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.*

Из проведенных исследований видно, что граф алгоритма и развертки являются теми объектами, которые могут стать основой создания математического аппарата, предназначенного для изучения информационной структуры алгоритмов. Но чтобы эти объекты превратились в действенные инструменты, они должны быть представлены в форме, удобной для проведения как теоретических исследований, так и практических расчетов. При выборе формы необходимо учитывать, что в реальности все операции с графом алгоритма и развертками в силу их сложности придется выполнять на компьютере. Принимая во внимание необходимость обеспечения широкой доступности процессов изучения структуры алгоритмов, этот компьютер, скорее всего, должен быть персональным. Поэтому выбор форм представления графов алгоритмов и разверток должен учитывать и эффективность предстоящей работы с ними.

В теоретическом отношении вроде бы все устроено достаточно хорошо. Представление графа алгоритма ориентированным ациклическим графом, а разверток вещественными функционалами оказалось эффективным. Кажется, что использование линейных функционалов в качестве разверток открывает неплохие практические перспективы. Однако все предположения, которые до сих пор были сделаны относительно графа алгоритма, к реальной практике имеют весьма слабое отношение.

Одно из первых теоретических предположений связано с размещением графа в некотором арифметическом пространстве подходящей размерности. Такое предположение существенно, так как с помощью достаточно простых средств позволяет обеспечить возможность построения разверток, выполнение операций над ними, введение линейных разверток через скалярное произведение и т.д. Но как *конструктивно* строить это пространство, какова его размерность и *как именно* в нем должны быть расположены вершины? Ведь уже на примере регулярных графов было отмечено, что допущение излишней свободы в размещении вершин может привести к потере важных структурных свойств. И вообще, *откуда* брать информацию о графе алгоритма? Он же почти никогда не бывает известен ни в теории, ни тем более на практике! В реальных алгоритмах число вершин графа настолько велико, что становится ясно: не может даже идти речь ни о каком описании графов, основанном на прямом перечислении всех его вершин и дуг. Но тогда как же его описывать и как исследовать?

Ответы на все эти вопросы надо искать на пути анализа используемых форм записи алгоритмов. Других источников, из которых можно было бы получить информацию о графах алгоритмов, просто не существует. Заметим, что для описания алгоритмов массово используются только две формы – это записи в виде различных математических соотношений и программы на алгоритмических языках. У каждой из форм имеются как достоинства, так и недостатки. Основное достоинство программ заключается в том, что с их помощью можно дать *точное* описание алгоритма без каких-либо недомолвок

и неоднозначностей. Однако их очень трудно анализировать, главным образом, из-за пересчета содержимого ячеек памяти. В математических записях нет пересчета памяти, и в этом заключается их большое преимущество перед программами. Но обычно в них остаются различные недомолвки и неоднозначности, особенно в отношении порядка выполнения операций. Поэтому для точного изложения содержания математических записей все равно приходится пользоваться какими-то алгоритмическими языками. Наиболее приемлемыми для выявления сведений о графе алгоритма могли бы быть программы на языках однократного присваивания, поскольку в них нет пересчета памяти, но сам алгоритм описывается точно. К сожалению, багаж алгоритмов, записанных на таких языках, очень скуден.

Принимая во внимание высказанные соображения, остановимся на языке типа фортран в качестве формы описания алгоритмов. Аргументов в пользу выбора данного языка несколько. Во-первых, он наиболее близок к математическому описанию алгоритмов. Во-вторых, в вычислительной математике этот язык всегда был и остается до сих пор самым используемым языком программирования. И, наконец, на нем накоплен самый большой в мире багаж алгоритмов. Иметь возможность использовать такой багаж для выявления структуры алгоритмов в вычислительной математике – заманчивая перспектива.

