П. И. Соболевский, Е. В. Адуцкевич

ПРОЕКТИРОВАНИЕ СПЕЦПРОЦЕССОРОВ НА СБИС (ОТ АЛГОРИТМА К АРХИТЕКТУРЕ)

Пособие для студентов механико-математического факультета

Минск
БГУ
2008

УДК 004.383'144:621.3.049.771.15(075.8)
ББК 32.973.26-02я7
С54

Рекомендовано
Ученым советом механико-математического факультета
31 октября 2007 г., протокол № 2

Рецензенты:
доктор физико-математических наук,
профессор *Н. А. Лиходед*;
доктор физико-математических наук,
профессор *В. М. Котов*

Соболевский, П. И.

С54 Проектирование спецпроцессоров на СБИС (от алгоритма к архитектуре) : пособие для студентов механико-математического факультета/ П. И. Соболевский, Е. В. Адуцкевич. – Минск : БГУ, 2008. – 126 с. : ил.

ISBN 978-985-485-988-0.

Излагается методика построения специальных представлений алгоритмов и способов их отображения на многопроцессорные матричные устройства. Принципиальной особенностью данного подхода является то, что в нем разработан формализованный метод получения архитектуры спецпроцессора и описания функционирования процессорных элементов исходя из описания алгоритма. Предназначено для студентов, магистрантов и аспирантов, обучающихся по специализации 1-31 03 01-01 25 «Математическая электроника».

УДК 004.383'144:621.3.049.771.15(075.8)
ББК 32.973.26-02я7

ВВЕДЕНИЕ

Значительные достижения в области микроэлектроники дали мощный импульс к появлению и быстрому развитию новых направлений в физике, математике, информатике. Так как возможности физических усовершенствований современных вычислительных устройств имеют естественные ограничения (размеры элементарных частиц, применяемых для хранения информации, скорость распространения электромагнитных сигналов и т. п.), то особые надежды на увеличение производительности в последние годы связывают с применением математических средств проектирования спецпроцессоров, обеспечивающих высокую степень соответствия (адекватности) реализуемых алгоритмов и архитектуры реализующих эти алгоритмы устройств.

Процесс изготовления кристаллов на основе СБИС-технологии накладывает ряд естественных ограничений на архитектуру оптимального вычислительного устройства. Существуют две исключительно важные математические задачи: задача построения специальных алгоритмов, хорошо приспособленных для реализации на СБИС-процессорах, и задача отображения уже известных алгоритмов на эти процессоры.

Требованиям СБИС-технологии в наибольшей степени отвечают матричные процессоры с архитектурой систолического типа (систолические процессоры). Под ними понимают специализированные многопроцессорные устройства, характеризующиеся регулярностью внутренней структуры, пространственной локальностью связей, временной локальностью, наличием небольшого числа типов процессорных элементов, расположением портов ввода-вывода только в граничных процессорных элементах, ритмичностью обработки и распространения данных, конвейерностью и параллельностью вычислений. Под регулярностью внутренней структуры понимается однородность, повторя-

емость связей и расположений всех типов процессорных элементов. Пространственная локальность связей означает, что обмен данными производят только соседние процессорные элементы. Временная локальность означает, что на передачу данных от одного процессорного элемента к другому затрачивается хотя бы одна единица времени. Конвейерность и параллельность вычислений предполагает, что потоки данных движутся между параллельно работающими процессорными элементами; результаты вычислений формируются по мере продвижения данных.

Методы синтеза систолических структур для реализации заданных алгоритмов базируются на построении специальной вычислительной модели алгоритма и отображении ее на систолическую структуру. Следует отметить, что способы отображения алгоритмов на систолические структуры можно рассматривать как способы построения параллельных форм алгоритмов для реализации на систолических процессорах. Такого рода методы разрабатывались и исследовались в Институте математики Национальной академии наук Беларуси, в том числе и авторами данного пособия.

В пособии излагается методика построения специальных представлений алгоритмов и способов их отображения на многопроцессорные матричные устройства.

Глава 1

МАТЕМАТИЧЕСКАЯ МОДЕЛЬ АЛГОРИТМА

1.1. АЛГОРИТМЫ И ИХ ГРАФЫ ЗАВИСИМОСТЕЙ

Когда говорят об алгоритме, то обычно предполагают наличие какого-нибудь правила, описывающего совокупность выполняемых операций и связей отдельных операций между собой, тем самым определяя отношение частичного порядка на множестве операций.

Будем считать, что алгоритм или описывающее его правило позволяет определить:

• множество переменных, в преобразовании которых заключается реализация алгоритма;

• множество операций, выполняемых в процессе реализации алгоритма;

• соответствие, показывающее, какие результаты операции являются аргументами для других операций.

Будем предполагать, что число аргументов и результатов каждой из операций алгоритма, в терминах выбранных переменных, ограничено и не зависит от общего числа всех операций алгоритма.

Под графом алгоритма (графом зависимостей алгоритма) понимают математическую модель алгоритма, построенную следующим образом:

1) множеству операций алгоритма поставим во взаимно однозначное соответствие некоторое множество точек V, часто это точки d-мерной целочисленной решетки $\mathbf{Z}^d$;

2) если аргумент операции есть результат выполнения другой операции, то соответствующие точки соединим дугой, исходящей из точки, откуда берется результат. Такие вершины графа будем называть *информационно зависимыми*. Множество дуг, определяемое множеством пар информационно зависимых вершин, обозначим E. Если аргумент является начальным данным или результат операции нигде не используется, то будем считать, что соответствующие дуги либо отсутствуют, либо не имеют одной из вершин.

Построенный таким образом граф $G = (V, E)$ алгоритма является

ориентированным ациклическим мультиграфом. Кроме того, это помеченный граф (вершины помечены операциями).

Пример 1.1. Рассмотрим алгоритм:

$$
\begin{aligned}
&a(i,j,k) = a(i-1,j,k),\\
&b(i,j,k) = b(i,j-1,k),\\
&c(i,j,k) = a(i-1,j,k)\cdot b(i,j-1,k) + c(i,j,k-1),
\end{aligned}
$$

где $1 \le i,j,k \le N$; a,b,c – переменные алгоритма, операции алгоритма: «умножение со сложением и переопределение». Каждому набору значений i,j,k соответствует одна операция. Таким образом, имеем N^3 операций. Поставим множество операций алгоритма во взаимно однозначное соответствие со множеством точек 3-мерной целочисленной решетки $V = \{\,(i,j,k) \in \mathbf{Z}^3 \mid 1 \le i,j,k \le N\,\}$. На рис. 1.1 изображен граф зависимостей алгоритма при $N = 2$.

