точек. Типичной для микропараллелизма является ситуация, когда все множества содержат только по одной точке.

В ближайших рассмотрениях особый интерес будут представлять различные множества, образованные группами вершин, лежащих на поверхностях уровней разверток. Выделяются два типа разверток. Один тип составляют развертки, которые обеспечивают отсутствие связей внутри множеств. Это строгие развертки. Они дают возможность обнаружить в алгоритме микропараллелизм. Второй тип составляют развертки, которые обеспечивают отсутствие связей между множествами. Такие развертки называются *расщепляющими*. Они позволяют расщепить алгоритм на не связанные между собой фрагменты или, другими словами, позволяют обнаружить макропараллелизм.

Продемонстрируем подобное расщепление на примере использования обобщенных разверток. Докажем сначала два полезных факта. Пусть для графа алгоритма G построены обобщенные развертки $f_1(x)$, $f_2(x)$. Как следует из определения разверток, функционал $f(x)=f_1(x)+f_2(x)$ также будет обобщенной разверткой. Рассмотрим какую-нибудь поверхность уровня развертки $f(x)$, содержащую не менее двух вершин-точек x_1 и x_2. Допустим, что для этих точек $f(x_1)=f(x_2)$, но $f_1(x_1)\neq f_1(x_2)$. Предположим, например, что $f_1(x_1)>f_1(x_2)$. Отсюда сразу же вытекает, что $f_2(x_1)< f_2(x_2)$. Если точки связаны путем графа

G, то для любой развертки путь может идти лишь из точки с меньшим ее значением в точку с бóльшим значением. Поэтому заключаем, что точки x_1 и x_2 не могут быть связаны путем графа G. Аналогичный вывод имеет место и в случае предположения $f_1(x_1)<f_1(x_2)$.

С другой стороны, при выполнении условий $f(x_1)=f(x_2)$ и $f_1(x_1)= f_1(x_2)$ будет также выполняться равенство $f_2(x_1)=f_2(x_2)$. Следовательно, точки x_1, x_2 из одной поверхности уровня развертки $f(x)$ будут находиться и на каких-то поверхностях уровней разверток $f_1(x)$ и $f_2(x)$. Более того, при выполнении указанных равенств для любой пары точек x_1, x_2, принадлежащих любому фиксированному множеству из одной поверхности уровня развертки $f(x)$, все точки множества будут лежать на *одной и той же* поверхности уровня развертки $f_1(x)$ и на *одной и той же* поверхности уровня развертки $f_2(x)$. Действительно, пусть в рассматриваемом множестве имеется точка x, отличная от точек x_1, x_2. Тогда при выполнении условий $f(x_1)=f(x)$ и $f_1(x_1)=f_1(x)$ для пары точек x_1, x будет выполняться и равенство $f_2(x_1)=f_2(x)$. Но значения $f_1(x_1)$ и $f_2(x_1)$ однозначно определяют поверхности уровней.

Конечно, в частном случае рассматриваемое множество может полностью совпадать со всей поверхностью уровня. Предположим, что каждая поверхность уровня развертки $f(x)$ содержится в какой-то поверхности уровня развертки $f_1(x)$ или $f_2(x)$. Все три развертки относятся к одному и тому же графу. Поэтому число всех вершин во всех поверхностях уровней для каждой развертки будет одним и тем же. Отсюда вытекает, что рассматриваемые как множества совокупности всех поверхностей уровней разверток $f_1(x)$, $f_2(x)$ и $f(x)$ совпадают.