Не все конструкции языка фортран будут использоваться в одинаковой мере. Для начала исследований выделим относительно простой, но достаточно содержательный класс программ. На нем попытаемся понять основные особенности устройства графов алгоритмов. И только после этого приступим к расширению выбранного класса. Будем считать сейчас, что алгоритм записан с помощью следующих средств языка:

– в программе может использоваться любое число простых переменных и переменных с индексами;

– единственным типом исполнительного оператора может быть оператор присваивания, правая часть которого есть арифметическое выражение; допускается любое число таких операторов;

– все повторяющиеся операции описываются только с помощью циклов DO; структура вложенности циклов может быть произвольной; шаги изменения параметров циклов всегда равны +1; если у цикла нижняя граница больше верхней, то цикл не выполняется;

– допускается использование любого числа условных и безусловных операторов перехода "вниз" по тексту; не допускается использование побочных выходов из циклов;

– все индексные выражения переменных, границы изменения параметров циклов и условия передач управления заданы явно; они являются *линейными* функциями как по параметрам циклов, так и по внешним переменным программы; коэффициенты всех линейных функций являются целыми числами;

– внешние переменные программы (размеры массивов и т.п.) всегда целочисленные, и вектора их значений принадлежат некоторым целочисленным многогранникам;

– в целях наглядности описания программы, а также для организации управления вычислительным процессом все или часть операторов могут быть помечены.

Программы, удовлетворяющие описанным условиям, будем называть *линейными* или принадлежащими *линейному классу*. Они и будут предметом ближайших исследований. Обратим внимание на одно очень важное обстоятельство, вызывающее, к тому же, немалые трудности при проведении этих исследований. Именно, все возникающие задачи придется решать в условиях, когда конкретные значения внешних переменных известны только перед началом работы программы и неизвестны в момент ее исследования. Поэтому все задачи неизбежно окажутся задачами с *неизвестными параметрами*.

Особая специфика записи операторов не потребуется. Если нужно приблизить форму записи алгоритмов к математической, то будем писать индексы переменных так же, как в математических соотношениях, т.е. справа от переменных снизу и/или сверху. Сделаем также некоторое уточнение, касающееся переменной с индексами. Обычно под этим понятием рассматривается весь массив простых переменных, объединенных общим идентификатором. При изучении тонкой структуры программы гораздо удобнее рассматривать массив как группу простых переменных, идентификаторы которых составлены из идентификатора массива и индексов. Всюду в дальнейшем будем понимать под переменной с индексами *отдельный* элемент массива.

Как показывает статистика, многие значимые фрагменты практически используемых программ непосредственно принадлежат линейному классу. Еще большее число фрагментов сводится к нему. Линейный класс играет *такую же* роль в изучении структуры программ как линейные функции в математическом анализе, линейные неравенства в задачах оптимизации, линейная алгебра в вычислительной математике и т.п.

Важнейшая цель ближайших исследований состоит в нахождении графа алгоритма, описанного программой из линейного класса. Подчеркнем, что достижение цели планируется обеспечить исключительно на основе анализа *текста программ* без привлечения какой-либо дополнительной информации. В силу большого объема выкладок мы не будем проводить сейчас детальные исследования, а ограничимся описанием лишь их общей схемы. Все необходимые подробности можно найти в [1].

Прежде всего нужно точно описать множество вершин графа алгоритма. Пусть задана произвольная линейная программа. Перенумеруем подряд сверху вниз все операторы присваивания и обозначим их через $F_1,\ldots, F_m$. С каждым оператором присваивания $F_i$ свяжем тесно вложенное гнездо циклов. Оно

получается путем оставления в программе только тех циклов DO, в тела которых входит рассматриваемый оператор. Рассмотрим арифметическое пространство, координаты которого определяются параметрами данного гнезда. Назовем это пространство *опорным* для оператора $F_i$ и будем считать, что в нем естественным образом введены линейные операции и скалярное произведение. Границы изменения параметров определяют в опорном пространстве линейный многогранник, который будем также называть *опорным* и обозначать $V_i$. Его границы зависят от внешних переменных. Поэтому в общем случае от них будут зависеть размеры всех опорных многогранников и их конфигурации. Как правило, с ростом значений внешних переменных размеры многогранников неограниченно увеличиваются. Напомним, что значения внешних переменных на момент проведения исследований *не известны*.

