ЛИХОДЕД Н.А.

МЕТОДЫ РАСПАРАЛЛЕЛИВАНИЯ ГНЕЗД ЦИКЛОВ

Курс лекций

МИНСК
БГУ
2008

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Под распараллеливанием алгоритма понимается выявление (указание) параллельных множеств операций алгоритма или построение параллельных форм алгоритма, или организация конвейера, выполняющего операции алгоритма. Найти параллельные множества операций алгоритма — это значит разбить операции на группы, которые могут быть выполнены параллельно, независимо друг от друга. Построить параллельную форму алгоритма — это значит разбить операции алгоритма на группы, в каждой из которых операции можно выполнять параллельно, и указать очередность выполнения этих групп операций. Будем считать, что алгоритм задан последовательной программой. Основное внимание будем уделять распараллеливанию гнезд циклов, так как именно циклами часто описывается большая часть вычислений, и основной ресурс параллелизма относится к циклам.

Целевым компьютером, т. е. компьютером, на котором требуется реализовать параллельную версию алгоритма, будем считать многопроцессорный компьютер с распределенной памятью. Такие компьютеры, в частности, кластерные мультикомпьютерные вычислительные системы, являются в настоящее время наиболее популярными и доступными параллельными вычислительными системами. Отметим, что рассматриваемые ниже модели и методы, кроме связанных с распределением и обменом данными, пригодны и для многопроцессорных компьютеров с общей памятью.

Для отображения алгоритмов, задаваемых последовательными программами, на параллельные компьютеры с распределенной памятью требуется распределить операции и данные алгоритма между процессорами, а также установить порядок выполнения операций и обмена данными. При выполнении алгоритмов на таких компьютерах значительное время занимают коммуникации (передача и прием данных) между процессорами. Поэтому распределение операций и дан-

ных между процессорами следует производить таким образом, чтобы уменьшить количество и объем обменов данными; кроме того, следует оптимизировать структуру коммуникаций.

Возникает ряд задач. Выделим наиболее важные из них.

- Поиск информационных зависимостей, получение информационной структуры алгоритма. Любые процедуры распараллеливания должны опираться на точное или избыточное описание зависимых операций алгоритма.

- Обнаружение (выявление) потенциального параллелизма. Прежде чем параллелизм использовать, его нужно выявить, желательно как можно больше, чтобы знать возможности выполнения операций алгоритма одновременно. Один из основных способов обнаружения параллелизма — временное отображение (таймирование, scheduling) — получение параллельных множеств операций или построение параллельных форм алгоритма с помощью так называемых таймирующих функций.

- Выявление независимых частей программы — самый важный для компьютеров с распределенной памятью случай обнаружения параллелизма.

- Пространственно-временное отображение (space-time mapping) — распределение операций между виртуальными процессорами и задание порядка выполнения операций.

- Согласование распределения операций и данных по процессорам (alignment). Целью согласования распределения операций и данных по процессорам является минимизация числа и объема обменов данными.

- Блокинг (blocking) — отображение операций на физические процессоры. Блок вычислений — множество операций алгоритма, выполняемых на одном процессоре.

- Приватизация данных — выделение массивов, локальных для каждого процессора.

- Задание оптимального порядка выполнения операций; улучшение локальности данных — быстрое, до исчезновения из памяти с быстрым доступом, использование данных. Приватизацию и улучшение локальности данных называют еще локализацией данных.
- Тайлинг (tiling) — выбор зерна вычислений. Под зерном вычислений (или тайлом) понимается множество операций алгоритма, выполняемых атомарно: вычисления, принадлежащие одному зерну, не могут прерываться синхронизацией или обменом данными, требуемыми для выполнения этих операций.
- Минимизация числа и объема временных массивов.
- Структурирование коммуникаций, организация обмена данными.
- Генерация кода.
- Автоматизация распараллеливания.

Мы рассмотрим некоторые подходы к решению этих задач. Конечно, предварительно необходимо ввести основные термины и понятия, математический аппарат и модели параллельных вычислений. Автор старался найти компромисс между полнотой изложения материала и его доступностью для первого знакомства с методами статического, т. е. осуществляемого до начала выполнения программы, распараллеливания. В целях технического упрощения изложения рассматриваются гнезда циклов не самой общей структуры вложенности, в формальный аппарат не введены в явном виде внешние переменные циклов, алгоритмы решения задач параллельных вычислений носят часто упрощенный характер. Не ставилась задача практического получения параллельных версий последовательных программ, изучения способов распараллеливания больших (сотни операторов) программ.

Вместе с тем изложение достаточно полное и строгое для того чтобы ввести читателя в проблематику статического распараллеливания, основанного на знании тонкой информационной структуры программ, дать теоретические основы распараллеливания алгоритмов и преобразования последовательных программ в параллельные.

Глава 1

МАТЕМАТИЧЕСКИЙ АППАРАТ И МОДЕЛИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

В этой главе рассматривается формальный аппарат, используемый в методах статического распараллеливания программ.

1.1 Основные определения

1.1.1 Функции, задающие индексы массивов данных

Будем рассматривать аффинные гнезда циклов; в этом случае выражения индексов элементов массивов и границы изменения параметров циклов являются аффинными функциями от параметров циклов и внешних переменных; кроме операторов присваивания допускается использование условного или безусловного перехода. Пусть в гнезде циклов имеется K операторов и используется L массивов данных. Простые переменные будем считать массивами размерности 0. Область изменения параметров гнезда циклов для оператора S_β будем называть областью итераций и обозначать V_β.

Индексы элементов l-го массива данных, встречающегося в операторе S_β и относящегося к q-му вхождению элементов этого массива в оператор, будем выражать аффинной функцией

$$\overline{F}_{l,\beta,q}(J) = F_{l,\beta,q}J + f^{(l,\beta,q)}, \quad J \in V_\beta,$$

где $F_{l,\beta,q} \in \mathbf{Z}^{\nu_l \times n_\beta}$, ν_l — размерность l-го массива данных, n_β — число циклов, окружающих оператор S_β, $f^{(l,\beta,q)} \in \mathbf{Z}^{\nu_l}$. Если l-й массив имеет размерность 0 (т. е. является простой переменной), то будем считать $\overline{F}_{l,\beta,q}(J)$ константой. Область изменения индексов l-го массива будем обозначать W_l $(W_l \subset \mathbf{Z}^{\nu_l})$.

Пример 1. Рассмотрим следующий цикл:

$$\begin{array}{l} \text{do}\ \ i = 1, N \\ \quad S_1:\ c(i) = a + 3 \\ \quad S_2:\ a = i + 1 \\ \quad S_3:\ b(i) = c(i) + a \\ \text{enddo} \end{array} \tag{1.1}$$

Цикл содержит три оператора $S_1,\ S_2,\ S_3$ и три массива данных $a,\ b,\ c$; $n_1 = n_2 = n_3 = 1,\ \nu_1 = 0,\ \nu_2 = \nu_3 = 1,\ V_1 = V_2 = V_3 = V = \{(i) \in \mathbf{Z} |\ 1 \leqslant i \leqslant N\},\ W_2 = W_3 = V,\ \overline{F}_{1,1,1}(i) = \overline{F}_{1,2,1}(i) = \overline{F}_{1,3,1}(i) = 0,\ \overline{F}_{2,3,1}(i) = \overline{F}_{3,1,1}(i) = \overline{F}_{3,3,1}(i) = i,\ i \in V.$ □

Пример 2. Рассмотрим задачу умножения матрицы A порядка N на N-мерный вектор b. Пусть a_{ij} — элементы матрицы $A,\ b_j$ — элементы вектора b, $c_i = \sum\limits_{j=1}^{N} a_{ij} b_j$, — элементы N-мерного вектора c, равного Ab. Алгоритм вычисления элементов вектора c можно записать в виде двумерного гнезда циклов

$$\begin{array}{l} \text{do}\ \ i = 1, N \\ \quad S_1:\ c(i) = 0 \\ \quad \text{do}\ \ j = 1, N \\ \quad\quad S_2:\ c(i) = c(i) + a(i,j) b(j) \\ \quad \text{enddo} \\ \text{enddo} \end{array} \tag{1.2}$$

В гнезде циклов два оператора $S_1,\ S_2$ и три массива данных $a,\ b,\ c$; $n_1 = 1,\ n_2 = 2,\ \nu_1 = 2,\ \nu_2 = \nu_3 = 1,\ V_1 = \{(i) \in \mathbf{Z} |\ 1 \leqslant i \leqslant N\},\ V_2 = \{(i,j) \in \mathbf{Z}^2 |\ 1 \leqslant i, j \leqslant N\},\ W_1 = V_2,\ W_2 = \{(j) \in \mathbf{Z} |\ 1 \leqslant j \leqslant N\},\ W_3 = V_1,\ \overline{F}_{1,2,1}(i,j) = E^{(2)}(i\ \ j)^{\mathrm{T}}$, где $E^{(2)}$ — единичная матрица второго порядка, $\overline{F}_{2,2,1}(i,j) = (0\ \ 1)(i\ \ j)^{\mathrm{T}},\ (i,j) \in V_2,\ \overline{F}_{3,1,1}(i) = i,\ i \in V_1,\ \overline{F}_{3,2,1}(i,j) = \overline{F}_{3,2,2}(i,j) = (1\ \ 0)(i\ \ j)^{\mathrm{T}},\ (i,j) \in V_2.$ □

1.1.2 Зависимости

Определение (операция, итерация). *Реализация (срабатывание, выполнение) оператора S_β при конкретных значениях вектора параметров цикла J называется операцией*[1] *и обозначается $S_\beta(J)$. Вы-*

[1]Операцию $S_\beta(J)$ иногда называют итерацией $S_\beta(J)$.