Пусть для графа алгоритма G построены обобщенные развертки $f_1(x)$, $f_2(x)$,…, $f_s(x)$, где $s\geq 2$. Функционал $F_k(x)=f_1(x)+f_2(x)+\ldots+f_k(x)$ также будет обобщенной разверткой при любом $k\geq 1$. Теперь по развертке $F_s(x)$ в соответствии с ее поверхностями уровней расщепим множество вершин графа алгоритма на последовательно связанные между собой группы. Возьмем далее развертку $F_{s-1}(x)$ и в соответствии с ее поверхностями уровней расщепим каждую из групп на подгруппы. Каждая из подгрупп соответствует пересечению поверхностей уровней разверток $F_s(x)$ и $F_{s-1}(x)$. Допустим, что на поверхности уровня развертки $F_s(x)$ имеются такие точки x_1 и x_2, что $F_{s-1}(x_1)\neq F_{s-1}(x_2)$. Согласно сказанному выше точки x_1 и x_2 не могут быть связаны путем графа G. Вследствие условия $F_{s-1}(x_1)\neq F_{s-1}(x_2)$ они заведомо принадлежат *разным* подгруппам. По этой причине любые две точки x_1 и x_2, взятые по одной из этих двух подгрупп, не могут удовлетворять условию $F_{s-1}(x_1)=F_{s-1}(x_2)$. Поэтому подгруппы оказываются *параллельными*.

Если на каждой поверхности уровня развертки $F_s(x)$ для любой пары точек x_1 и x_2 будет выполняться равенство $F_{s-1}(x_1)=F_{s-1}(x_2)$, то в соответствии со сказанным ранее это означает, что поверхности уровней разверток $F_s(x)$, $F_{s-1}(x)$ и $f_s(x)$ совпадают. Следовательно, по сравнению с набором разверток $f_1(x)$, $f_2(x)$,…, $f_{s-1}(x)$ развертка $f_s(x)$ не добавляет новой информации в отношении

распределения вершин-точек графа по поверхностям уровней и ее можно исключить из рассмотрения. В случае успешного использования развертки $f_s(x)$ далее пытаемся расщепить подгруппы на параллельные множества с помощью развертки $F_{s-2}(x)$ и т.д.

Заметим, что сейчас не идет речь о поиске наилучших в каком-либо смысле параллельных множеств. Демонстрируется лишь возможность использования обобщенных разверток для обнаружения какого-то параллелизма.

Вообще говоря, развертки устроены достаточно сложно. Множество обобщенных разверток замкнуто в отношении некоторых операций над ними. Из определения разверток можно заключить, что обобщенной разверткой является

- сумма обобщенных разверток,
- произведение обобщенной развертки на неотрицательное число,
- максимум из обобщенных разверток,
- минимум из обобщенных разверток.

Можно также показать, что в отношении трех последних операций множество обобщенных разверток представляет *полумодуль*. В нем существуют "нулевая" и "единичная" развертка, а также "оптимальная" развертка, обеспечивающая реализацию алгоритма за минимальное время при наличии *ограничений* снизу на времена выполнения операций и времена передачи данных. Различные нетривиальные свойства разверток описаны в [1] и приведенной там литературе.

Будем по-прежнему считать, что граф алгоритма расположен в арифметическом пространстве X подходящей размерности. В этом случае без ограничения общности дуги графа можно рассматривать как векторы. С практической точки зрения целесообразно использовать самые простые развертки. Развертка задается функционалом, определенным на конечном множестве вершин-точек. Тем не менее, ничто не мешает рассматривать любые подходящие расширения функционалов на все пространство X. Ясно, что в первую очередь следует изучить возможность использования линейных функционалов.

Пусть векторы $s_1,\ldots, s_p$ описывают множество дуг графа. Предположим, что для некоторого вектора q выполняются условия $(s_i,q)>0$ ($(s_i,q)\geq 0$) для всех i. Будем называть граф *строго направленным* (*направленным*) относительно вектора q, а сам вектор q – *строго направляющим* (*направляющим*) вектором графа. Рассмотрим в пространстве X линейный функционал (x,q) и его поверхности уровней $(x,q)=c$ для различных значений константы c. Если дуга s графа идет из вершины u в вершину v, то $s=v-u$. Согласно определению, для строго направленного (направленного) относительно вектора q графа должно выполняться неравенство $(v-u,q)>0$ ($(v-u,q)\geq 0$) или $(v,q)>(u,q)$ ($(v,q)\geq(u,q)$). Поэтому для строго направленного (направленного) относительно вектора q графа функционал (x,q) определяет строгую (обобщенную) развертку. Развертки вида (x,q) будем называть *линейными*.