Каждая операция алгоритма *однозначно* определяется номером $i$ соответствующего оператора $F_i$ и значениями параметров относящегося к нему гнезда циклов. Исторически принято отдельное срабатывание оператора $F_i$ называть *итерацией.* Поэтому совокупность опорных областей $V_i$ для $i$=1, 2,…, $m$ будем называть *пространством итераций.* Оно состоит из линейных многогранников, принадлежащих *разным* арифметическим пространствам, имеющим, чаще всего, *разные* размерности. Факт, что некоторые или даже все многогранники могут порождаться одними и теми же значениями параметров циклов, не имеет сейчас никакого значения. Это будет отражаться лишь в том, что такие многогранники будут в чем-то похожи по расположению в своих пространствах и иметь какие-то размеры одинаковыми. В пространстве итераций положение всех вершин графа алгоритма определено однозначно. Конечно, построенное пространство не является арифметическим, как это предполагалось ранее. Однако ничто не мешает при необходимости расширить пространство итераций до арифметического пространства, считать вершины графа векторами, ввести линейные операции над ними и т.д.

Значительно сложнее подобраться к пониманию того, в какой форме должны описываться дуги. Представление графа алгоритма произвольным ориентированным ациклическим графом не подсказывает никакой идеи о формах их описания в реальных алгоритмах. Может даже создаться впечатление, что допустимым является любое расположение дуг вплоть до хаотического. Как будто бы близкое к этому расположение даже реализуется на алгоритмах, связанных с нерегулярными и адаптивными сетками. С другой стороны, большого хаоса вроде бы не должно быть. Дуги графа алгоритма отражают информационные отношения между отдельными операциями. Эти же отношения, но только в скрытом виде, присутствуют и в математических записях, и в программах. Принимая во внимание огромный объем вычислений в реальных задачах, хаотические информационные отношения между отдельными операциями могут описываться только через очень длинные записи и программы. Однако на практике и математические записи и

программы, как правило, достаточно компактны. Поэтому, скорее всего, дуги графов реальных алгоритмов должны описываться как-то иначе и проще. Например, с помощью каких-то не слишком сложных функций. Начнем исследование возможных форм представления дуг с линейного класса программ.

Количество входных переменных в каждом операторе конечно. Поэтому формально граф алгоритма можно описать некоторой конечнозначной функцией Ф. Для любой точки $I$ пространства итераций множество значений функции $\Phi(I)$ есть множество тех точек того же пространства, из которых идут дуги графа в точку $I$. В общем случае число значений функции Ф может быть различным в разных точках. Более того, разные значения функции Ф в одной точке $I$ могут принадлежать *разным* опорным многогранникам. Всегда существуют точки, в которых функция Ф не определена. К ним относятся точки, соответствующие операциям, где в качестве аргументов используются лишь входные данные. Принципиальный вопрос состоит в выяснении того, как устроена функция Ф.

Пусть в пространстве итераций задана система однозначных функций $\Phi_k$. Предположим, что область определения каждой из них принадлежит какому-то одному опорному многограннику, а область значений – тому же или другому многограннику. Допустим, что для каждой точки $I$ пространства итераций при любых значениях внешних переменных найдется среди функций $\Phi_k$ такая подсистема, что множество значений функции $\Phi(I)$ совпадает со значениями в точке $I$ функций из данной подсистемы. Будем говорить, что в этом случае граф *покрывается* системой функций $\Phi_k$, а сами функции $\Phi_k$ будем называть *покрывающими*. Для всех исследований, проводимых с графом алгоритма, фундаментальное значение имеет [1]

*Теорема об информационном покрытии*. Для любой линейной программы граф алгоритма покрывается конечной системой функций, линейных как по пространству итераций, так и по пространству внешних переменных программы. Число этих функций не зависит от значений внешних переменных. Каждая из функций определена на линейном многограннике, расположенном в одном из опорных многогранников со значениями в том же или другом опорном многограннике. Грани многогранников описываются уравнениями, также линейными как по пространству итераций, так и по пространству внешних переменных.