полнение всех операций, зависящих от J, называется J-й итерацией (итерацией J).

Определение (зависимость между операциями). *Операция $S_\beta(J)$, $J \in V_\beta$, зависит от операции $S_\alpha(I)$, $I \in V_\alpha$, (обозначение: $S_\alpha(I) \to S_\beta(J)$) если:*

1) $S_\alpha(I)$ выполняется раньше $S_\beta(J)$;

2) $S_\alpha(I)$ и $S_\beta(J)$ используют один и тот же элемент какого-либо массива (т. е. $\overline{F}_{l,\alpha,p}(I) = \overline{F}_{l,\beta,q}(J)$ для некоторых l, p, q), и, по крайней, мере одно из использований есть переопределение (изменение) элемента;

3) между операциями $S_\alpha(I)$ и $S_\beta(J)$ этот элемент не переопределяется.

Наличие зависимости $S_\alpha(I) \to S_\beta(J)$ означает, что нельзя (без анализа функционального содержания операторов[2]) переупорядочить операции гнезда циклов таким образом, чтобы $S_\beta(J)$ выполнялась раньше $S_\alpha(I)$.

Отметим, что между операциями $S_\alpha(I)$ и $S_\beta(J)$ может быть более одной зависимости, если равенство $\overline{F}_{l,\alpha,p}(I) = \overline{F}_{l,\beta,q}(J)$ выполняется более чем для одного набора l, p, q. В этом случае можно использовать следующие обозначения зависимости между операциями: $S_\alpha(I) \overset{a_l}{\to} S_\beta(J)$, $S_\alpha(I) \overset{l,p,q}{\to} S_\beta(J)$.

Определение (лексикографический порядок). *Пусть имеются d_1-мерный вектор $(x_1, \cdots, x_{d_1})$ и d_2-мерный вектор $(y_1, \cdots, y_{d_2})$; обозначим $d = \min(d_1, d_2)$. Вектор y лексикографически больше вектора x (обозначение: $y >_{lex} x$), если первая ненулевая координата вектора $(y_1 - x_1, \cdots, y_d - x_d)$ положительна. Вектор y лексикографически равен вектору x ($y =_{lex} x$), если $(y_1 - x_1, \cdots, y_d - x_d) = 0$. Вектор y лексикографически больше или равен вектору x ($y \geqslant_{lex} x$), если $y >_{lex} x$ или $y =_{lex} x$.*

Пусть имеется гнездо вложенных циклов. Зависимость между операциями $S_\alpha(I)$ и $S_\beta(J)$ называется внутриитерационной (не вызванной циклом, loop independent dependence), если $I =_{lex} J$; если же $I <_{lex} J$, то имеет место зависимость по циклу (loop carry dependence). Таким образом, в случае внутриитерационной зависимости $S_\alpha(I) \to S_\beta(J)$ значения параметров всех общих для операторов S_α и $S_\beta(J)$ циклов

[2]Можно, например, переставить стоящие рядом операторы $m = m + 1$ и $m = m + 2$.

не отличаются; в случае зависимости по циклу значения параметра фиксированного цикла отличаются, а значения параметров всех более внешних циклов (если они имеются) не отличаются.

Пример 1 (продолжение). Найдем зависимости между операциями алгоритма (1.1). Рассмотрим первые три итерации цикла:

$S_1(1):\ c(1) = a + 3$
$S_2(1):\ a = 2$
$S_3(1):\ b(1) = c(1) + a$

$S_1(2):\ c(2) = a + 3$
$S_2(2):\ a = 3$
$S_3(2):\ b(2) = c(2) + a$

$S_1(3):\ c(3) = a + 3$
$S_2(3):\ a = 4$
$S_3(3):\ b(3) = c(3) + a$

Используя определение зависимостей сделаем вывод: массив a порождает зависимости $S_1(i) \to S_2(i),\ S_2(i) \to S_3(i),\ 1 \leqslant i \leqslant N,\ \ S_2(i-1) \to S_1(i),\ S_3(i-1) \to S_2(i),\ S_2(i-1) \to S_2(i),\ 2 \leqslant i \leqslant N$, массив b зависимостей не порождает, массив c порождает зависимость $S_1(i) \to S_3(i),\ 1 \leqslant i \leqslant N$. Три зависимости являются внутриитерационными и три — зависимостями по циклу. □

Пример 2 (продолжение). Рассмотрим итерации $i = 1,\ i = 2,\ i = 3$:

$S_1(1):\ \ c(1) = 0$
$S_2(1,1):\ c(1) = c(1) + a(1,1)b(1)$
$S_2(1,2):\ c(1) = c(1) + a(1,2)b(2)$
. . . . . .
$S_2(1,N):\ c(1) = c(1) + a(1,N)b(N)$

$S_1(2):\ \ c(2) = 0$
$S_2(2,1):\ c(2) = c(2) + a(2,1)b(1)$
$S_2(2,2):\ c(2) = c(2) + a(2,2)b(2)$
. . . . . .
$S_2(2,N):\ c(2) = c(2) + a(2,N)b(N)$

$S_1(3):\ \ c(3) = 0$
$S_2(3,1):\ c(3) = c(3) + a(3,1)b(1)$
$S_2(3,2):\ c(3) = c(3) + a(3,2)b(2)$
. . . . . .
$S_2(3,N):\ c(3) = c(3) + a(3,N)b(N)$

Анализ итераций позволяет сделать вывод, что имеются зависимости $S_1(i) \to S_2(i,1)$, $1 \leqslant i \leqslant N$, $S_2(i,j-1) \to S_2(i,j)$, $1 \leqslant i \leqslant N$, $2 \leqslant j \leqslant N$. Вторая зависимость является зависимостью по циклу с параметром j. $\square$

1.1.3 Представление зависимостей

Определение (развернутый граф зависимостей, extended dependence graph). *Развернутым графом зависимостей алгоритма называется ориентированный граф, полученный следующим образом: каждая операция $S_\beta(J)$ порождает вершину $v_\beta(J)$ графа, каждая зависимость $S_\alpha(I) \to S_\beta(J)$ порождает дугу $(v_\alpha(I), v_\beta(J))$.*

Развернутый граф зависимостей дает полную количественную информацию о зависимостях алгоритма.

Пусть

$$P = \{(\alpha,\beta) |\ \exists\ I \in V_\alpha,\ J \in V_\beta,\ \ S_\alpha(I) \to S_\beta(J)\}$$

— множество таких пар (α,β), что для каких-либо I и J существует дуга $(v_\alpha(I), v_\beta(J))$ в развернутом графе зависимостей. Множество P определяет пары зависимых операторов. Если между операторами $S_\alpha(I)$ и $S_\beta(J)$ существует более одной зависимости, то будем рассматривать пары $(\alpha,\beta)_1$, $(\alpha,\beta)_2$, $\ldots$ Пару зависимостей, порождаемую равенством $\overline{F}_{l,\alpha,p}(I) = \overline{F}_{l,\beta,q}(J)$, будем также обозначать $(\alpha,\beta)_{l,p,q}$.

Для каждой пары $(\alpha,\beta) \in P$ обозначим через $V_{\alpha,\beta}$ подмножество таких $J \in V_\beta$, что для какого-либо I существует дуга $(v_\alpha(I), v_\beta(J))$ в развернутом графе зависимостей:

$$V_{\alpha,\beta} = \{J \in V_\beta |\ \exists\ \ S_\alpha(I) \to S_\beta(J)\}.$$

Определение (функция зависимостей). *Функцию $\overline{\Phi}_{\alpha,\beta} : V_{\alpha,\beta} \to V_\alpha$ такую, что если $S_\alpha(I) \to S_\beta(J)$, $I \in V_\alpha$, $J \in V_{\alpha,\beta} \subseteq V_\beta$, то $I = \overline{\Phi}_{\alpha,\beta}(J)$, будем называть функцией зависимостей.*

Функции зависимостей описывают зависимости алгоритма: зная $\overline{\Phi}_{\alpha,\beta}$ и $V_{\alpha,\beta}$ можно для любой операции $S_\beta(J)$ найти все операции $S_\alpha(I)$, от которых $S_\beta(J)$ зависит.