Уравнения $(x,q)=c$ при разных значениях c задают в X некоторое семейство гиперплоскостей. По отношению к дугам графа это семейство обладает *важными свойствами*, которые нагляднее всего описать геометрически. Именно, для строго направленного графа дуги могут проходить через любую гиперплоскость только из отрицательного (неположительного) полупространства в неотрицательное (положительное) полупространство. Никакие дуги не могут лежать на самой гиперплоскости, поскольку вершины графа на гиперплоскости представляют не что иное как поверхность уровня развертки (x,q). Для направленного графа дуги могут проходить через гиперплоскость только из неположительного полупространства в неотрицательное полупространство. Какие-то дуги могут лежать на гиперплоскости. Но ни для какого направленного графа дуги не могут пересекать ни одну гиперплоскость в *противоположных* направлениях. Единственное, что допускается, – это нахождение каких-то дуг на каких-то гиперплоскостях и только для графа, направленного не строго относительно вектора q.

Выберем *возрастающую* последовательность чисел c_0, c_1,…, c_m. Будем считать для определенности, что в отрицательном (неотрицательном) полупространстве гиперплоскости $(x,q)=c_0$ ($(x,q)=c_m$) нет ни одной вершины графа, а в каждом из полуслоев $c_j\le(x,q)<c_{j+1}$ для всех j, $1\le j\le m-1$, имеется хотя бы по одной вершине. С позиций макровычислений полуслои похожи на "толстые" поверхности уровней обобщенных разверток. Допустим, что дуга связывает две вершины из *разных* полуслоев. Если $j=1,...,m-1$ считать номером полуслоя, то любая дуга может идти лишь из полуслоя с меньшим номером в полуслой с большим номером. Это тривиальное замечание вскоре будет *эффектно* использовано. Очевидно, что соответствующие полуслоям операции алгоритма можно выполнять последовательно полуслой за полуслоем согласно росту номера j.

Пусть снова векторы s_1,…, s_p описывают все множество дуг графа. Но теперь предположим, что для графа найдено $r\ge2$ *линейно независимых* векторов q_1,…, q_r, строгой или нестрогой направленности. Для всех i, j построим по описанным только что правилам гиперплоскости $(x,q_i)=c_j^i$ и полуслои $c_j^i\le(x,q_i)<c_{j+1}^i$. Перенумеруем относящиеся к векторам q_i полуслои подряд натуральными числами α_j^i в соответствии с ростом номеров j. Пересечение любых r полуслоев, соответствующих разным векторам q_i, представляет полуоткрытый параллелепипед. Поэтому все вершины графа оказываются распределенными по некоторой r-мерной системе непересекающихся полуоткрытых параллелепипедов, гранями которых являются построенные гиперплоскости. Внутри каждого параллелепипеда расположен некоторый подграф, описывающий какой-то фрагмент алгоритма. Тем самым получено разбиение на отдельные фрагменты всего алгоритма. Основной вопрос заключается в том, возможно ли правильно реализовать алгоритм *в целом*, выполняя в каком-либо порядке *отдельные* его фрагменты, и если возможно, то *как* это делать?