Утверждение теоремы кажется удивительным во многих отношениях. В самом деле, если программа зависит от внешних переменных, то от их значений будет зависеть и граф алгоритма. Из определения графа алгоритма не видно, как влияют на его структуру внешние переменные. Однако, как уже отмечалось, при увеличении значений внешних переменных граф может увеличиваться неограниченно. По условию принадлежности программы линейному классу все свободные члены гиперплоскостей, описывающих грани

опорных многогранников, являются функциями, линейно зависящими от внешних переменных. Согласно утверждению теоремы вся зависимость покрывающих функций от внешних переменных сосредоточена в их свободных членах и в свободных членах гиперплоскостей, описывающих грани областей определения. И снова все свободные члены являются функциями, линейно зависящими от внешних переменных. В дополнение к этому напомним, что направляющие векторы гиперплоскостей-граней всех участвующих в рассмотрении многогранников от внешних переменных не зависят. Поэтому все многогранники обладают очень характерным свойством: направления всех их ребер *не зависят* от значений внешних переменных.

Вершины графа алгоритма любой программы из линейного класса расположены в некоторой системе многогранников, зависящих от внешних переменных. Многогранники меняют свои размеры и конфигурации при изменении этих переменных. Но все эти изменения таковы, что меняются только длины ребер многогранников, а направления ребер остаются постоянными. Разве можно обнаружить это свойство, лишь разглядывая текст линейной программы?

Число покрывающих функций зависит от многих факторов. В частности, оно зависит от арифметической природы коэффициентов линейных выражений и числа исполняемых операторов программы. На практике число покрывающих функций оказывается не очень большим. Исследование реальных линейных программ показало наличие двух особенностей. Во-первых, общее число линейных покрывающих функций пропорционально числу операторов присваивания в программе. Коэффициент пропорциональности не превосходит нескольких единиц. На меньшее число функций рассчитывать не приходится, если операторы связаны друг с другом. Во-вторых, почти всегда система покрывающих функций описывает граф алгоритма *абсолютно точно*. В этих случаях для любой покрывающей функции $\Phi_k$ и любой целочисленной точки $I$ из ее области определения пара точек $\Phi_k(I)$ и $I$ задает дугу графа алгоритма. Известно лишь несколько реальных примеров, в которых системы покрывающих функций определяют не граф алгоритма, а какое-то его *расширение*.

Итак, вопрос о форме представления дуг в графах алгоритмов, когда сами алгоритмы описаны программами из линейного класса, полностью исследован. Знание одних только покрывающих функций позволяет решать многие интересные задачи: определять параллелизм в циклах, выявлять избыточные вычисления, восстанавливать из программ математические соотношения и т.п. Поэтому, естественно, возникает вопрос о разработке метода определения покрывающих функций по текстам программ. Для программ из линейного класса такой метод построен [1]. Он достаточно сложен, но эффективен по времени реализации. Тем не менее, мы не будем здесь его обсуждать, поскольку показ устройства самого метода не демонстрирует никакие новые

идеи, полезные для понимания процессов изучения информационной структуры алгоритмов.

Несколько ранее было показано, что линейные обобщенные развертки могут служить весьма эффективным инструментом для изучения структуры алгоритмов, в том числе, на макроуровне. Однако необходимым условием для их применения являлось размещение вершин графа алгоритма в линейном арифметическом пространстве со скалярным произведением. В целом пространство итераций не является таковым. Поэтому прежде всего нужно понять, можно ли в этом пространстве построить аналог обобщенных линейных разверток. Рассмотрим в пространстве итераций вещественный функционал, обладающий следующими свойствами:

– его область определения есть линейный замкнутый многогранник, принадлежащий одному из опорных многогранников;

– он линеен как по точкам опорного пространства, так и по внешним переменным.