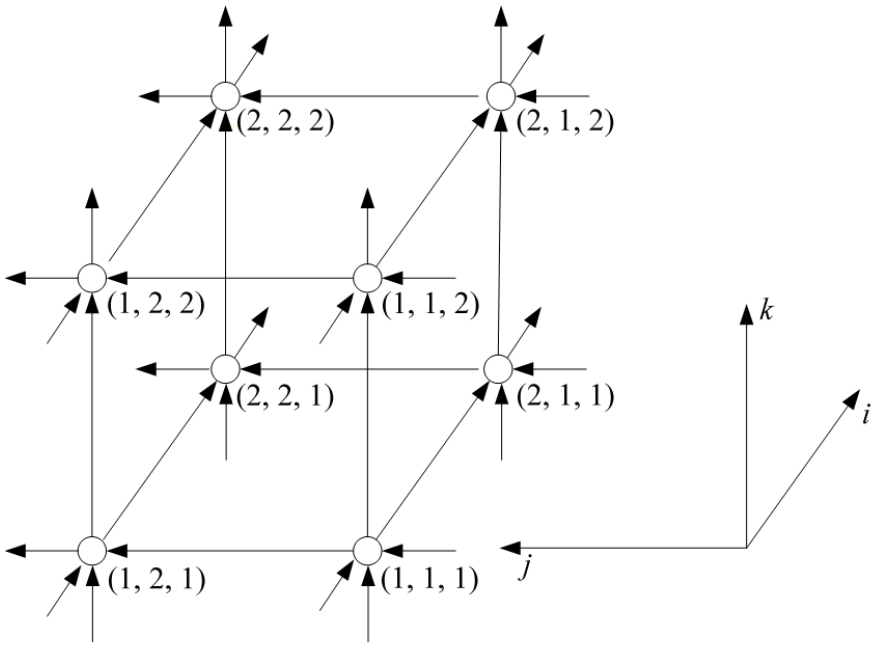

Рис. 1.1. Граф зависимостей алгоритма при $N = 2$

1.2. ПАРАЛЛЕЛЬНЫЕ ФОРМЫ АЛГОРИТМА

Пусть задан некоторый алгоритм, и $G = (V, E)$ является его графом зависимостей. Разбиение множества вершин графа алгоритма V на N непересекающихся подмножеств Υ_s: $\bigcup\limits_{s=1}^{N} \Upsilon_s = V, \quad \Upsilon_i \cap \Upsilon_j = \emptyset,\ i \neq$

$\neq j$, таких, что если $v \in \Upsilon_i$ и $u \in \Upsilon_j$, при этом $(v, u) \in E$, то $i < j$, называется *параллельной формой алгоритма*. Подмножества Υ_s называются *ярусами параллельной формы*. Число N называется *высотой параллельной формы*, а количество вершин в k-м ярусе $|\Upsilon_k|$ – *шириной k-го яруса*; число $\max\limits_{1 \le k \le N} |\Upsilon_k|$ называется *шириной параллельной формы алгоритма*.

Параллельная форма минимальной высоты называется *максимальной параллельной формой*, а ее высота – *высотой алгоритма*. Одна из задач распараллеливания алгоритма заключается в том, чтобы найти параллельную форму алгоритма с как можно меньшей высотой.

Максимальная параллельная форма называется *канонической*, если в каждую вершину k-го яруса ведет хотя бы один путь длиной $k - 1$.

Свойства параллельной формы алгоритма:

1) никакие две вершины яруса не связаны дугой;

2) дуги, входящие в вершины первого яруса, не имеют начальных вершин или отсутствуют;

3) высота канонической параллельной формы алгоритма на единицу больше длины критического пути его графа.

Топологическая сортировка

Пусть $G = (V, E)$ – граф, описывающий некоторый алгоритм. Рассмотрим один из возможных способов распараллеливания алгоритма, основанный на специальной разметке вершин его графа, называемой *топологической сортировкой*. Выберем в графе любое число вершин, не имеющих предшествующих, и пометим их индексом 1 (вершины первого яруса параллельной формы). Удалим из графа помеченные вершины и инцидентные им дуги. В оставшемся графе вновь выберем любое число вершин, не имеющих предшествующих, и пометим их индексом 2 (вершины второго яруса параллельной формы). Продолжаем этот процесс до тех пор, пока не исчерпаем весь граф. Таким образом, все вершины графа будут распределены по ярусам параллельной формы так, что для любой дуги $(v_i, v_j) \in E$ вершины v_i и v_j из V, помеченные соответственно индексами i и j, будут размещены в разных ярусах, причем $i < j$.

Пример 1.1 (продолжение). Применим алгоритм топологической сортировки для рассматриваемого алгоритма при $N = 2$ (рис. 1.2).

$$V = \{\, v(i,j,k) \in \mathbf{Z}^3 \mid 1 \le i,j,k \le 2 \,\} = \bigcup_{\alpha=1}^{4} \Upsilon_\alpha,$$
$$\Upsilon_1 = \{\, (1,1,1) \,\}, \Upsilon_2 = \{\, (1,1,2), (1,2,1), (2,1,1) \,\},$$
$$\Upsilon_3 = \{\, (1,2,2), (2,1,2), (2,2,1) \,\},\ \Upsilon_4 = \{\, (2,2,2) \,\}.$$

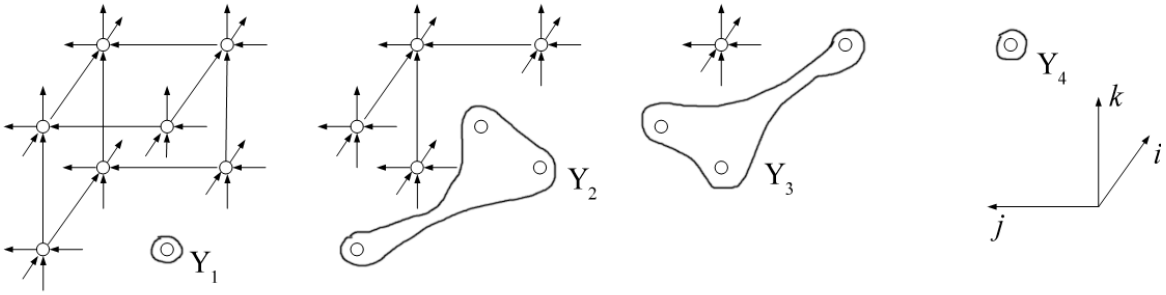

Рис. 1.2. Топологическая сортировка операций алгоритма, приводимого в примере 1.1 при $N = 2$

Распараллеливание строго направленных графов алгоритмов

Граф алгоритма будем называть *решетчатым*, если его вершины размещены в узлах d-мерной целочисленной решетки $\mathbf{Z}^d$. Вершины решетчатых графов можно описывать d-мерными векторами с целочисленными координатами (радиусами-векторами). Дуги, представляющие собой направленные прямолинейные отрезки, также можно характеризовать векторами с целочисленными координатами, которые будем называть *векторами зависимостей*. Множество всех векторов зависимостей обозначим Φ.

Пусть выделен некоторый набор из r d-мерных векторов зависимостей $\Phi = \{\, \varphi_1, \ldots, \varphi_r \,\}$, таких, что каждая дуга графа алгоритма задается одним из этих векторов: $(v,u) \in E \ \Leftrightarrow\ \exists \varphi \in \Phi$, $u = v + \varphi$. Предположим, существует такая прямая в $\mathbf{Z}^d$ с направляющим вектором $\tau \in \mathbf{Z}^d$, что проекции всех дуг на эту прямую ненулевые и одинаково направлены, т. е. $\mathrm{Пр}_\tau \varphi = |\varphi| \cos(\varphi, \tau) = \dfrac{\tau \cdot \varphi}{|\tau|} > 0$ (< 0), $\quad \forall \varphi \in \Phi$.

Граф G называется *строго направленным*, если *конус допустимых направлений графа* $K(G) = \{\, \tau \in \mathbf{Z}^d | \ \tau \cdot \varphi > 0, \quad \forall \varphi \in \Phi \,\}$ не пуст.

Прямая с направляющим вектором τ из непустого конуса допустимых направлений $K(G)$ определяет параллельную форму алгоритма. Каждый ярус параллельной формы содержит те и только те вершины, которые проецируются в одну точку на прямой. Все вершины одного яруса находятся в гиперплоскости $\tau \cdot v + t_0 = C,\ \tau \in K(G),\ v \in V$, проходящей через соответствующую проекцию. Вектор $\tau \in K(G)$ является нормальным вектором для этой гиперплоскости. Сами ярусы упорядочиваются в соответствии с упорядочиванием проекций (в соответствии с величиной ранжирующей константы C). При подходящем выборе параметра t_0 номера ярусов будут начинаться с 1.