Каждый параллелепипед однозначно характеризуется r-мерной совокупностью своих номеров $\alpha_1, \alpha_2,\ldots, \alpha_r$. Рассмотрим два параллелепипеда с номерами $\alpha_1, \alpha_2,\ldots, \alpha_r$ и $\beta_1, \beta_2,\ldots, \beta_r$. По построению, дуга из первого параллелепипеда может идти во второй только в том случае, когда для всех i=1, 2,…, r выполняются нестрогие неравенства $\alpha_i \leq \beta_i$ и хотя бы для одного значения i, например, равного j, имеет место строгое неравенство $\alpha_j < \beta_j$. Просуммировав почленно все эти неравенства, заключаем, что необходимо должно выполняться суммарное неравенство $\alpha_1+\alpha_2+\ldots+\alpha_r<\beta_1+\beta_2+\ldots+\beta_r$. Разобьем параллелепипеды на группы, относя к одной группе те и только те из них, которые будут иметь одинаковые суммы номеров $\alpha=\alpha_1+\alpha_2+\ldots+\alpha_r$. Как вытекает из суммарного неравенства, в одной группе не могут существовать параллелепипеды, связанные между собой дугами графа. Из него же следует, что дуга не может идти из параллелепипеда с большей суммой номеров в параллелепипед с меньшей суммой номеров. Упорядочим группы по росту суммы номеров, начиная с α=1. Возможно, некоторые из групп окажутся пустыми. Однако это не мешает выполнять группы фрагментов *последовательно* друг за другом в порядке роста суммы номеров. Внутри же каждой группы фрагменты не связаны между собой и их можно выполнять *параллельно.*

Итак, знание хотя бы двух независимых линейных разверток, причем не обязательно строгих, позволяет перейти от описания алгоритма в терминах исходных операций к описанию того же алгоритма, но уже в терминах его фрагментов или, другими словами, в терминах более крупных макроопераций. Для макроописания алгоритма легко находится параллельная форма. Чем больше известно независимых разверток, тем больше ширина ярусов у этой параллельной формы. Но чем больше ширина ярусов, тем больше параллелизма удается выявить в алгоритме. С этой точки зрения наиболее интересным является случай, когда число известных разверток совпадает с размерностью того пространства, в котором размещен граф алгоритма.

Пусть граф является строго направленным относительно какого-то вектора q. Из соображений непрерывности ясно, что всегда можно найти полный базис близких к q векторов, по отношению к которым граф также является строго направленным. Однако их прямое использование не всегда бывает целесообразным или даже становится невозможным. Если среди дуг графа много таких, которые *близки к ортогональным* по отношению к вектору q, то это приводит к появлению сильно сжатых вдоль вектора q параллелепипедов. Нередко такое сжатие оказывается тем сильнее, чем больше сам граф, что, в свою очередь, влечет за собой большие вычислительные и организационные трудности в реализации макроопераций. На практике более удобно иметь дело с направляющими векторами, близкими к ортогональным между собой, даже если по отношению к ним графы направлены и не строго.

Очень важно, что размеры всех макроопераций можно регулировать за счет выбора "толщины" полуслоев. Предположим, что макрооперации реализуются на отдельных процессорах многопроцессорной вычислительной системы. В общем случае, при увеличении параллелепипеда количество попавших в него операций алгоритма, т.е. время реализации макрооперации, растет как объем параллелепипеда. Количество же связей с другими макрооперациями, т.е. количество дуг, пересекающих грани параллелепипеда, растет как площадь его поверхности. Объем растет быстрее площади поверхности. Поэтому при увеличении макроопераций полезная загруженность процессоров будет увеличиваться, поскольку на выполнение собственно самих операций будет тратиться относительно больше времени, чем на обмен информацией с другими процессорами.

Одним из самых интересных классов алгоритмов, графы которых оказываются направленными, являются рекуррентные соотношения. Рассмотрим конечномерное пространство целочисленных вектор-индексов x с лексикографическим порядком и в нем непустую область D. Соотношения вида

$$u(x)=F_x(u(x-x_1), u(x-x_2),\ldots,u(x-x_r)),\ x\in D,$$

называются *рекуррентными соотношениями* с линейными индексами, если вектор-индексы $x_1,\ldots, x_r$ фиксированы, целочисленные и не зависят от x. Функции F_x могут быть произвольными, в том числе как линейными, так и нелинейными. Выполняются эти соотношения в порядке лексикографического роста вектор-индекса x. Если какой-либо из векторов $x-x_i$ не принадлежит области D, то соответствующая переменная $u(x-x_i)$ считается заданной и представляет одно из входных данных алгоритма.