Будем искать развертки, представленные системой подобных функционалов. Если такие развертки существуют, то будем их называть *кусочно-линейными*.

Имеется определенное сходство между системой покрывающих функций и рассматриваемыми линейными функционалами. Но между ними существует и принципиальное различие. Пусть граф алгоритма представлен системой покрывающих функций. Многогранники, на которых они определены, могут пересекаться по пространству итераций и никогда не покрывают его полностью. Допустим, что развертку можно представить системой линейных функционалов. В этом случае многогранники, на которых заданы функционалы, не могут пересекаться по пространству итераций и обязаны в совокупности покрывать его полностью.

Рассмотрим случай, когда область определения каждого из линейных функционалов совпадает с одним из опорных многогранников $V_1,\ldots, V_m$. Пусть покрывающие функции графа алгоритма определены на многогранниках $V_{ij}$. Как уже говорилось, эти многогранники могут пересекаться и иметь разные размерности. Если многогранник $V_{ij}$ имеет непустое пересечение с многогранником $V_i$, то по построению он входит в $V_i$ полностью. Рассмотрим векторы $N = (N_1,\ldots, N_s)$ внешних переменных. Предположим, что они принадлежат некоторой совокупности многогранников, в общем случае неограниченных. Ясно, что координаты вершин всех рассматриваемых здесь многогранников $V_1, \ldots, V_m$ и $V_{ij}$ являются неоднородными линейными функциями переменных $N_1,\ldots, N_s$. Координаты же вершин многогранников, задающих внешние переменные, постоянны.

Допустим, что заданная на многограннике $V_{ij}$ покрывающая функция имеет вид $x=Jy+\varphi$ и ее значения принадлежат $V_k$. Здесь $J$ есть числовая матрица, вектор $\varphi$ линейно зависит от внешних переменных. Согласно теореме об информационном покрытии, граф алгоритма линейной программы удовлетворяет этим требованиям. Пусть обобщенная развертка имеет вид (*b,*

$y)+\delta$ на $V_i$ и вид $(a, x)+\gamma$ на $V_k$. Направляющие векторы $a$, $b$ и свободные члены $\gamma$, $\delta$ неизвестны и подлежат нахождению. Будем искать векторы $a$, $b$ как не зависящие от переменных $N_1,\ldots, N_s$, а свободные члены $\gamma$, $\delta$ как неоднородные линейные функции от этих переменных. Так как из функционалов $(a, x)+\gamma$ и $(b, y)+\delta$ должна составляться обобщенная развертка, то для всех $y\in V_{ij}$ обязано выполняться неравенство $(a, x)+\gamma\leq(b, y)+\delta$ или, другими словами $(J^T a-b, y)\leq-(a, \varphi)+\delta-\gamma$.

Обозначим через $y^l$ вершины многогранника $V_{ij}$. Из теории линейных неравенств известно, что каждая точка ограниченного линейного многогранника может быть представлена как выпуклая линейная комбинация его вершин. Следовательно, последнее неравенство эквивалентно такой системе неравенств $(J^T a-b, y^l)\leq-(a, \varphi)+\delta-\gamma$. Векторы $y^l$ и $\varphi$ являются неоднородными линейными функциями от $N_1,\ldots, N_s$. Эти переменные принадлежат некоторой системе многогранников, в общем случае неограниченных. Каждая точка неограниченного линейного многогранника может быть представлена как выпуклая линейная комбинация его вершин и направляющих векторов неограниченных ребер. Принимая во внимание это представление, уже не трудно свести последнюю систему неравенств к системе неравенств относительно координат векторов $a$, $b$ и коэффициентов разложения свободных членов $\delta$, $\gamma$ по системе параметров $1, N_1,\ldots, N_s$ [1].