Таким образом для строго направленных графов можно ввести понятие *ранжирующей* (*таймирующей*) *функции* $t : V \longrightarrow Z_+$:

$$t(v) = \tau \cdot v + t_0,\ \tau \in K(G),\ v \in V,\ t_0 = \text{const}.$$

К одному ярусу параллельной формы относят те вершины (операции), в которых функция t принимает одинаковое значение.

Пример 1.1 (продолжение). Для графа алгоритма $G = (V, E)$, изображенного на рис. 1.1, имеем $V = \{\, v(i,j,k) \in \mathbf{Z}^3 \mid 1 \le i,j,k \le 2 \,\}$, $E = \{\, (v_1, v_2) \in V \times V,\ v_2 = v_1 + \varphi,\ \varphi \in \Phi \,\}$, $\Phi = \{\, e_1, e_2, e_3 \,\}$, $e_1 = (1,0,0)$, $e_2 = (0,1,0)$, $e_3 = (0,0,1)$. Определим конус допустимых направлений графа G. Для этого найдем такой вектор $\tau = (\tau_1, \tau_2, \tau_3)$, что скалярное произведение вектора τ и векторов зависимости $\varphi \in \Phi$ будет положительным или в силу целочисленности обоих векторов не меньше 1. Координаты вектора τ должны удовлетворять системе неравенств:

$$\begin{cases} \tau \cdot e_1 \ge 1, \\ \tau \cdot e_2 \ge 1, \\ \tau \cdot e_3 \ge 1, \end{cases} \text{ т. е. } \begin{cases} \tau_1 \ge 1, \\ \tau_2 \ge 1, \\ \tau_3 \ge 1. \end{cases}$$

Конус допустимых направлений для нашего графа G – это пирамида с вершиной в точке $(1,1,1)$. Поскольку конус допустимых направлений не пуст, то граф G является строго направленным. Множество вершин $v(i,j,k) \in V$ графа G можно разбить на ярусы плоскостями $i + j + k - 2 = C$:

1) $\{\, (1,1,1) \,\}$;
2) $\{\, (1,1,2), (1,2,1), (2,1,1) \,\}$;
3) $\{\, (1,2,2), (2,1,2), (2,2,1) \,\}$;
4) $\{\, (2,2,2) \,\}$.

При таком разбиении вершин графа алгоритма на ярусы высота параллельной формы равна 4, а ее ширина – 3.

Для рассматриваемого примера можно выбрать другой вектор $\tau = (1,2,3) \in K(G)$. Вычислим значения таймирующей функции в вершинах графа: $t(1,1,1) = 6 + t_0,\ t(2,1,1) = 7 + t_0,\ t(1,2,1) = 8 + t_0,\ t(1,1,2) = 9 + t_0,\ t(1,2,2) = 11 + t_0,\ t(2,1,2) = 10 + t_0,\ t(2,2,1) = 9 + t_0,\ t(2,2,2) = 12 + t_0$. При таком разбиении вершин графа алгоритма на ярусы высота параллельной формы равна 7, а ее ширина — 2. Заметим, что предыдущее разбиение соответствует выбору таймирующей функции $t(v) = \tau \cdot v + t_0$, у которой $\tau = (1,1,1) \in K(G),\ t_0 = -2$.

Как видно из примера 1.1, для одного и того же алгоритма можно получить различные параллельные формы, которые отличаются друг от друга высотой и шириной. Операции, приписанные вершинам одного яруса, можно выполнить одновременно, поскольку между вершинами одного яруса нет дуг, а следовательно, нет информационной зависимости между операциями, соответствующими этим вершинам. Согласно найденным разбиениям вершин графа алгоритма на ярусы, рассматриваемый в примере 1.1 алгоритм можно выполнять на трех процессорах за четыре такта и на двух процессорах за семь тактов. Отсюда видно, что при нахождении подходящей параллельной формы алгоритма, нужно решать задачу поиска параллельной формы минимальной высоты с шириной, не большей, чем число имеющихся процессоров. Конус допустимых направлений, как правило, содержит несколько направляющих векторов, которые дают различные варианты распараллеливания.

1.3. СИСТЕМЫ РЕКУРРЕНТНЫХ УРАВНЕНИЙ

Будем предполагать, что алгоритм, для которого требуется построить параллельную форму и спроектировать спецпроцессор, задан в виде *системы рекуррентных уравнений*

$$\begin{gathered} x^{(k)}(v) = F_{\lambda}^{(k)}(x^{(1)}(v - \varphi^{(1)}), \ldots, x^{(K)}(v - \varphi^{(K)})), \\ 1 \le k \le K, \quad 1 \le \lambda \le \Lambda, \quad K, \Lambda \in \mathbf{N}, \quad v \in V_{\lambda} \subset \mathbf{Z}^d, \end{gathered} \tag{1.1}$$

где $x^{(k)},\ 1 \le k \le K$, – переменные алгоритма; v – точка d-мерной целочисленной решетки $\mathbf{Z}^d$; $F_{\lambda}^{(k)},\ 1 \le \lambda \le \Lambda, \quad 1 \le k \le K$, – функции K

переменных (не обязательно строго зависящие от всех K переменных), описывающие операции алгоритма; $\varphi^{(k)},\ 1 \le k \le K$, – d-мерные векторы с целочисленными координатами, векторы зависимостей; $V_\lambda,\ 1 \le \le \lambda \le \Lambda$, – можества точек v, которым приписаны однотипные вычисления, $V_{\lambda_1} \cap V_{\lambda_2} = \emptyset$, при $\lambda_1 \ne \lambda_2$.

Множество $V = \bigcup_{\lambda=1}^{\Lambda} V_\lambda$ будем называть *областью вычислений алгоритма* (1.1).

Если точка $v \in V$, а точка $v - \varphi^{(k)} \notin V,\ \varphi^{(k)} \in \Phi$, то это будет означать, что $x^{(k)}(v - \varphi^{(k)})$ является *начальным данным*. Если $v \in V$ и $v+\varphi \notin V$, то вершину $v \in V$ будем называть *граничной по направлению вектора* φ. Множество граничных по всем направлениям φ, $\|\varphi\|_\infty \le 1$, вершин множества V будем обозначать ∂V.

Операцией алгоритма (1.1), приписанной точке $v \in V_\lambda$, назовем правило вычисления переменной $x^{(k)}$ в этой точке. Операция алгоритма (1.1) определяется функцией $F_\lambda^{(k)}$. *Макрооперацией алгоритма* (1.1), приписанной точке $v \in V_\lambda$, назовем правило вычисления всех переменных $x^{(k)},\ 1 \le k \le K$, в этой точке, т. е. макрооперация – это множество операций $F_\lambda^{(k)},\ 1 \le k \le K$, приписанных точке $v \in V_\lambda$.

В виде (1.1) может быть представлен широкий класс алгоритмов вычислительной математики, линейной алгебры, обработки сигналов и изображений и др.

При построении $G = (V, E)$ – графа зависимостей алгоритма, заданного в виде (1.1), будем считать, что множество его вершин V совпадает с областью вычислений алгоритма. Каждой вершине $v \in V_\lambda$ соответствует вычисление, состоящее из выполнения K операций, задаваемых функциями $F_\lambda^{(k)},\ 1 \le k \le K$, т. е. каждой вершине графа алгоритма соответствует одна макрооперация. Множество дуг $E \subset V \times V$ определяется множеством пар $(v - \varphi^{(k)}, v)$ информационно зависимых вершин. Каждую дугу $(v - \varphi^{(k)}, v)$, определяющую обмен информацией между вершинами $v - \varphi^{(k)},\ v \in V$, можно характеризовать вектором $\varphi^{(k)}$. Таким образом, множество дуг графа зависимостей алгоритма (1.1) характеризуется набором векторов зависимостей $\Phi = \{\, \varphi^{(k)},\ 1 \le k \le \le K \,\}$. Вершины $v \in V_\lambda$ графа зависимостей алгоритма будем называть *вершинами типа* λ.