Будем размещать вершины графа алгоритма в точках области D с целочисленными координатами. Вершине, задаваемой вектором x, поставим в соответствие функцию F_x. Если $x\in D$, то из рекуррентных соотношений вытекает, что в вершину x будут входить дуги из вершин $x-x_1,\ldots, x-x_r$ и только из этих вершин. В случае, когда какой-то из векторов $x-x_i$ не принадлежит области D, вектор $x-x_i$ будет символизировать функцию ввода переменной $u(x-x_i)$. Построенный таким образом граф имеет очень простую структуру. Если дуги задавать векторами, то в каждую вершину из области D будет входить один и тот же пучок дуг, который переносится параллельно от одной вершины к другой. Графы подобного вида называются *регулярными*, а образующие их векторы $x_1,\ldots, x_r$ – *базовыми*. Заметим, что при других размещениях вершин графа алгоритма регулярная структура дуг может нарушаться. Данный пример наглядно подтверждает важность согласования формы записи алгоритма с формой представления его графа.

Регулярные графы совсем не обязательно связывать с рекуррентными соотношениями. Важно лишь, чтобы вершины графов располагались в точках

с целочисленными координатами. Тогда сами графы можно строить, просто перенося пучок заданных базовых векторов $x_1,\ldots, x_r$ от одной вершины к другой. В общем случае такие графы могут иметь контуры, т.е. не быть графами никаких алгоритмов. Однако доказано [1], что регулярный граф, вершины которого расположены в точках с целочисленными координатами, не имеет контуры тогда и только тогда, когда существует вектор q, относительно которого граф является строго направленным. Не ограничивая общности, вектор q можно считать целочисленным. Поскольку все дуги графа описываются базовыми векторами $x_1,\ldots, x_r$, то должны выполняться строгие неравенства $(x_1,q)>0$, $(x_2,q)>0,\ldots, (x_r,q)>0$. В этом случае заведомо существует столько строгих независимых линейных разверток, какова размерность линейной оболочки векторов $x_1,\ldots, x_r$. Тем не менее, как уже отмечалось, использовать их можно лишь с определенной осторожностью. С практической точки зрения во многих случаях удобнее брать линейные обобщенные развертки, направляющие векторы которых совпадают с направляющими векторами граней выпуклого конуса, образованными векторами $x_1,\ldots, x_r$. В силу целочисленности координат базовых векторов, направляющие векторы разверток всегда можно выбрать целочисленными.

Допустим, что для регулярного графа найдена линейная развертка (x,q) с целочисленным вектором q. Так как вершины графа расположены в точках с целочисленными координатами, то уравнение любой поверхности уровня $(x,q)= c$ есть уравнение гиперплоскости с целыми коэффициентами. Очевидно, что граф покрывается конечной системой таких гиперплоскостей. Расстояние между соседними гиперплоскостями не меньше, чем $d\|q\|^{-1}_E$, где d – наибольший общий делитель модулей ненулевых координат вектора q, $\|\cdot\|_E$ – евклидова норма вектора [1]. Желание минимизировать время реализации алгоритма приводит к минимизации числа гиперплоскостей, покрывающих граф. Если не принимать во внимание частные особенности графов, то вектор q следует выбирать так, чтобы величина $d\|q\|^{-1}_E$ была максимальной.

ЛЕКЦИЯ 8

Новый математический аппарат

Содержание: *выбор формы описания алгоритмов*, *линейный класс программ, пространство итераций, размещение вершин графа, покрывающие функции, теорема об информационном покрытии, инвариантность линейных многогранников, кусочно-линейные развертки, теорема о кусочно-линейных развертках, косвенная адресация и хаос в дугах, унифицированное описание алгоритмов, локальные алгоритмы и графы, задача укладки графов.*