Обозначим через $t$ вектор, составленный для всех многогранников $V_k$. из координат направляющих векторов $a$ и коэффициентов разложения свободных членов $\gamma$ по параметрам $1, N_1,\ldots, N_s$. Будем называть его *направляющим вектором* кусочно-линейной развертки. Зафиксируем какой-нибудь многогранник, в котором определен вектор внешних переменных $N$. Соберем далее вместе неравенства для всех покрывающих функций графа. Левую их часть можно представить в виде произведения $At$, где $A$ есть числовая матрица. Проведенные исследования показывают, что имеет место

*Теорема о кусочно-линейных развертках*. Для любой программы из линейного класса и любого многогранника, задающего область изменения внешних переменных, существует не зависящая от значений внешних переменных матрица $A$ такая, что любой ненулевой вектор $t$, удовлетворяющий векторному неравенству $At\leq 0$, является направляющим вектором кусочно-линейной развертки.

Эта теорема совместно с теоремой об информационном покрытии создает мощный математический аппарат для исследования структуры графов алгоритмов. Находить развертки из системы неравенств можно, например, с помощью симплекс-метода. Ответ на вопрос, существуют ли кусочно-линейные развертки, не связанные с решением неравенства $At\leq 0$, определяется тем, насколько избыточно набор покрывающих функций описывает граф алгоритма. Как уже отмечалось, графы алгоритмов для реальных программ почти всегда описываются точно. Если многогранники $V_i$ и $V_{ij}$ фиксированы,

то остается зависимость матрицы $A$ от многогранника, в котором заданы внешние переменные. В том случае, когда область определения внешних переменных состоит из нескольких многогранников, направляющие векторы кусочно-линейных разверток для каждого из многогранников будут удовлетворять своей системе неравенств.

Знание графа и его разверток для алгоритма, описанного программой из линейного класса, позволяет решать практически все вопросы, связанные с его информационной и, в том числе, параллельной структурой [1]. Принадлежность программы к линейному классу устанавливается довольно просто – нужно всего лишь проверить выполнение описанных выше условий. Однако имеется огромное число программ, формально не являющихся линейными, но которые могут быть сведены к таковым после некоторых преобразований.

Набор таких преобразований очень велик и постоянно расширяется. В частности, в него входят прямая подстановка переменных при вычислении параметров циклов, преобразование циклов *go to* в циклы DO, уточнение вида многогранников $V_i$ за счет учета влияния ветвлений, а также более аккуратное описание многогранников, задающих внешние переменные. Нередко при написании программ возникают нелинейные индексные выражения просто потому, что есть настоятельная необходимость экономии памяти при задании массивов данных. В действительности анализ структуры таких программ не намного сложнее анализа линейных программ. Как правило, при анализе структуры программ не требуется детально учитывать внутреннюю структуру подпрограмм, функций и каких-то отдельных частей самих программ. Замена соответствующих фрагментов условными программами специального вида также дает возможность ограничиться рассмотрением программ из линейного класса. С различными преобразованиями программ к линейным можно познакомиться в [1].

Необходимым этапом выполнения анализа структуры реальных программ является приведение самих программ к некоторому каноническому виду. При этом наибольшие трудности вызывает процесс распутывания их текстов. По существу, общий успех анализа определяется как раз тем, насколько успешно будет проведено это распутывание. Практика показывает поразительные примеры изощренного запутывания текстов программ. Иногда запутывание определяется желанием что-то оптимизировать, но гораздо чаще оно связано с индивидуальным стилем программирования. Стиль программирования редко учитывает то обстоятельство, что при переходе на вычислительные системы другой архитектуры программы придется переписывать. Пока еще правило "программировать надолго – это программировать просто" не стало руководством к действию. Правда, надо признать, что не всегда понятно, что значит "программировать просто". Ясно лишь одно