Далее будем предполагать, что граф $G = (V, E)$ является строго направленным. Это означает (см. п. 1.2), что существует вектор $\tau = (\tau_1, \ldots, \tau_d) \in \mathbf{Z}^d$, образующий острые углы со всеми векторами множества Φ. То есть будем предполагать, что конус допустимых направлений графа G не пуст: $K(G) = \{ \tau \in \mathbf{Z}^d \mid \tau \cdot \varphi^{(k)} > 0, \quad \forall \varphi^{(k)} \in \Phi \} \neq \emptyset$.

Областью влияния данного назовем множество точек области вычислений алгоритма, в которых это данное используется в качестве аргумента. Область влияния начального данного A будем обозначать $\Omega(A)$. Под *магистралью* будем понимать последовательность точек области вычислений алгоритма, расположенных на прямой. Для приведения алгоритмов к виду (1.1) часто используется принцип магистральной рассылки данных: данные подаются к граничным точкам области их влияния и распространяются по магистралям ко всем остальным точкам области влияния.

Пример 1.2. Рассмотрим алгоритм, заданный в виде системы рекуррентных уравнений:

$$
\begin{aligned}
&x_1(v) = x_1(v - e_1),\\
&x_2(v) = x_1(v - e_1) + x_2(v - e_2),\\
&x_3(v) = x_1(v - e_1) \cdot x_2(v - e_2) + x_3(v - e_1 + e_2 + e_3),\\
&v \in V = \{ v(i,j,k) \in \mathbf{Z}^3 \mid 1 \le i,j,k \le N \}.
\end{aligned}
$$

Имеем: $d = 3,\ K = 3,\ \Lambda = 1$, область вычислений алгоритма $V = \{ v(i,j,k) \in \mathbf{Z}^3 \mid 1 \le i,j,k \le N \}$, множество векторов зависимостей $\Phi = \{ e_1, e_2, e_1 - e_2 - e_3 \}$, где $e_1 = (1,0,0),\ e_2 = (0,1,0),\ e_1 - e_2 - e_3 = (1,-1,-1)$.

Определим конус допустимых направлений графа зависимостей алгоритма. Для этого решим систему неравенств:

$$
\left\{ \begin{array}{l} \tau \cdot e_1 > 0,\\ \tau \cdot e_2 > 0,\\ \tau \cdot (e_1 - e_2 - e_3) > 0, \end{array} \right. \Leftrightarrow \left\{ \begin{array}{l} \tau_1 > 0,\\ \tau_2 > 0,\\ \tau_1 - \tau_2 - \tau_3 > 0, \end{array} \right. \Leftrightarrow \left\{ \begin{array}{l} \tau_1 > 0,\\ \tau_2 > 0,\\ \tau_3 < \tau_1 - \tau_2. \end{array} \right.
$$

Таким образом, конус допустимых направлений графа зависимостей алгоритма $K(G) = \{ \tau \in \mathbf{Z}^3 \mid \tau_1 > 0,\ \tau_2 > 0,\ \tau_3 < \tau_1 - \tau_2 \}$ не пуст, следовательно, граф зависимостей алгоритма строго направленный.

Граф зависимостей алгоритма при $N = 2$ изображен на рис. 1.3.

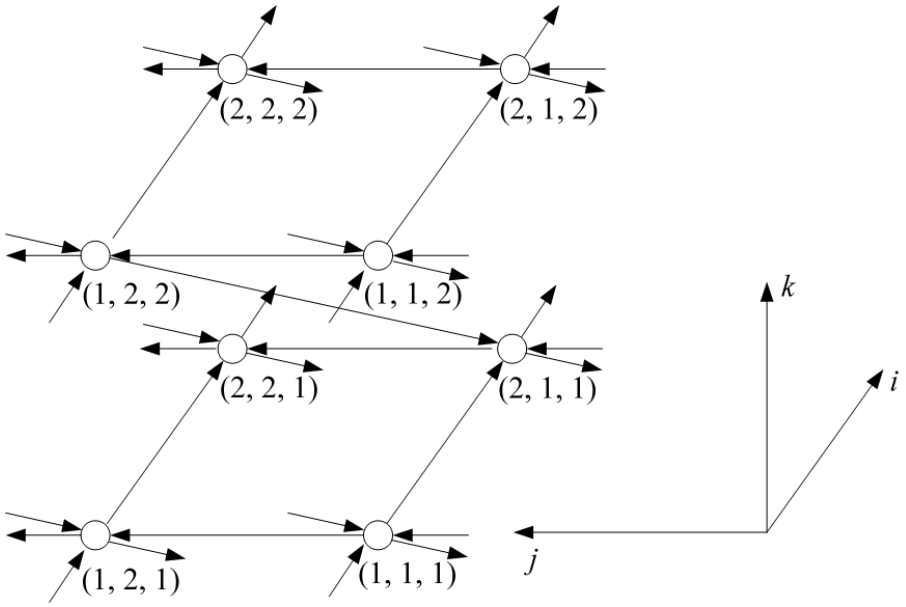

Рис. 1.3. Граф зависимостей алгоритма при $N = 2$

1.4. ПРИМЕРЫ ПОСТРОЕНИЯ АЛГОРИТМОВ В ВИДЕ СИСТЕМЫ РЕКУРРЕНТНЫХ УРАВНЕНИЙ

Приведем примеры трех типичных задач, алгоритмы решения которых можно представить системой рекуррентных уравнений вида (1.1). Обратим внимание на то, что такое представление не однозначно.

Задача о свертке

Пусть даны две последовательности: $w_1, w_2, \ldots, w_n$ *и* $x_1, x_2, \ldots, x_m$, $n \le m$. *Необходимо найти последовательность* $y_1, y_2, \ldots, y_m$, *такую, что* $y_i = \sum\limits_{k=1}^{\min(i,n)} w_k x_{i-k+1},\ i = 1, 2, \ldots, m.$

Построим алгоритм решения данной задачи в виде системы рекуррентных уравнений. Введем переменную $y(i,j) = \sum\limits_{k=1}^{j} w_k x_{i-k+1},\ 1 \le$ $\le i \le m,\ 1 \le j \le \min(i,n)$. Решением поставленной задачи являются числа $y_i = y(i, \min(i,n))$. Правило вычисления переменной $y(i,j)$ можно записать в следующем виде: $y(i,j) = w_j x_{i-j+1} + \sum\limits_{k=1}^{j-1} w_k x_{i-k+1}$. Если

положить $y(i,0)=0,\ 1\le i\le m$, то можно записать $y(i,j)=w_jx_{i-j+1}+$ $+\,y(i,j-1)$.

Таким образом, получили рекуррентное уравнение

$$y(i,j)=w_jx_{i-j+1}+y(i,j-1),$$
$$1\le i\le m,\ 1\le j\le \min(i,n).$$

Для полученного алгоритма $d=2,\ K=1,\ \Lambda=1$, область вычислений $V=\{\,(i,j)\in\mathbf{Z}^2\mid 1\le i\le m,\ 1\le j\le \min(i,n)\,\}$, множество векторов зависимостей состоит из одного вектора $\Phi=\{\,e_2\,\}$, где $e_2=$ $=(0,1)$.

Граф зависимостей алгоритма свертки последовательностей при $n=$ $=2,\ m=4$ изображен на рис. 1.4.