В дальнейшем предполагается, что функции зависимостей являются аффинными:

$$\begin{gathered}\overline{\Phi}_{\alpha,\beta}(J) = \Phi_{\alpha,\beta} J - \varphi^{(\alpha,\beta)},\quad J \in V_{\alpha,\beta},\quad (\alpha,\beta) \in P,\\ \Phi_{\alpha,\beta} \in \mathbf{Z}^{n_\alpha \times n_\beta},\ \varphi^{(\alpha,\beta)} \in \mathbf{Z}^{n_\alpha}.\end{gathered} \tag{1.3}$$

В этом случае зависимости называются аффинными.

Пусть зависимость $S_\alpha(I) \to S_\beta(J)$ порождается элементом массива a_l (так что имеет место равенство $\overline{F}_{l,\alpha,p}(I) = \overline{F}_{l,\beta,q}(J)$ для некоторых p, q), и существует матрица $F^{-1}_{l,\alpha,p}$. Тогда $F_{l,\alpha,p}I + f^{(l,\alpha,p)} = F_{l,\beta,q}J + f^{(l,\beta,q)}$, $I = F^{-1}_{l,\alpha,p}(F_{l,\beta,q}J + f^{(l,\beta,q)} - f^{(l,\alpha,q)})$. Таким образом, функции зависимостей описываются функциями вида $\overline{\Phi}_{\alpha,\beta}(J) = F^{-1}_{l,\alpha,p}F_{l,\beta,q}J + F^{-1}_{l,\alpha,p}(f^{(l,\beta,q)} - f^{(l,\alpha,p)})$, если $F_{l,\alpha,p}$ является невырожденной квадратной матрицей.

Заметим, что во многих случаях получение функций зависимостей является трудной задачей и требует применения специальных методов.

Определение (вектор зависимости). *Пусть $(\alpha, \beta) \in P$, I, J — векторы одинаковой размерности такие, что существует зависимость $S_\alpha(I) \to S_\beta(J)$. Вектор $d^{(\alpha,\beta)} = J - I$ называется вектором зависимости.*

Если при фиксированных α и β векторы зависимостей $J - I$ одинаковы для всех I, J, то зависимость называется однородной. В этом случае $\Phi_{\alpha,\beta}$ — единичная матрица, $J - I = \varphi^{(\alpha,\beta)}$. Если для всех $(\alpha, \beta) \in P$ векторы зависимостей $J - I$ являются однородными, то гнездо циклов называется однородным, а соответствующий алгоритм — алгоритмом с однородными зависимостями.

Отметим, что в случае гнезда вложенных циклов любой вектор зависимостей $J - I$ является лексикографически неотрицательным, т. е. первая отличная от нуля координата вектора есть число положительное.

Определение (уровень зависимости). *Пусть существует зависимость $S_\alpha(I) \to S_\beta(J)$. Номер l первой ненулевой компоненты вектора зависимостей $d^{(\alpha,\beta)} = J - I$ называется уровнем зависимости. Если $J - I = 0$, то принимается $l = \infty$.*

В случае гнезда вложенных циклов уровень зависимости указывает цикл, по которому имеется зависимость. Для внутриитерационной зависимости уровень зависимости принимается бесконечно большим.

Определение (редуцированный граф зависимостей, reduced dependence graph). *Редуцированным графом зависимостей алгоритма называется ориентированный граф, полученный следующим образом: каждому оператору S_β ставится в соответствие вершина v_β, существует дуга (v_α, v_β), если для каких-либо I, J существует зави-*

симость $S_\alpha(I) \to S_\beta(J)$ (т. е. существует дуга $(v_\alpha(I), v_\beta(J))$ в развернутом графе зависимостей). Дуги могут быть помечены функциями зависимостей, векторами зависимостей, уровнями зависимостей или другими величинами, характеризующими зависимости алгоритма.

Редуцированный граф зависимостей дает сжатое представление развернутого графа зависимостей.

Пример 1 (продолжение). Все полученные ранее зависимости алгоритма (1.1) являются однородными и характеризуются одномерными векторами $\varphi^{(1,2)} = \varphi^{(2,3)} = \varphi^{(1,3)} = (0)$, $\varphi^{(2,1)} = \varphi^{(3,2)} = \varphi^{(2,2)} = (1)$. Развернутый ($N = 4$) и редуцированный графы зависимостей изображены на рис. 1.1.

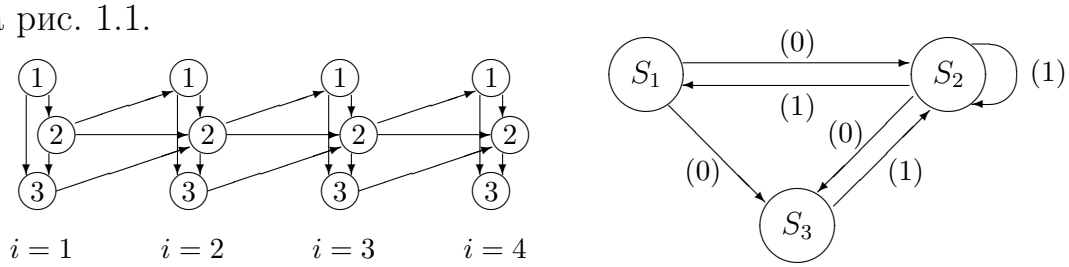

Рис. 1.1

На рис. 1.1 вершины развернутого графа помечены только номером оператора, соответствующего операции. Вершины можно пометить обозначением операции (рис. 1.2). □

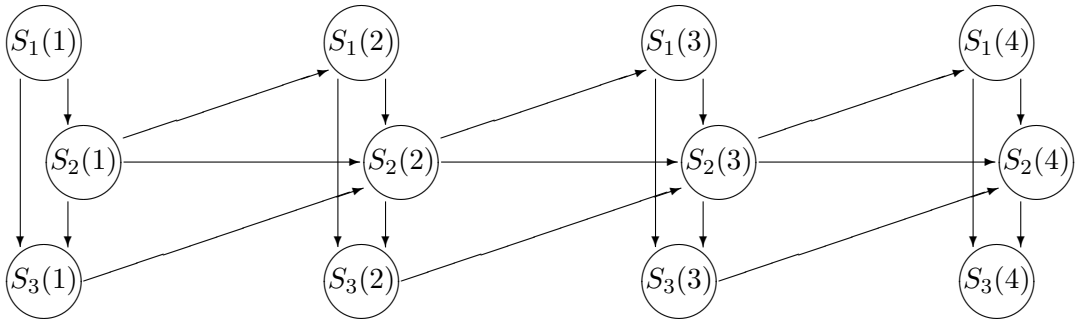

Рис. 1.2

Пример 2 (продолжение). Зависимости между операциями алгоритма (1.2) можно описать функциями зависимостей $\overline{\Phi}_{1,2}(i,1) = i$ ($\Phi_{1,2} = (1\ 0)$, $\varphi^{(1,2)} = 0$), $(i,1) \in V_{1,2} = \{(i,1) \in \mathbf{Z}^2 |\ 1 \leqslant i \leqslant N\}$,

$\overline{\Phi}_{2,2}(i,j) = (i,j-1)$ $(\Phi_{2,2} = E^{(2)},\ \varphi^{(2,2)} = (0,1))$, $(i,j) \in V_{2,2} = \{(i,j) \in \mathbf{Z}^2 |\ 1 \leqslant i \leqslant N,\ 2 \leqslant j \leqslant N\}$. Развернутый $(N = 3)$ и редуцированный графы зависимостей изображены на рис. 1.3. $\square$

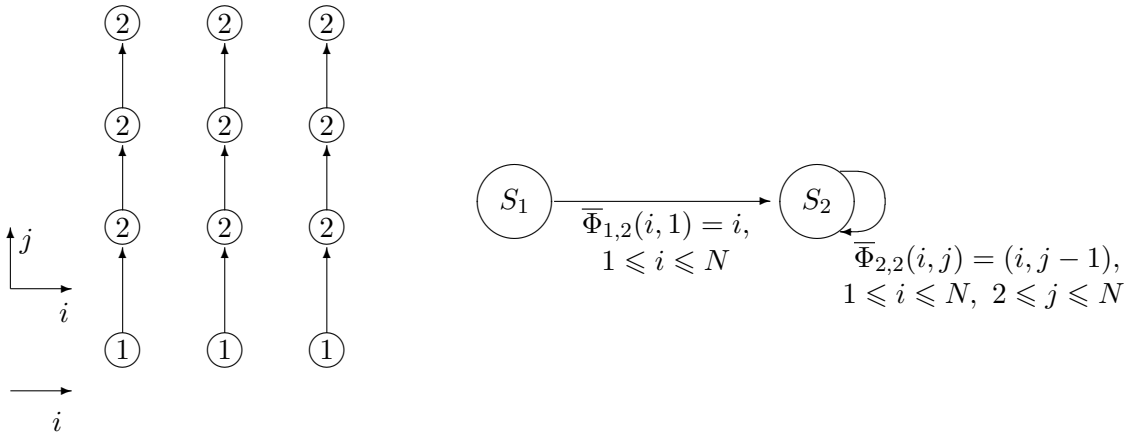

Рис. 1.3

1.2 Типы зависимостей. Устранение антизависимостей и зависимостей по выходу

В разделе 1.1 было дано определение зависимости между операциями алгоритма. Различают зависимости нескольких типов.