*– простота программирования должна начинаться с простоты, точности и тщательной структурированности математических описаний алгоритмов.*

Изложенный выше метод построения графа алгоритма для программ из линейного класса связан с использованием некоторой технологии, основанной на сравнении индексных выражений [1]. Она непосредственно применима только в тех случаях, когда все индексные выражения в тексте программы заданы *явно*. Но индексные выражения в программах могут задаваться и неявно с помощью так называемой *косвенной адресации*. Никаких формальных ограничений на характер индексных выражений при их неявном задании языки программирования не накладывают. Поэтому кажется, что именно косвенная адресация может быть источником появления хаоса в дугах графа алгоритма. Как уже отмечалось, вроде бы близкое к этому расположение дуг даже реализуется на алгоритмах, связанных с нерегулярными и адаптивными сетками. На самом деле разработчики программ, реализующих такие и подобные им алгоритмы, мыслят не терминами хаотических отношений между отдельными операциями, а реализуют какие-то иные, вполне определенные образы отношений. Эти образы и накладывают соответствующие ограничения на реальную косвенную адресацию при разработке конкретных алгоритмов. Для анализа структуры алгоритмов важно уметь выявлять и использовать возникающие ограничения.

Приведем пример того, как можно учитывать подобные ограничения. Пусть в пространстве $X$ задана область $D$. Предположим, что алгоритм сводится к построению в области $D$ некоторой упорядоченной последовательности точек и в вычислении в этих точках значений каких-то векторных функций. Точки последовательности могут выбираться как угодно. В частности, часть из них или даже все они могут многократно повторяться. Функции могут быть сколь угодно сложными, в том числе, разными в разных точках. Важно только одно – *откуда берутся аргументы* при вычислении очередного значения функции.

Допустим, что в пространстве $X$ фиксирован замкнутый острый конус $K$, боковая поверхность которого образована системой гиперплоскостей. Назовем этот конус *опорным* и будем перемещать его параллельно самому себе в пространстве $X$. Единственное ограничение на построение алгоритма состоит в следующем: если значение функции вычисляется в точке $x$ из области $D$, то аргументы функции могут браться лишь из тех построенных точек последовательности, которые попадают в опорный конус, при условии, что вершина конуса находится в точке $x$. Очень часто опорный конус берется *ограниченным*. В этом случае все аргументы для точки $x$ берутся от точек, принадлежащих некоторой ее локальной окрестности. Построенные таким образом алгоритмы и их графы называются *локальными*. Отметим, что все локальные алгоритмы легко разбиваются на параллельно реализуемые фрагменты. Более того, не известны никакие другие массово используемые структуры, кроме локальных, которые легко масштабируются под требования многопроцессорных систем с распределенной памятью.

Алгоритм, представленный в описанном виде, имеет много достоинств. Не требуется проявлять никакой особой заботы о регулярности расположения дуг.

Любой подобный алгоритм прекрасно распараллеливается, так как имеет полный набор обобщенных линейных разверток. В качестве их можно взять, например, линейные функционалы с направляющими векторами, совпадающими с направляющими векторами гиперплоскостей, образующих грани опорного конуса. Эта схема представляет яркий пример алгоритма, имеющего направленный граф.

Как показывает практика, огромное число самых разных алгоритмов укладывается в описанную схему. Исходя из традиционных описаний алгоритмов, не всегда в конкретных случаях сразу видны подходящие образы пространства $X$, области $D$, опорного конуса $K$ и последовательности перебора точек. Но такие образы почти всегда находятся.