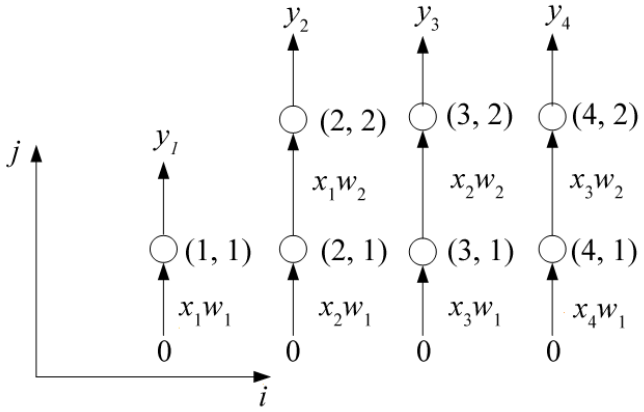

Рис. 1.4. Граф зависимостей алгоритма свертки последовательностей при $n=2,\ m=4$

Определим области влияния начальных данных алгоритма. В случае $n=2,\ m=4$:

$$\Omega(w_1)=\{\,(1,1),\,(2,1),\,(3,1),\,(4,1)\,\};$$
$$\Omega(w_2)=\{\,(2,2),\,(3,2),\,(4,2)\,\};$$
$$\Omega(x_1)=\{\,(1,1),\,(2,2)\,\};$$
$$\Omega(x_2)=\{\,(2,1),\,(3,2)\,\};$$
$$\Omega(x_3)=\{\,(3,1),\,(4,2)\,\};$$
$$\Omega(x_4)=\{\,(4,1)\,\}.$$

В общем случае:

$$\Omega(w_\alpha)=\{\,(i,\alpha)\in V\mid \alpha\le i\le m\,\},\ 1\le\alpha\le n;$$
$$\Omega(x_\alpha)=\{\,(i,j)\in V\mid i-j+1=\alpha,\ \alpha\le i\le m\,\},\ 1\le\alpha\le m.$$

Для того чтобы начальные данные поступали извне только в граничные точки области вычислений алгоритма, используем принцип магистральной рассылки данных. Для транспортировки начальных данных $w_1, w_2, \ldots, w_n$ введем переменную $w(i,j)$. Поскольку точки области влияния начального данного w_α располагаются на прямой, то возможны два алгоритма транспортировки этого начального данного:

1) $w(i,j) = w(i-1,j),\ w(\alpha-1,\alpha) = w_\alpha$;

2) $w(i,j) = w(i+1,j),\ w(m+1,\alpha) = w_\alpha$.

Выберем первый вариант. Аналогично для транспортировки начальных данных $x_1, x_2, \ldots, x_n$ введем переменную $x(i,j)$ и выберем следующий алгоритм транспортировки начального данного $x_\alpha : x(i,j) = x(i-1,j-1),\ x(\alpha-1,0) = x_\alpha$.

Таким образом, получили алгоритм, записанный в виде системы рекуррентных уравнений:

$$\begin{cases} y(i,j) = w(i-1,j)x(i-1,j-1) + y(i,j-1), \\ x(i,j) = x(i-1,j-1), \\ w(i,j) = w(i-1,j), \\ 1 \le i \le m,\ 1 \le j \le \min(i,n). \end{cases} \tag{1.2}$$

Для алгоритма (1.2): $d = 2,\ K = 3,\ \Lambda = 1$, область вычислений $V = \{\,(i,j) \in \mathbf{Z}^2 \mid 1 \le i \le m,\ 1 \le j \le \min(i,n)\,\}$, множество векторов зависимостей алгоритма $\Phi = \{\,e_2, e_1 + e_2, e_1\,\}$, где $e_2 = (0,1),\ e_1 + e_2 = (1,1),\ e_1 = (1,0)$.

Граф зависимостей алгоритма (1.2) при $n = 2,\ m = 4$ изображен на рис. 1.5.

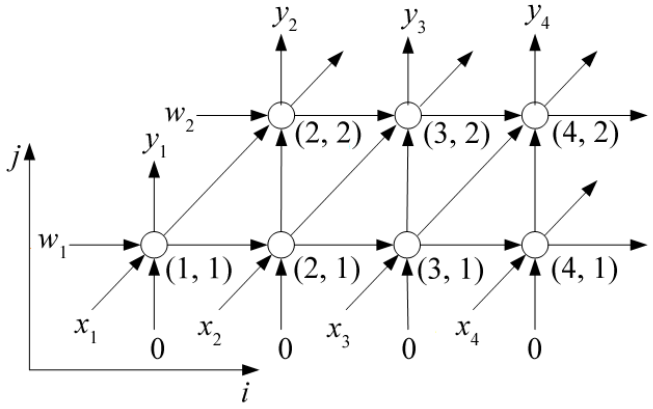

Рис. 1.5. Граф зависимостей алгоритма (1.2) при $n = 2,\ m = 4$

Определим конус допустимых направлений графа зависимостей алгоритма (1.2). Для этого решим систему неравенств:

$$\begin{cases} \tau \cdot e_1 > 0, \\ \tau \cdot e_2 > 0, \\ \tau \cdot (e_1 + e_2) > 0, \end{cases} \Leftrightarrow \begin{cases} \tau_1 > 0, \\ \tau_2 > 0, \\ \tau_1 + \tau_2 > 0. \end{cases}$$

Таким образом, конус допустимых направлений графа зависимостей алгоритма $K(G) = \{\, \tau \in \mathbf{Z}^2 \mid \tau_1 > 0,\ \tau_2 > 0 \,\}$ не пуст, следовательно, граф зависимостей алгоритма (1.2) строго направленный.

Следует заметить, что выбрав другой алгоритм транспортировки начальных данных, можно получить пустой конус допустимых направлений. Например, если выбрать следующий алгоритм транспортировки начального данного $x_\alpha : x(i,j) = x(i+1, j+1)$, $x(\min(\alpha+1, m)+1, \min(2, m-\alpha+1)+1) = x_\alpha$, то получим систему уравнений:

$$\begin{cases} \begin{cases} y(i,j) = w(i-1,j)x(i+1,j+1) + y(i,j-1), \\ x(i,j) = x(i+1,j+1), \\ w(i,j) = w(i-1,j), \end{cases} \\ 1 \le i \le m,\ 1 \le j \le \min(i,n). \end{cases} \tag{1.3}$$

В данном случае конус допустимых направлений пуст: $K(G) = \{\, \tau \in \in \mathbf{Z}^2 \mid \tau_1 > 0,\ \tau_2 > 0,\ \tau_1 + \tau_2 < 0 \,\} = \emptyset$. Граф зависимостей не является строго направленным, он содержит контуры, а система уравнений (1.3) не является алгоритмом.

Задача перемножения ленточной матрицы на вектор

Даны ленточная матрица A порядка m с шириной ленты $p+q-1$ и вектор b:

$$A = \begin{pmatrix} a_{1\,1} & \dots & a_{1\,q} & \dots & \dots & 0 \\ \vdots & & & \ddots & & \vdots \\ a_{p\,1} & & & & \ddots & \vdots \\ \vdots & \ddots & & & & a_{m-q+1\,m} \\ \vdots & & \ddots & & & \vdots \\ 0 & \dots & \dots & a_{m\,m-p+1} & \dots & a_{m\,m} \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ b_m \end{pmatrix}.$$

Необходимо найти вектор $c = A \cdot b$, *такой, что* $c_i = \sum\limits_{k=\max(1,i-p+1)}^{\min(m,i+q-1)} a_{ik}b_k$.