Определение (истинная зависимость, антизависимость, зависимость по выходу). *Зависимость называется: истинной (flow dependence), если элемент массива сначала определяется, а затем его значение используется в качестве аргумента; антизависимостью (anti dependence), если значение элемента массива сначала используется как аргумент, а затем переопределяется; зависимостью по выходу (output dependence), если элемент массива определяется, а затем переопределяется.*

Если функция $\overline{F}_{l,\alpha,p}(I)$ (см. определение зависимости в предыдущем разделе) встречается в левой части оператора S_α, а $\overline{F}_{l,\beta,q}(J)$ встречается в правой части оператора S_β, то зависимость является истинной. Если $\overline{F}_{l,\alpha,p}(I)$ встречается в правой части S_α, а $\overline{F}_{l,\beta,q}(J)$ — в левой части S_β, то имеет место антизависимость. Если обе функции $\overline{F}_{l,\alpha,p}(I)$ и

$\overline{F}_{l,\beta,q}(J)$ встречаются в левых частях операторов, то зависимость является зависимостью по выходу.

Иногда рассматривают зависимость по входу (input dependence): 1) $S_\alpha(I)$ выполняется раньше $S_\beta(J)$; 2) $S_\alpha(I)$ и $S_\beta(J)$ используют как аргумент один и тот же элемент какого-либо массива (т. е. $\overline{F}_{l,\alpha,p}(I) = \overline{F}_{l,\beta,q}(J)$ для некоторых l, p, q, причем обе функции $\overline{F}_{l,\alpha,p}(I)$ и $\overline{F}_{l,\beta,q}(J)$ встречаются в правых частях операторов); 3) между операциями $S_\alpha(I)$ и $S_\beta(J)$ этот элемент не используется. Зависимости по входу не обязательно учитывать при переупорядочении операций алгоритма, но их надо иметь в виду при распределении данных между процессорами и при улучшении локальности гнезда циклов.

Антизависимости и зависимости по выходу являются ложными, искусственными (содержание общей ячейки памяти, используемой зависимыми операторами, при использовании операторами вообще говоря разное). Они обусловлены экономным хранением результатов вычислений и могут быть исключены за счет использования большей памяти. Устранение ложных зависимостей может оказаться очень полезным при распараллеливании алгоритма, так как чем меньше зависимостей между операциями, тем больше потенциальной возможности выполнять операции параллельно.

Рассмотрим некоторые способы устранения ложных зависимостей.

1.2.1 Переобозначение

Пусть элемент массива (или переменная, которая, напомним, считается массивом размерности 0) локально используется несколько раз в разных операторах. Если этому элементу дать в разных частях программы разные имена, то ложные зависимости исчезнут.

do $i = 1, N$
  $S_1:\ a(i) = \sin i$
  $S_2:\ b(i+1) = a(i) + c(i)$
  $S_3:\ a(i) = \cos i$
  $S_4:\ c(i+1) = a(i)$
enddo

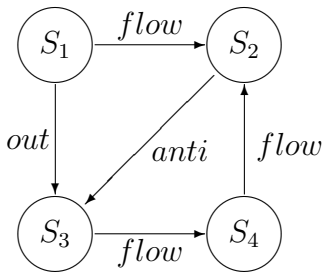

Рис. 1.4

На рис. 1.4 изображен цикл и соответствующий ему редуцированный граф зависимостей, на дугах которого указан тип зависимостей. Если в операторах S_1 и S_2 элемент $a(i)$ переобозначить, то ложные зависимости исчезнут (рис. 1.5).

do $i = 1, N$
  $S_1:\ t(i) = \sin i$
  $S_2:\ b(i+1) = t(i) + c(i)$
  $S_3:\ a(i) = \cos i$
  $S_4:\ c(i+1) = a(i)$
enddo

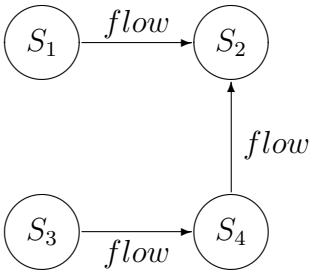

Рис. 1.5

1.2.2 Увеличение размерности

Пусть элемент массива определяется более, чем на одной итерации. Тогда существуют зависимости по выходу. Эти зависимости можно устранить, если вместо данного массива использовать массив большей размерности. В качестве примера рассмотрим цикл (1.1) из раздела 1.1 (рис. 1.6).

do $i = 1, N$
  $S_1:\ c(i) = a + 3$
  $S_2:\ a = i + 1$
  $S_3:\ b(i) = c(i) + a$
enddo

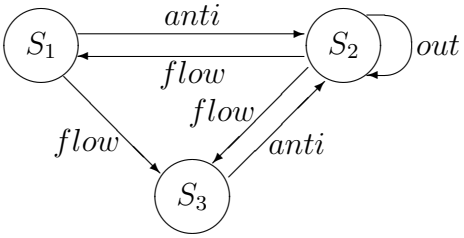

Рис. 1.6

Если вместо переменной a использовать одномерный массив, то ложные зависимости исчезнут (рис. 1.7).

1.2.3 Расщепление операторов

Расщепление операторов применяется для устранения циклов с ложными зависимостями в редуцированных графах зависимостей. Расщепление операторов является полезным преобразованием для распараллеливания гнезд циклов.

$t(0) = a$
do $i = 1, N$
$S_1 : \ c(i) = t(i-1) + 3$
$S_2 : \ t(i) = i + 1$
$S_3 : \ b(i) = c(i) + t(i)$
enddo
$a = t(N)$

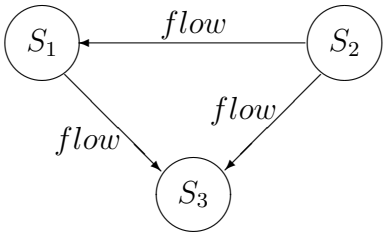

Рис. 1.7

Приведем сначала пример. Рассмотрим цикл и его редуцированный граф зависимостей (рис. 1.8). Разобьем оператор S_1 на два оператора (рис. 1.9). Смысл этого преобразования в том, что исчезает цикл в редуцированном графе зависимостей. Теперь все операции $S_1'(i)$ могут быть выполнены раньше любой операции $S_2(i)$ и раньше любой операции $S_1(i)$, а все операции $S_2(i)$ могут быть выполнены раньше любой операции $S_1(i)$. Отметим, что применение приемов переобозначения и увеличения размерности также приводило к устранению циклов в редуцированном графе зависимостей.

do $i = 1, N$
$S_1 : \ a(i) = b(i) + c(i)$
$S_2 : \ a(i+1) = a(i) + 2d(i)$
enddo

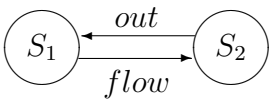

Рис. 1.8

do $i = 1, N$
$S_1' : \ t(i) = b(i) + c(i)$
$S_1 : \ a(i) = t(i)$
$S_2 : \ a(i+1) = t(i) + 2d(i)$
enddo

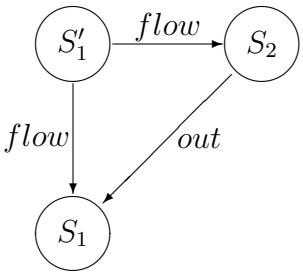

Рис. 1.9

Изобразим развернутые графы зависимостей исходного и преобразованного алгоритмов (рис. 1.10, 1.11). Нетрудно заметить, что преобразованный алгоритм можно распараллелить.