С помощью введения какого-то числа дополнительных условных передач управления любую программу можно свести к тесно вложенному гнезду циклов. Это может подсказать, как должны выглядеть пространство $X$ и область $D$. Кроме этого, возможность сведения программы к гнезду циклов подсказывает, что по-видимому во многих случаях граф из пространства итераций, которое вообще-то устроено достаточно сложно, можно уложить в какое-то пространство малой размерности. Практика говорит о том, что такую укладку не только можно осуществить достаточно часто, но и сами уложенные графы при этом имеют весьма изящную структуру. Следовательно, такую же изящную структуру имеют и сами алгоритмы. Задача переноса графа алгоритма из пространства итераций в подходящее пространство малой размерности называется задачей *укладки графа*. Она исключительно интересна и имеет прекрасные прикладные перспективы. Предположим, что каждый используемый алгоритм будет описан графом, по которому сразу можно сказать все о структуре самого алгоритма. Насколько проще станет их изучение и использование!

Иногда для поиска подходящих образов приходится обращаться к каким-то иным интерпретациям алгоритмов. Если исходить из геометрической интерпретации, то в приведенное выше описание сразу укладываются все алгоритмы, использующие явные разностные схемы, в том числе, с нерегулярными и адаптивными сетками. Но если за основу взять матричную их интерпретацию, а в качестве одного из важнейших модулей выбрать умножение разреженной матрицы на вектор, то в такой интерпретации трудно что-то разглядеть.

В связи со сказанным возникает следующая идея. Известно, что одна из трудностей восприятия численных методов, особенно в отношении информационной и параллельной структуры, связана с отсутствием единой методологической основы их изложения. Предложенная схема вполне подходит для такой основы. Она очень проста и для ее понимания не требуются никакие специальные знания об алгоритмах. Начинать обсуждать схему можно на самых ранних этапах компьютерного образования. Наполнять же ее конкретным содержанием вполне возможно по мере необходимости.

Схему легко модифицировать, отказавшись от требований остроты и замкнутости опорного конуса. Подобная модификация позволит распространить ее на алгоритмы с ограниченным ресурсом параллелизма. Единственное, чего нехватает для реализации данной идеи, – это достаточного числа наполнений предложенной схемы конкретными алгоритмами. Но можно надеяться, что создание таких наполнений является лишь вопросом времени.

# ЛЕКЦИЯ 9

## Типовые информационные структуры

Содержание: *перемножение матриц, решение треугольных систем, неожиданный эффект, система с блочно-двухдиагональной матрицей, макро- и микрореализации, явная схема для уравнения теплопроводности, макро- и микропараллелизм, локальный алгоритм, очень "простой" пример, гипотеза о типовых структурах*.

Рассмотрим некоторые примеры графов конкретных алгоритмов. Мы не будем строить графы в пространствах итераций и не будем описывать их покрывающие функции. Со всем этим можно познакомиться в [1]. Мы будем размещать графы в подходящих пространствах малой размерности. Выбор размещения будет определяться только тем, чтобы можно было легко показать информационную структуру самих алгоритмов и продемонстрировать возможные наборы линейных разверток. В соответствии с определением графа алгоритма не нужно указывать дуги, связанные с рассылкой одних и тех же входных данных по вершинам графа. И очень часто они показываться не будут, главным образом, из-за сложности их изображения. Но иногда мы будем отступать от этого правила.

**Перемножение матриц.** Пусть решается классическая задача вычисления произведения $A = BC$ двух квадратных матриц $B$, $C$ порядка $n$. Будем считать элементы матриц $A$, $B$, $C$ числами и обозначим их $a_{ij}$, $b_{ik}$, $c_{kj}$. Согласно определению операции умножения матриц имеем

$$a_{ij} = \sum_{k=1}^{n} b_{ik} c_{kj}, \quad i, j = 1, 2, \ldots, n.$$

Эти формулы довольно часто используются для непосредственного вычисления элементов матрицы $A$. Сами по себе они не определяют алгоритм однозначно, так как не определен порядок суммирования произведений $b_{ik}c_{kj}$ под знаком суммы. Однако заметим, что явно виден параллелизм вычислений. Выражается он отсутствием указания о соблюдении какого-либо порядка перебора индексов $i$, $j$.