Введем частичную сумму $c(i,j) = \sum\limits_{k=\max(1,i-p+1)}^{j} a_{ik}b_k$, $1 \le i \le m$, $\max(1, i-p+1) \le j \le \min(m, i+q-1)$. Координатами вектора c являются числа $c_i = c(i, \min(m, i+q-1))$. Если положить $c(i, \max(0, i-p)) = 0$, то получим рекуррентное уравнение:

$$\begin{gathered} c(i,j) = a_{ij}b_j + c(i,j-1), \\ 1 \le i \le m,\ \max(1, i-p+1) \le j \le \min(m, i+q-1). \end{gathered} \tag{1.4}$$

Для алгоритма (1.4) $d = 2$, $K = 1$, $\Lambda = 1$, $V = \{\,(i,j) \in \mathbf{Z}^2 \mid 1 \le i \le m,\ \max(1, i-p+1) \le j \le \min(m, i+q-1)\,\}$, $\Phi = \{\,e_2\,\}$, где $e_2 = (0,1)$.

Области влияния начальных данных алгоритма:

$\Omega(b_\alpha) = \{\,(i,\alpha) \in V \mid \max(1, \alpha-q+1) \le i \le \min(m, \alpha+p-1)\,\}$,

$\Omega(a_{\alpha\beta}) = \{\,(\alpha,\beta) \in V\,\}$.

Для транспортировки начальных данных $b_1, \ldots, b_m$ введем переменную $b(i,j)$ и применим следующий алгоритм: $b(i,j) = b(i-1,j)$, $b(\max(0, \alpha-q), \alpha) = b_\alpha$. Областью влияния каждого начального данного a_{ij}, $1 \le i,j \le m$, является одна точка, поэтому применять алгоритм транспортировки для этих начальных данных не нужно.

Таким образом, получили алгоритм, записанный в виде системы рекуррентных уравнений:

$$\begin{gathered} \begin{cases} c(i,j) = a_{ij}b(i-1,j) + c(i,j-1), \\ b(i,j) = b(i-1,j), \end{cases} \\ 1 \le i \le m,\ \max(1, i-p+1) \le j \le \min(m, i+q-1). \end{gathered} \tag{1.5}$$

Для алгоритма (1.5): $d = 2$, $K = 2$, $\Lambda = 1$, $V = \{\,(i,j) \in \mathbf{Z}^2 \mid 1 \le i \le m,\ \max(1, i-p+1) \le j \le \min(m, i+q-1)\,\}$, $\Phi = \{\,e_2, e_1\,\}$, где $e_2 = (0,1), e_1 = (1,0)$.

Граф зависимостей алгоритма (1.5) при $m = 5$, $p = 2$, $q = 3$ изображен на рис. 1.6.

Определим конус допустимых направлений графа зависимостей алгоритма (1.5):

$$\begin{cases} \tau \cdot e_1 > 0, \\ \tau \cdot e_2 > 0, \end{cases} \Leftrightarrow \begin{cases} \tau_1 > 0, \\ \tau_2 > 0. \end{cases}$$

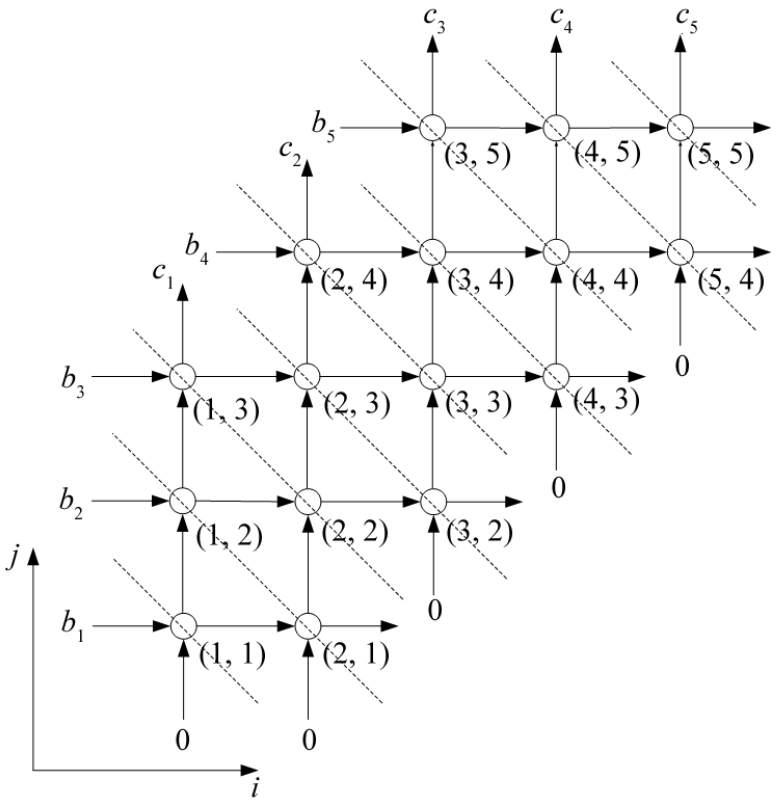

Рис. 1.6. Граф зависимостей алгоритма (1.5) при $m = 5,\ p = 2,\ q = 3$

$K(G) = \{\, \tau \in \mathbf{Z}^2 \mid \tau_1 > 0,\ \tau_2 > 0 \,\}$. Поскольку $K(G) \neq \emptyset$, то граф зависимостей алгоритма (1.5) строго направленный.

На рис. 1.6 пунктирными линиями отмечены ярусы максимальной параллельной формы, полученной с помощью таймирующей функции $t(v) = \tau \cdot v,\ v \in V,\ \tau = (1,1) \in K(G)$.

Если для транспортировки начальных данных $b_1, \ldots, b_m$ применить другой алгоритм: $b(i,j) = b(i+1,j),\ b(\min(m+1, \alpha+p), \alpha) = b_\alpha$, то придем к системе рекуррентных уравнений

$$\begin{cases} c(i,j) = a_{ij} b(i+1,j) + c(i,j-1), \\ b(i,j) = b(i+1,j), \end{cases} \qquad (1.6)$$
$$1 \le i \le m,\ \max(1, i-p+1) \le j \le \min(m, i+q-1).$$

Для алгоритма (1.6) все параметры, кроме множества векторов зависимостей $\Phi = \{e_2, -e_1\}$, такие же, как и для алгоритма (1.5).

Граф зависимостей алгоритма (1.6) при $m = 5,\ p = 2,\ q = 3$ изображен на рис. 1.7.

Определим конус допустимых направлений графа зависимостей ал-

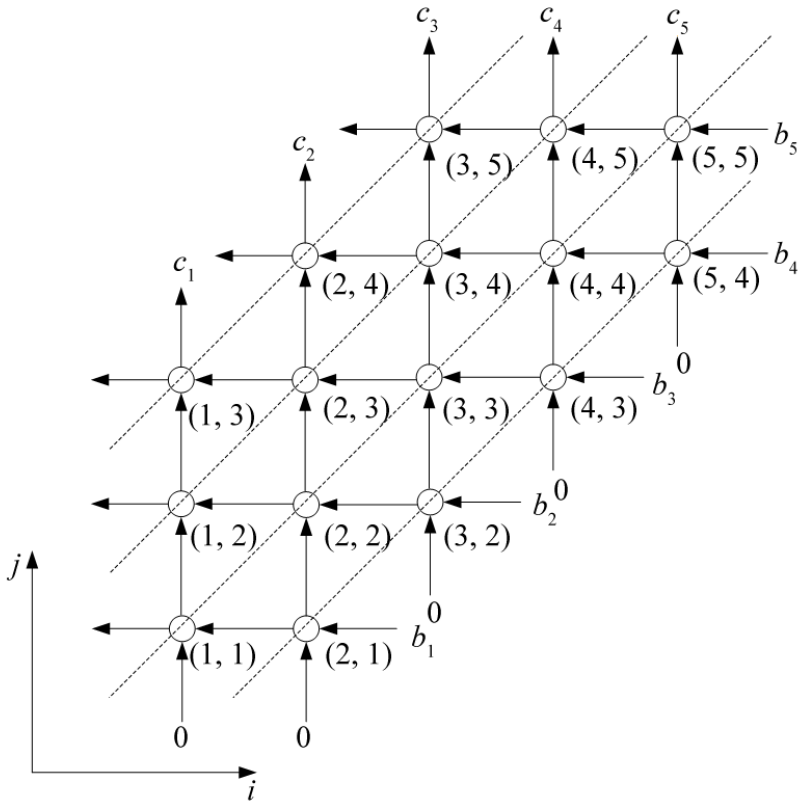

Рис. 1.7. Граф зависимостей алгоритма (1.6) при $m = 5,\ p = 2,\ q = 3$. Пунктирными линиями отмечены ярусы максимальной параллельной формы