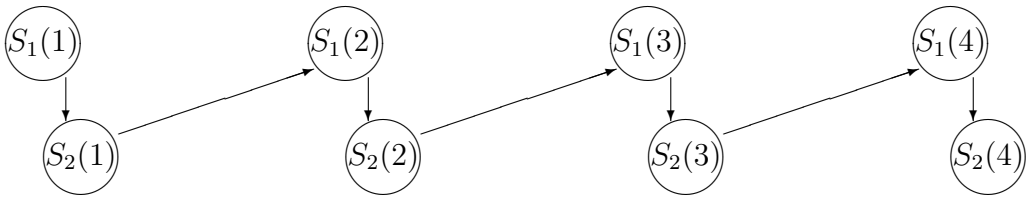

Рис. 1.10

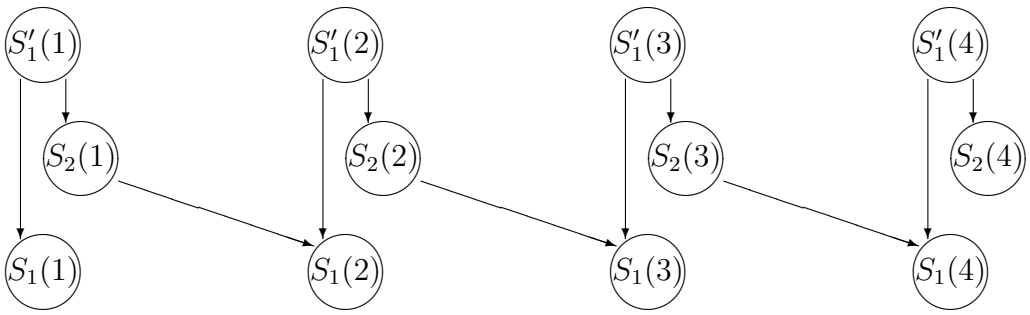

Рис. 1.11

В общем случае способ расщепления операторов заключается в следующем. Оператор

$$S: \ a(f(J)) = \ldots$$

заменяется двумя операторами:

$$S': \ t(f(J)) = \ldots$$
$$S: \ a(f(J)) = t(f(J))$$

Если в правой части какого-либо оператора встречается элемент массива a, встречающийся также в левой части оператора S, то он заменяется на соответствующий элемент массива t.

При замене в редуцированном графе зависимостей вершины S на вершины S' и S : входящие до преобразования в S ложные зависимости после преобразования снова входят в S, входящие в S истинные зависимости входят в S'; выходящие до преобразования из S истинные и антизависимости выходят после преобразования из S', выходящие из S зависимости по выходу выходят снова из S. Изобразим схематично все возможные случаи, в которых среди входящих или выходящих зависимостей есть ложные зависимости (рис. 1.12).

Из приведенных схем видно, что циклы в редуцированном графе зависимостей исчезают в первых четырех случаях и не исчезают в остальных случаях; заметим, что в случае, когда и входящая и выхо-

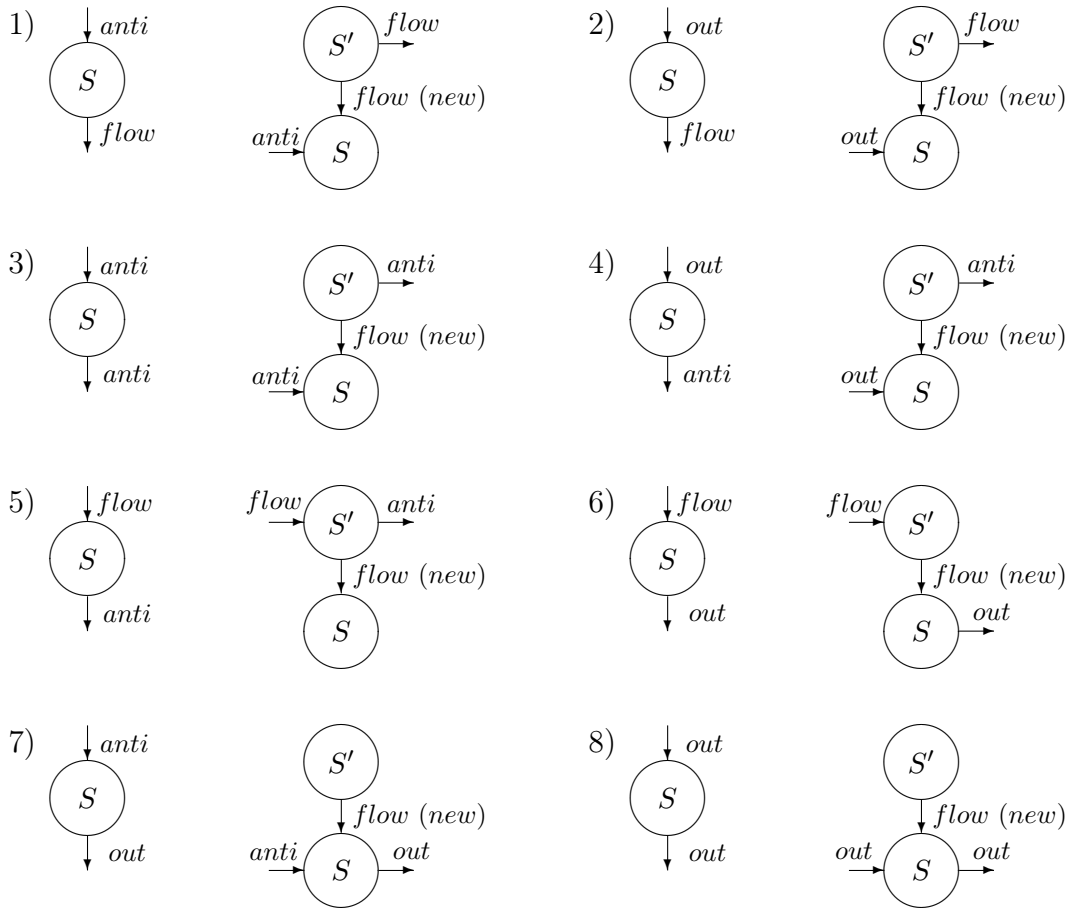

Рис. 1.12

дящая зависимости являются истинными, циклы также не исчезают. Поэтому расщепление операторов применяется в тех случаях, когда входная дуга определяет любую ложную зависимость, а выходная дуга определяет истинную или антизависимость.

Отметим, что не следует стремиться исключать все ложные зависимости, поскольку при каждом исключении требуется дополнительная память для хранения дополнительных массивов. Имеет смысл устранять только такие ложные зависимости, которые препятствуют распараллеливанию алгоритма.

Отметим еще, что преобразования кода могут устранить не только выходные, анти-, но и истинные зависимости. Таким преобразованием является, например, подстановка переменной. Осуществим в цикле

(1.1) подстановку переменной a в третий оператор:

$$\begin{array}{l}\text{do}\ \ i = 1, N\\ \quad S_1:\ c(i) = a + 3\\ \quad S_2:\ a = i + 1\\ \quad S_3:\ b(i) = c(i) + i + 1\\ \text{enddo}\end{array}$$

Анти- и истинные зависимости между вторым и третьим операторами исчезли.

1.3 Таймирующие функции

Пусть любая из функций $t^{(\beta)} : V_\beta \to \mathbf{Z},\ \ 1 \leqslant \beta \leqslant K$, ставит в соответствие каждой операции $S_\beta(J)$ алгоритма целое число $t^{(\beta)}(J)$.

Определение (таймирующие функции).

Функции $t^{(1)}, t^{(2)}, \ldots, t^{(K)}$ называются таймирующими (t-функциями), если

$$t^{(\beta)}(J) \geqslant t^{(\alpha)}(I), \quad J \in V_\beta,\ I \in V_\alpha,\ \ S_\alpha(I) \to S_\beta(J). \tag{1.4}$$

Набор функций $\{t^{(1)}, t^{(2)}, \ldots, t^{(K)}\}$ называется таймированием и обозначается t. Условия (1.4) называются условиями сохранения зависимостей.

Функции $t^{(1)}, t^{(2)}, \ldots, t^{(K)}$ называются расщепляющими (f-функциями[3]), если

$$t^{(\beta)}(J) = t^{(\alpha)}(I), \quad J \in V_\beta,\ I \in V_\alpha,\ \ S_\alpha(I) \to S_\beta(J). \tag{1.5}$$

Функции $t^{(1)}, t^{(2)}, \ldots, t^{(K)}$ называются строгими таймирующими (строгими t-функциями), если

$$t^{(\beta)}(J) > t^{(\alpha)}(I), \quad J \in V_\beta,\ I \in V_\alpha,\ \ S_\alpha(I) \to S_\beta(J). \tag{1.6}$$

Условия (1.6) называются строгими условиями сохранения зависимостей.

Из определения следует, что f-функции и строгие t-функции являются частными случаями t-функций.

[3]fission — расщепление, fibering — расслоение

Если зависимости алгоритма заданы функциями вида (1.3), то условия (1.4), (1.5), (1.6) примут соответственно вид

$$t^{(\beta)}(J) \geqslant t^{(\alpha)}(\Phi_{\alpha,\beta}J - \varphi^{(\alpha,\beta)}), \quad J \in V_{\alpha,\beta}, \quad (\alpha,\beta) \in P, \tag{1.7}$$

$$t^{(\beta)}(J) = t^{(\alpha)}(\Phi_{\alpha,\beta}J - \varphi^{(\alpha,\beta)}), \quad J \in V_{\alpha,\beta}, \quad (\alpha,\beta) \in P, \tag{1.8}$$

$$t^{(\beta)}(J) > t^{(\alpha)}(\Phi_{\alpha,\beta}J - \varphi^{(\alpha,\beta)}), \quad J \in V_{\alpha,\beta}, \quad (\alpha,\beta) \in P. \tag{1.9}$$

Операция, которой приписано меньшее значение таймирующей функции, не может зависеть от операции, которой приписано большее значение. Чтобы убедиться в этом, предположим противное: $t^{(\beta)}(J) < t^{(\alpha)}(I)$ и $S_\beta(J)$ зависит от $S_\alpha(I)$. Так как $S_\alpha(I) \to S_\beta(J)$, то из определения таймирующих функций $t^{(\beta)}(J) \geqslant t^{(\alpha)}(I)$, что противоречит предположению $t^{(\beta)}(J) < t^{(\alpha)}(I)$.

Если заданы таймирующие функции $t^{(1)}, t^{(2)}, \ldots, t^{(K)}$, то алгоритм можно разбить на части, отнеся к одной части операции алгоритма с одним и тем же значением функций: $t^{(\alpha)}(I) = t^{(\beta)}(J)$ для любых операций $S_\alpha(I)$ и $S_\beta(J)$ из одной части. Эти части алгоритма можно выполнять последовательно друг за другом в порядке возрастания значений функции. Таймирующие функции определяют так называемые информационные разрезы алгоритма.

Если функции являются строгими таймирующими, то все операции в каждой из упомянутых частей алгоритма могут быть выполнены параллельно. Действительно, если для строгих таймирующих функций имеет место $t^{(\alpha)}(I) = t^{(\beta)}(J)$, то операции $S_\alpha(I)$ и $S_\beta(J)$ не могут быть зависимыми. В противном случае из условий (1.6) следовало бы или $t^{(\alpha)}(I) > t^{(\beta)}(J)$, или $t^{(\beta)}(J) > t^{(\alpha)}(I)$. Таким образом, строгие t-функции задают параллельную форму алгоритма. К одному ярусу параллельной формы относятся операции с одним и тем же значением строгих t-функций.

Если функции являются расщепляющими, то все части алгоритма могут быть выполнены параллельно и независимо друг от друга. Действительно, согласно (1.5) все зависимые операции имеют одинаковые значения f-функций, и попадают в одну часть алгоритма. Таким образом, f-функции разбивают операции алгоритма на параллельные множества.

Параллелизм, задаваемый таймирующими функциями, называется скошенным. Частным случаем скошенного параллелизма является

координатный параллелизм. Координатный параллелизм определяется параллельным циклом, параметром которого является один из параметров исходного гнезда циклов.

Значения t-функций $t^{(1)}, t^{(2)}, \ldots, t^{(K)}$ можно рассматривать как номера итераций некоторых циклов (каждой функции $t^{(\beta)}$ соответствует один цикл, но циклы, соответствующие двум функциям $t^{(\alpha)}$ и $t^{(\beta)}$ не обязательно отличаются). Такая интерпретация допустима: как было показано выше, операция, которой приписано меньшее значение t-функции (операция, выполняемая на итерации с меньшим номером), не может зависеть от операции, которой приписано большее значение (операция, выполняемая на итерации с большим номером).

Пример 2 (продолжение). Приведем примеры таймирующих функций для алгоритма (1.2).

Строгими t-функциями являются функции $t^{(1)}(i) = 0, \;\; i \in V_1$, $t^{(2)}(i,j) = j, \;\; (i,j) \in V_2$. Действительно, покажем, что эти функции удовлетворяют условиям (1.9). Так как $P = \{(1,2),(2,2)\}$, то условия (1.9) примут вид: $t^{(2)}(J) > t^{(1)}(\Phi_{1,2}J - \varphi^{(1,2)}), \;\; J = (i,1) \in V_{1,2}, \;\; t^{(2)}(J) > t^{(2)}(\Phi_{2,2}J - \varphi^{(2,2)}), \;\; J = (i,j) \in V_{2,2}$. Если $(i,1) \in V_{1,2}$, то $t^{(1)}(\Phi_{1,2}(i\;1)^{\mathrm{T}} - \varphi^{(1,2)}) = t^{(1)}((1\;0)(i\;1)^{\mathrm{T}}) = t^{(1)}(i) = 0 < 1 = t^{(2)}(i,1)$. Если $(i,j) \in V_{2,2}$, то $t^{(2)}(\Phi_{2,2}(i\;\;j)^{\mathrm{T}} - \varphi^{(2,2)}) = t^{(2)}((i,j) - (0,1)) = t^{(2)}(i, j-1) = j - 1 < j = t^{(2)}(i,j)$. Таким образом, все операции $S_1(i), \;\; i \in V_1$, могут быть выполнены параллельно; все операции $S_2(i,j), \;\; (i,j) \in V_2$, с одинаковым значением j, могут быть выполнены параллельно. Алгоритм (1.2), распараллеленный с помощью строгих t-функций $t^{(1)}(i) = 0, \;\; t^{(2)}(i,j) = j$, можно записать в следующем виде[4]:

```
dopar  i = 1, N
   S1 :  c(i) = 0
enddo
do  j = 1, N
   dopar  i = 1, N
      S2 :  c(i) = c(i) + a(i, j)b(j)
   enddo
enddo
```

Функции $t^{(1)}(i) = i, \;\; i \in V_1, \;\; t^{(2)}(i,j) = i, \;\; (i,j) \in V_2$ являются

[4]Алгоритм преобразования гнезд циклов с помощью таймирующих функций рассмотрен в разделе 2.5

f-функциями. Действительно, покажем, что эти функции удовлетворяют условиям (1.8): если $(i,1) \in V_{1,2}$, то $t^{(1)}(\Phi_{1,2}(i\ 1)^{\mathrm{T}} - \varphi^{(1,2)}) = t^{(1)}((1\ 0)(i\ 1)^{\mathrm{T}}) = t^{(1)}(i) = i = t^{(2)}(i,1)$; если $(i,j) \in V_{2,2}$, то $t^{(2)}(\Phi_{2,2}(i\ j)^{\mathrm{T}} - \varphi^{(2,2)}) = t^{(2)}((i,j) - (0,1)) = t^{(2)}(i,j-1) = i = t^{(2)}(i,j)$. Таким образом, все операции $S_1(i)$, $i \in V_1$, и $S_2(i,j)$, $(i,j) \in V_2$, с одинаковым значением i могут быть выполнены независимо от других операций. Алгоритм (1.2), распараллеленный с помощью f-функций $t^{(1)}(i) = i$, $t^{(2)}(i,j) = i$, можно записать в виде

```
dopar  i = 1, N
   S_1 :  c(i) = 0
   do  j = 1, N
      S_2 :  c(i) = c(i) + a(i,j)b(j)
   enddo
enddo
```

□

Обозначим $n = \max\limits_{1 \leqslant \beta \leqslant K} n_\beta$. Пусть любая из функций $\bar{t}^{(\beta)} : V_\beta \to \mathbf{Z}^n$, $1 \leqslant \beta \leqslant K$, ставит в соответствие каждой операции $S_\beta(J)$ вектор $(t_1^{(\beta)}(J), \cdots, t_n^{(\beta)}(J))$ с целочисленными координатами. Будем предполагать, что функции $t_\xi^{(\beta)}$ являются аффинными (для каждого фиксированного β и ξ):

$$\begin{gathered} t_\xi^{(\beta)}(J) = \tau^{(\beta,\xi)} J + a_{\beta,\xi}, \quad J \in V_\beta, \\ \tau^{(\beta,\xi)} = (\tau_1^{(\beta,\xi)}, \ldots, \tau_{n_\beta}^{(\beta,\xi)}) \in \mathbf{Z}^{n_\beta},\ a_{\beta,\xi} \in \mathbf{Z}, \\ 1 \leqslant \beta \leqslant K,\ \ 1 \leqslant \xi \leqslant n. \end{gathered} \tag{1.10}$$

Равенства (1.10) можно записать в матрично-векторном виде:

$$\bar{t}^{(\beta)}(J) = T^{(\beta)} J + a^{(\beta)}, \quad J \in V_\beta,\ \ 1 \leqslant \beta \leqslant K,$$

где $T^{(\beta)}$ — матрица, строки которой составлены из векторов $\tau^{(\beta,1)}, \ldots, \tau^{(\beta,n)}$, $\quad a^{(\beta)} = \begin{pmatrix} a_{\beta,1} \\ \cdots \\ a_{\beta,n} \end{pmatrix}$.

Определение (векторные таймирующие функции). *Функции $\bar{t}^{(1)}, \bar{t}^{(2)}, \ldots, \bar{t}^{(K)}$ называются векторными таймирующими функциями (векторными t-функциями, n-мерными t-функциями), если*

$$\operatorname{rank} T^{(\beta)} = n_\beta, \quad 1 \leqslant \beta \leqslant K, \tag{1.11}$$

$$\bar{t}^{(\beta)}(J) \geqslant_{lex} \bar{t}^{(\alpha)}(I), \quad J \in V_\beta,\ I \in V_\alpha,\ S_\alpha(I) \to S_\beta(J). \tag{1.12}$$

Условия (1.12) называются условиями сохранения зависимостей.