горитма (1.6):

$$\begin{cases} \tau \cdot (-e_1) > 0, \\ \tau \cdot e_2 > 0, \end{cases} \Leftrightarrow \begin{cases} -\tau_1 > 0, \\ \tau_2 > 0. \end{cases}$$

$K(G) = \{\, \tau \in \mathbf{Z}^2 \mid \tau_1 < 0,\ \tau_2 > 0 \,\} \neq 0$. Полагая $t(v) = \tau \cdot v,\ \tau = (-1, 1) \in K(G)$, придем к максимальной параллельной форме, ярусы которой отмечены на рис. 1.7 пунктирными линиями.

Задача перемножения двух квадратных матриц

Даны две квадратные матрицы порядка N: X^1 *и* X^2. *Необходимо найти матрицу* $X^3 = X^2 \cdot X^1$.

Обозначим элементы матриц $X^1,\ X^2,\ X^3$ как $x^1_{i_1 i_2},\ x^2_{i_1 i_2},\ x^3_{i_1 i_2},\ 1 \leq \leq i_1, i_2 \leq N$, соответственно. Тогда элементы матрицы X^3 вычисляются по следующему правилу: $x^3_{i_1 i_2} = \sum\limits_{k=1}^{N} x^2_{i_1 k} x^1_{k i_2},\ 1 \leq i_1, i_2 \leq N$.

Введем частичную сумму $x^{(3)}(i_1, i_2, i_3) = \sum\limits_{k=1}^{i_3} x^2_{i_1 k} x^1_{k i_2},\ 1 \leq i_1, i_2,$

$i_3 \le N$. Решением поставленной задачи являются числа $x^3_{i_1 i_2} = x^{(3)}(i_1, i_2, N)$, $1 \le i_1, i_2 \le N$. Для определения всех значений $x^{(3)}(i_1, i_2, i_3)$ получим рекуррентное уравнение:

$$x^{(3)}(i_1, i_2, i_3) = x^2_{i_1 i_3} x^1_{i_3 i_2} + x^{(3)}(i_1, i_2, i_3 - 1),$$
$$1 \le i_1, i_2, i_3 \le N.$$

При $i_3 = 1$ полученное рекуррентное соотношение не определено. Доопределим его нулевым начальным значением $x^3(i_1, i_2, 0) = 0$ либо определим

$$x^{(3)}(i_1, i_2, i_3) = \begin{cases} x^2_{i_1 i_3} x^1_{i_3 i_2}, & i_3 = 1, \\ x^2_{i_1 i_3} x^1_{i_3 i_2} + x^{(3)}(i_1, i_2, i_3 - 1), & 1 < i_3 \le N. \end{cases} \tag{1.7}$$

Для алгоритма (1.7) $d = 3,\ K = 1,\ \Lambda = 2$;

$$V_1 = \{(i_1, i_2, i_3) \in \mathbf{Z}^3 \mid 1 \le i_1, i_2 \le N,\ i_3 = 1\};$$

$$V_2 = \{(i_1, i_2, i_3) \in \mathbf{Z}^3 \mid 1 \le i_1, i_2 \le N,\ 1 < i_3 \le N\};$$

$$V = \{\, (i_1, i_2, i_3) \in \mathbf{Z}^3 \mid 1 \le i_1, i_2, i_3 \le N \,\};$$

$$\Phi = \{\, e_3 \,\},\ e_3 = (0, 0, 1).$$

Области влияния начальных данных алгоритма:

$$\Omega(x^1_{i_3 i_2}) = \{\, (i_1, i_2, i_3) \in V \mid 1 \le i_1 \le N \,\},$$
$$\Omega(x^2_{i_1 i_3}) = \{\, (i_1, i_2, i_3) \in V \mid 1 \le i_2 \le N \,\}.$$

Для транспортировки начальных данных $x^1_{i_3 i_2}$ и $x^2_{i_1 i_3}$ введем переменные $x^{(1)}(i_1, i_2, i_3)$ и $x^{(2)}(i_1, i_2, i_3)$. Точки области влияния начального данного $x^1_{i_3 i_2}$ располагаются на прямой, следовательно, возможны два алгоритма его транспортировки:

1) $x^{(1)}(i_1, i_2, i_3) = x^{(1)}(i_1 - 1, i_2, i_3),\ x^{(1)}(0, i_2, i_3) = x^1_{i_3 i_2}$;
2) $x^{(1)}(i_1, i_2, i_3) = x^{(1)}(i_1 + 1, i_2, i_3),\ x^{(1)}(N + 1, i_2, i_3) = x^1_{i_3 i_2}$.

Выберем первый вариант. Аналогично для транспортировки начального данного $x^2_{i_1 i_3}$ возможны два алгоритма:

1) $x^{(2)}(i_1, i_2, i_3) = x^{(2)}(i_1, i_2 - 1, i_3),\ x^{(2)}(i_1, 0, i_3) = x^2_{i_1 i_3}$;
2) $x^{(2)}(i_1, i_2, i_3) = x^{(2)}(i_1, i_2 + 1, i_3),\ x^{(2)}(i_1, N + 1, i_3) = x^2_{i_1 i_3}$.

Также выберем первый вариант.

Заменим в рекуррентном соотношении (1.7) начальные данные введенными переменными и получим алгоритм, записанный в виде системы рекуррентных уравнений:

$$x^{(3)}(i_1,i_2,i_3)=\begin{cases} x^{(2)}(i_1,i_2-1,i_3)x^{(1)}(i_1-1,i_2,i_3), \\ \qquad\qquad\qquad\qquad i_3=1, \\ x^{(2)}(i_1,i_2-1,i_3)x^{(1)}(i_1-1,i_2,i_3)+ \\ \qquad +x^{(3)}(i_1,i_2,i_3-1),\ 1<i_3\le N, \end{cases} \tag{1.8}$$

$$x^{(1)}(i_1,i_2,i_3)=x^{(1)}(i_1-1,i_2,i_3),$$

$$x^{(2)}(i_1,i_2,i_3)=x^{(2)}(i_1,i_2-1,i_3),$$

$$1\le i_1,i_2,i_3\le N.$$

Для алгоритма (1.8): $d=3,\ K=3,\ \Lambda=2$, V_1, V_2, V такие же, как в алгоритме (1.7), $\Phi=\{e_1,e_2,e_3\}$, где $e_1=(1,0,0), e_2=(0,1,0), e_3=(0,0,1)$.

Граф зависимостей алгоритма (1.8) при $N=2$ изображен на рис. 1.8

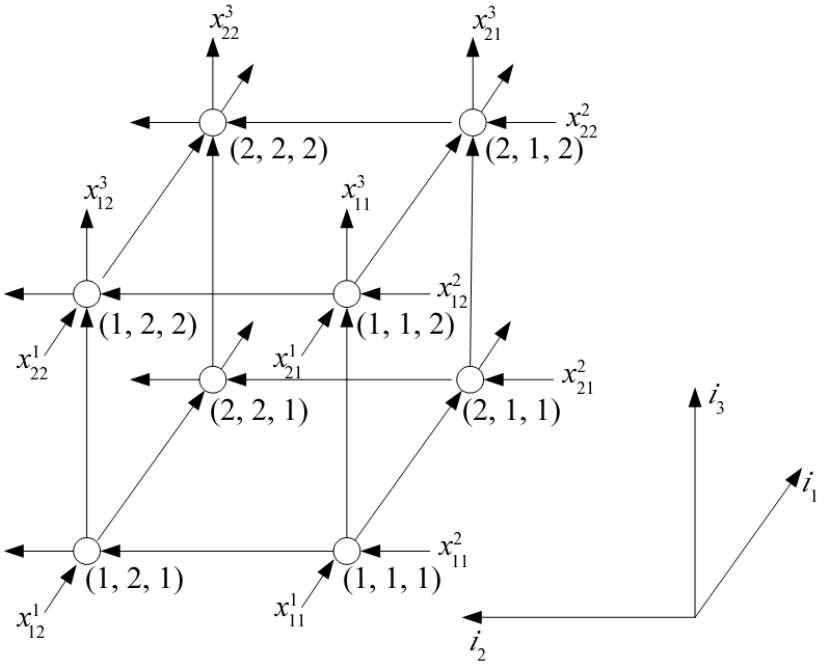

Рис. 1.8. Граф зависимостей алгоритма (1.8) при $N=2$

и содержит вершины двух типов. Область вычислений алгоритма делится на две непересекающиеся подобласти: $V=V_1\cup V_2$. Каждой вер-

шине $v \in V_1$ соответствует макрооперация «умножение и переопределения», каждой вершине $v \in V_2$ соответствует макрооперация «умножение со сложением и переопределения».

Таблица 1.1

Основные характеристики 16 алгоритмов перемножения двух квадратных матриц порядка N, представимых в виде системы рекуррентных уравнений (1.1)

№ п/п	$(\varphi^{(1)}, \varphi^{(2)}, \varphi^{(3)})$	v_{in} для $x^1_{i_3 i_2}$	v_{in} для $x^2_{i_1 i_3}$	v_{out} для $x^3_{i_1 i_2}$
$x^{(3)}(i_1,i_2,i_3) = \sum_{k=1}^{i_3} x^2_{i_1 k} x^1_{k i_2}$				
1	$(\ e_1,\ \ e_2,\ \ e_3)$	$(1, i_2, i_3)$	$(i_1, 1, i_3)$	(i_1, i_2, N)
2	$(\ e_1, -e_2,\ \ e_3)$	$(1, i_2, i_3)$	(i_1, N, i_3)	(i_1, i_2, N)
3	$(-e_1,\ \ e_2,\ \ e_3)$	(N, i_2, i_3)	$(i_1, 1, i_3)$	(i_1, i_2, N)
4	$(-e_1, -e_2,\ \ e_3)$	(N, i_2, i_3)	(i_1, N, i_3)	(i_1, i_2, N)
$x^{(3)}(i_1,i_2,i_3) = \sum_{k=N-i_3+1}^{N} x^2_{i_1 k} x^1_{k i_2}$				
5	$(\ e_1,\ \ e_2,\ \ e_3)$	$(1, i_2, N+1-i_3)$	$(i_1, 1, N+1-i_3)$	(i_1, i_2, N)
6	$(\ e_1, -e_2,\ \ e_3)$	$(1, i_2, N+1-i_3)$	$(i_1, N, N+1-i_3)$	(i_1, i_2, N)
7	$(-e_1,\ \ e_2,\ \ e_3)$	$(N, i_2, N+1-i_3)$	$(i_1, 1, N+1-i_3)$	(i_1, i_2, N)
8	$(-e_1, -e_2,\ \ e_3)$	$(N, i_2, N+1-i_3)$	$(i_1, N, N+1-i_3)$	(i_1, i_2, N)
$x^{(3)}(i_1,i_2,i_3) = \sum_{k=i_3}^{N} x^2_{i_1 k} x^1_{k i_2}$				
9	$(\ e_1,\ \ e_2, -e_3)$	$(1, i_2, i_3)$	$(i_1, 1, i_3)$	$(i_1, i_2, 1)$
10	$(\ e_1, -e_2, -e_3)$	$(1, i_2, i_3)$	(i_1, N, i_3)	$(i_1, i_2, 1)$
11	$(-e_1,\ \ e_2, -e_3)$	(N, i_2, i_3)	$(i_1, 1, i_3)$	$(i_1, i_2, 1)$
12	$(-e_1, -e_2, -e_3)$	(N, i_2, i_3)	(i_1, N, i_3)	$(i_1, i_2, 1)$
$x^{(3)}(i_1,i_2,i_3) = \sum_{k=1}^{N-i_3+1} x^2_{i_1 k} x^1_{k i_2}$				
13	$(\ e_1,\ \ e_2, -e_3)$	$(1, i_2, N+1-i_3)$	$(i_1, 1, N+1-i_3)$	$(i_1, i_2, 1)$
14	$(\ e_1, -e_2, -e_3)$	$(1, i_2, N+1-i_3)$	$(i_1, N, N+1-i_3)$	$(i_1, i_2, 1)$
15	$(-e_1,\ \ e_2, -e_3)$	$(N, i_2, N+1-i_3)$	$(i_1, 1, N+1-i_3)$	$(i_1, i_2, 1)$
16	$(-e_1, -e_2, -e_3)$	$(N, i_2, N+1-i_3)$	$(i_1, N, N+1-i_3)$	$(i_1, i_2, 1)$

Запишем алгоритм (1.8) в виде системы рекуррентных уравне-

ний (1.1):

$$\begin{aligned} x^{(3)}(v) &= \begin{cases} x^{(2)}(v-e_2)x^{(1)}(v-e_1), & v \in V_1, \\ x^{(2)}(v-e_2)x^{(1)}(v-e_1) + x^{(3)}(v-e_3), & v \in V_2, \end{cases} \\ x^{(1)}(v) &= x^{(1)}(v-e_1), \qquad v \in V_1 \cup V_2, \\ x^{(2)}(v) &= x^{(2)}(v-e_2), \qquad v \in V_1 \cup V_2. \end{aligned} \tag{1.9}$$

Конус допустимых направлений графа зависимостей алгоритма (1.8) $K(G) = \{\, \tau \in \mathbf{Z}^3 \mid \tau_1 > 0,\, \tau_2 > 0,\, \tau_3 > 0 \,\}$. Так как $K(G) \neq \emptyset$, то граф зависимостей алгоритма строго направленный.

Для задачи перемножения двух квадратных матриц таким же образом можно получить 16 различных алгоритмов, записанных в виде системы рекуррентных уравнений вида (1.1). Получить эти алгоритмы можно путем различного выбора частичной суммы $x^{(3)}(i_1, i_2, i_3)$ и алгоритмов транспортировки начальных данных $x^1_{i_3 i_2}$ и $x^2_{i_1 i_3}$.

В табл. 1.1 представлены основные характеристики 16 алгоритмов для перемножения квадратных матриц. Здесь: $\varphi^{(k)}$ – вектор зависимости, соответствующий переменной алгоритма $x^{(k)}$, $k = 1, 2, 3$; v_{in} для $x^k_{i_\alpha i_\beta}$ – вершина ввода начального данного $x^k_{i_\alpha i_\beta}$, $k = 1, 2$; v_{out} для $x^3_{i_1 i_2}$ – вершина вывода результата.