Определение (многомерное таймирование). *Набор векторных t-функций* $\{\bar{t}^{(1)}, \bar{t}^{(2)}, \ldots, \bar{t}^{(K)}\}$ *называется многомерным таймированием и обозначается* $\bar{t}$; *набор функций* $\{t_\xi^{(1)},\ t_\xi^{(2)}, \ldots,\ t_\xi^{(K)}\}$ *называется* ξ*-й координатой (компонентой) многомерного таймирования и обозначается* t_ξ. *Примем обозначения многомерного таймирования:* $\bar{t} = \{\bar{t}^{(1)}, \bar{t}^{(2)}, \ldots, \bar{t}^{(K)}\}$, $\bar{t} = (t_1, t_2, \ldots, t_n)$.

Из определения следует, что первая координата многомерного таймирования составлена из одномерных t-функций, остальные координаты — из функций, не обязательно являющихся одномерными t-функциями.

Функции $t_\xi^{(\beta)}$ можно использовать для преобразования циклов, полагая, что операция $S_\beta(J)$, выполняемая на итерации J, будет выполняться на итерации $\bar{t}^{(\beta)}(J)$. Таким образом, компоненты вектора $\bar{t}^{(\beta)}$ интерпретируются как параметры преобразованного гнезда циклов для оператора S_β : $t_1^{(\beta)}$ — параметр самого внешнего цикла, $t_n^{(\beta)}$ — параметр самого внутреннего цикла.

Условие (1.11) формализует требование невозможности более чем однократного срабатывания одного оператора на одной итерации преобразованного гнезда циклов. Действительно, если (1.11) не выполняется, то $\dim\big(\ker T^{(\beta)}\big) = n_\beta - \operatorname{rank} T^{(\beta)} \geq 1$. Пусть u — вектор из $\ker T^{(\beta)}$; тогда для $J,\ J + u \in V_\beta$ имеем $\bar{t}^{(\beta)}(J + u) = \bar{t}^{(\beta)}(J)$, т. е. оператор S_β срабатывает более одного раза на итерации $\bar{t}^{(\beta)}(J)$.

1.4 Аффинные преобразования гнезд вложенных циклов

Известно много различных преобразований циклов. Наиболее широкое применение на практике получили такие преобразования: распределение циклов, слияние циклов, унимодулярные преобразования (перестановка, обращение, скашивание циклов), блокинг, переупорядочение операторов, расщепление пространства итераций, выделение стандартных операций, полная или частичная раскрутка цикла по итерациям, вынесение за пределы цикла нескольких первых или послед-

них итераций. Заметим, что в разделе 1.1 применялась раскрутка циклов по нескольким итерациям для получения зависимостей алгоритма, а в разделе 2.1 будет применяться распределение циклов для выявления координатного параллелизма. В этом разделе рассматриваются аффинные преобразования гнезд вложенных циклов. Аффинные преобразования обобщают некоторые известные (в частности, унимодулярные) преобразования циклов.

1.4.1 Преобразования тесно вложенных циклов

Рассмотрим сначала подробно случай аффинного преобразования гнезда тесно вложенных циклов с глубиной вложенности n. Индексную область операторов обозначим через V. Пусть $\bar{t}^{(\beta)} : V \to \mathbf{Z}^n$ являются такими векторными t-функциями, координаты которых имеют вид (1.10), причем при фиксированных ξ векторы $\tau^{(\beta,\xi)}$ одинаковы для всех β :

$$\begin{gathered} t_{\xi}^{(\beta)}(J) = \tau^{(\xi)} J + a_{\beta,\xi}, \quad J \in V, \\ \tau^{(\xi)} = (\tau_1^{(\xi)}, \ldots, \tau_n^{(\xi)}) \in \mathbf{Z}^n, \quad a_{\beta,\xi} \in \mathbf{Z}, \\ 1 \leqslant \beta \leqslant K, \quad 1 \leqslant \xi \leqslant n. \end{gathered} \tag{1.13}$$

В матрично-векторной форме записи равенства (1.13) имеют вид

$$\bar{t}^{(\beta)}(J) = TJ + a^{(\beta)}, \quad J \in V, \quad 1 \leqslant \beta \leqslant K,$$

где T — матрица, строки которой составлены из векторов $\tau^{(1)}, \ldots, \tau^{(n)}$, J — вектор-столбец, $a^{(\beta)} = \begin{pmatrix} a_{\beta,1} \\ \cdots \\ a_{\beta,n} \end{pmatrix}$.

Определение (допустимое преобразование циклов). *Преобразование гнезда циклов называется допустимым, если оно сохраняет порядок выполнения зависимых операций.*

Зафиксируем d, $1 \leqslant d \leqslant n$, и обозначим $\bar{t}_{1:d}^{(\beta)} = (t_1^{(\beta)}, \cdots, t_d^{(\beta)})$. Введем еще в рассмотрение граф $G_0(\bar{t})$ — остовный подграф редуцированного графа зависимостей, в котором оставлены только такие дуги (v_α, v_β), что существуют зависимые операции $S_\alpha(I)$, $S_\beta(J)$, для которых $\bar{t}^{(\alpha)}(I) = \bar{t}^{(\beta)}(J)$. Граф $G_0(\bar{t})$ выражает внутриитерационные зависимости преобразованного гнезда циклов.

Теорема 1.1 *Пусть* $\bar{t}^{(\beta)} = (t_1^{(\beta)}, \ldots, t_n^{(\beta)})$, $1 \leqslant \beta \leqslant K$, — *векторные таймирующие функции.*

1. Функции $\bar{t}^{(\beta)}$ *задают допустимое преобразование гнезда циклов.*

2. Если для какого-либо d, $1 \leqslant d \leqslant n-1$, *выполняется*

$$\left[\begin{array}{l} \bar{t}^{(\beta)}_{1:d}(J) >_{lex} \bar{t}^{(\alpha)}_{1:d}(I) \text{ или} \\ \bar{t}^{(\beta)}(J) = \bar{t}^{(\alpha)}(I), \end{array}\right. \qquad I, J \in V, \quad S_\alpha(I) \to S_\beta(J), \tag{1.14}$$

то $n-d$ *внутренних циклов преобразованного гнезда циклов являются параллельными.*

3. Если для какого-либо d, $1 \leqslant d \leqslant n-1$, *функции* $t_{d+1}^{(\beta)}, \ldots, t_n^{(\beta)}$, $1 \leqslant \beta \leqslant K$, *являются одномерными t-функциями, то векторные функции* $(t_1^{(\beta)} + \sum\limits_{\xi=d+1}^{n} t_\xi^{(\beta)}, t_2^{(\beta)}, \ldots, t_n^{(\beta)})$ *являются таймирующими и задают преобразование гнезд циклов, приводящее к* $n-d$ *параллельным внутренним циклам.*

4. Если для какого-либо d, $1 \leqslant d \leqslant n$, *выполняется*

$$\bar{t}^{(\beta)}_{1:d}(J) = \bar{t}^{(\alpha)}_{1:d}(I), \quad I, J \in V, \quad S_\alpha(I) \to S_\beta(J), \tag{1.15}$$

то d *внешних циклов преобразованного гнезда циклов являются параллельными.*

Доказательство. Будем полагать, что операторы, выполняемые на одной итерации преобразованного гнезда циклов, упорядочены в соответствии с графом $G_0(\bar{t})$. Покажем, что такое упорядочение возможно.

Для этого докажем, что граф $G_0(\bar{t})$ является ациклическим. Предположим противное, наличие контура $(v_{\alpha_1}, v_{\alpha_2}), (v_{\alpha_2}, v_{\alpha_3}), \ldots, (v_{\alpha_k}, v_{\alpha_1})$. Тогда существуют зависимости

$$\begin{array}{l} S_{\alpha_1}(I^{(1)}) \to S_{\alpha_2}(J^{(1)}),\ S_{\alpha_2}(I^{(2)}) \to S_{\alpha_3}(J^{(2)}), \ldots, \\ S_{\alpha_k}(I^{(k)}) \to S_{\alpha_1}(J^{(k)}) \end{array} \tag{1.16}$$

такие, что $TI^{(1)} + a^{(\alpha_1)} = TJ^{(1)} + a^{(\alpha_2)}$, $TI^{(2)} + a^{(\alpha_2)} = TJ^{(2)} + a^{(\alpha_3)}$, $\ldots$, $TI^{(k)} + a^{(\alpha_k)} = TJ^{(k)} + a^{(\alpha_1)}$. Сложив эти равенства, получим $T\sum\limits_{j=1}^{k}(J^{(j)} - I^{(j)}) = 0$. Обозначим T_i — i-й столбец матрицы T, $J(J_1, \ldots, J_n) = \sum\limits_{j=1}^{k}(J^{(j)} - I^{(j)})$. Тогда $T_1J_1 + \ldots + T_nJ_n = 0$. Столбцы матрицы T

линейно независимы (это следует из определения векторной таймирующей функции), поэтому должно выполняться $J_1 = 0, \ldots, J_n = 0$, т. е. $\sum_{j=1}^{k}(J^{(j)} - I^{(j)}) = 0$. Так как $J^{(j)} \geqslant_{lex} I^{(j)}$, то получим $J^{(j)} = I^{(j)}$; все зависимости (1.16) являлись внутриитерационными и до преобразования гнезда циклов. Получили противоречие: операторы $S_{\alpha_1}, S_{\alpha_2}, \ldots, S_{\alpha_k}$ не могли быть упорядочены в исходном гнезде циклов, так как была внутриитерационная зависимость оператора с меньшим номером от оператора с большим номером.

Перейдем непосредственно к доказательству утверждений теоремы.

1. Пусть $S_\alpha(I) \to S_\beta(J)$. Требуется доказать, что после преобразования гнезда циклов $S_\alpha(I)$ выполняется раньше $S_\beta(J)$. Операция $S_\alpha(I)$ выполняется на итерации $\overline{t}^{(\alpha)}(I)$, операция $S_\beta(J)$ — на итерации $\overline{t}^{(\beta)}(J)$. Так как $\overline{t}^{(1)}, \ldots, \overline{t}^{(K)}$ являются n-мерными t-функциями, то $\overline{t}^{(\beta)}(J) \geqslant_{lex} \overline{t}^{(\alpha)}(I)$. Если $\overline{t}^{(\beta)}(J) >_{lex} \overline{t}^{(\alpha)}(I)$, то это означает, что $S_\alpha(I)$ выполняется раньше $S_\beta(J)$. Если $\overline{t}^{(\beta)}(J) = \overline{t}^{(\alpha)}(I)$, то $S_\alpha(I)$ выполняется раньше $S_\beta(J)$, так как операторы, выполняемые на одной итерации, упорядочены в соответствии с графом $G_0(\overline{t})$.

2. Для параллельности $n - d$ внутренних циклов преобразованного гнезда циклов требуется доказать, что после преобразования циклов не существует зависимостей по $n - d$ внутренним циклам. Формально, требуется показать, что если операция $S_\beta(J)$ зависит от операции $S_\alpha(I)$ и $\overline{t}^{(\beta)}_{1:d}(J) = \overline{t}^{(\alpha)}_{1:d}(I)$ (т. е. параметры d внешних циклов после преобразования совпадают), то $\overline{t}^{(\beta)}(J) = \overline{t}^{(\alpha)}(I)$. Нетрудно видеть, что требуемое является прямым следствием условия (1.14). Таким образом, все итерации $n - d$ внутренних циклов могут выполняться одновременно; любая зависимость $S_\alpha(I) \to S_\beta(J)$ будет после преобразования или внутриитерационной (если $\overline{t}^{(\beta)}(J) = \overline{t}^{(\alpha)}(I)$), или зависимостью по какому-либо из d внешних циклов (если $\overline{t}^{(\beta)}_{1:d}(J) >_{lex} \overline{t}^{(\alpha)}_{1:d}(I)$).

3. Для доказательства того, что векторные функции $(t_1^{(\beta)} + \sum_{\xi=d+1}^{n} t_\xi^{(\beta)}, t_2^{(\beta)}, \ldots, t_n^{(\beta)})$ являются таймирующими, покажем выполнение для них условий (1.12) (справедливость условия (1.11) очевидна: при элементарных преобразованиях ранг матрицы не меняется). Пусть

$S_\alpha(I) \to S_\beta(J)$. Так как функции $t_1^{(\beta)}$, $t_{d+1}^{(\beta)}$, ..., $t_n^{(\beta)}$ являются одномерными t-функциями, то

$$\begin{array}{l} t_1^{(\beta)}(J) \geqslant t_1^{(\alpha)}(I), \\ t_{d+1}^{(\beta)}(J) \geqslant t_{d+1}^{(\alpha)}(I), \ldots, t_n^{(\beta)}(J) \geqslant t_n^{(\alpha)}(I). \end{array} \tag{1.17}$$

Следовательно, $t_1^{(\beta)}(J) + \sum\limits_{\xi=d+1}^{n} t_\xi^{(\beta)}(J) \geqslant t_1^{(\alpha)}(I) + \sum\limits_{\xi=d+1}^{n} t_\xi^{(\alpha)}(I)$;

поэтому, с учетом $\bar{t}^{(\beta)}(J) \geqslant_{lex} \bar{t}^{(\alpha)}(I)$, получим $(t_1^{(\beta)}(J) + \sum\limits_{\xi=d+1}^{n} t_\xi^{(\beta)}(J)$, $t_2^{(\beta)}(J), \ldots, t_n^{(\beta)}(J)) \geqslant_{lex} (t_1^{(\alpha)}(I) + \sum\limits_{\xi=d+1}^{n} t_\xi^{(\alpha)}(I), t_2^{(\alpha)}(I), \ldots, t_n^{(\alpha)}(I))$.

Для доказательства параллельности $n-d$ внутренних циклов покажем, что эти циклы после преобразования могут порождать только внутриитерационные зависимости. Действительно, если предположить наличие зависимости $S_\alpha(I) \to S_\beta(J)$ по какому-либо циклу ξ, $\xi > d$, то получим противоречие: с одной стороны, должно быть $t_1^{(\beta)}(J) + \sum\limits_{\xi=d+1}^{n} t_\xi^{(\beta)}(J) = t_1^{(\alpha)}(I) + \sum\limits_{\xi=d+1}^{n} t_\xi^{(\alpha)}(I)$, с другой стороны, $t_1^{(\beta)}(J) + \sum\limits_{\xi=d+1}^{n} t_\xi^{(\beta)}(J) > t_1^{(\alpha)}(I) + \sum\limits_{\xi=d+1}^{n} t_\xi^{(\alpha)}(I)$ ввиду справедливости неравенств (1.17) и $t_\xi^{(\beta)}(J) > t_\xi^{(\alpha)}(I)$.

4. Пусть выполняются условия (1.15). Тогда любые две зависимые операции будут после преобразования циклов иметь одинаковые значения параметров d внешних циклов. Поэтому d внешних циклов являются параллельными. □

Пример 3. Рассмотрим гнездо циклов

```
do  i = 1, N
   do  j = 1, N
      S1 :  a(i, j) = i
      S2 :  b(i, j) = b(i, j − 1) + a(i, j)c(i − 1, j)
      S3 :  c(i, j) = 2b(i, j) + a(i, j)
   enddo
enddo
```
(1.18)

и t-функции $t_1^{(\beta)}(i,j) = (1,0)(i,j)$, $t_2^{(\beta)}(i,j) = (0,1)(i,j)$, $\beta = 1,2,3$. Эти таймирующие функции будут получены в подразделе 2.2 Согласно

третьему утверждению теоремы 1.1, векторные функции $\overline{t}^{(\beta)}(i,j) = (((1,0)+(0,1))(i,j),(0,1)(i,j)) = (i+j,j)$, $\beta = 1,2,3$, приводят к гнезду циклов с параллельным внутренним циклом.

Получим преобразованное гнездо циклов.

Аффинное преобразование переводит пару (i,j) в пару (i',j'), где $i' = i+j$, $j' = j$, $1 \leqslant i,j \leqslant N$. Следовательно, $j = j'$, $i = i' - j = i' - j'$, $1 \leqslant i' - j' \leqslant N$, $1 \leqslant j' \leqslant N$. Выразим границы изменения новых параметров циклов в явном виде. Добавив к системе неравенств $1 \leqslant i' - j' \leqslant N$, $1 \leqslant j' \leqslant N$ сумму этих неравенств $2 \leqslant i' \leqslant 2N$, и представив неравенство $1 \leqslant i' - j' \leqslant N$ в виде $i' - N \leqslant j' \leqslant i' - 1$, получим систему $2 \leqslant i' \leqslant 2N$, $i' - N \leqslant j' \leqslant i' - 1$, $1 \leqslant j' \leqslant N$. Таким образом, i' изменяется в пределах от 2 до $2N$, а j' в пределах от $\max(1, i' - N)$ до $\min(i' - 1, N)$. Преобразованное гнездо циклов имеет следующий вид:

```
do  i' = 2, 2N
   dopar  j' = max(1, i' - N), min(i' - 1, N)
      S1 :  a(i' - j', j') = i' - j'
      S2 :  b(i' - j', j') = b(i' - j', j' - 1)+
                           +a(i' - j', j')c(i' - j' - 1, j')
      S3 :  c(i' - j', j') = 2b(i' - j', j') + a(i' - j', j')
   enddo
enddo
```

Границы изменения новых параметров циклов можно также получить, если изобразить преобразованную индексную область (рис. 1.13). При изображении учтено, что угловые точки $(1,1)$, $(1,N)$, $(N,1)$, (N,N) индексной области переходят соответственно в точки $(2,1)$, $(N+1,N)$, $(N+1,1)$, $(2N,N)$. □

Отметим основные этапы получения кода преобразованных гнезд циклов: выразить параметры исходного гнезда циклов через новые параметры циклов, выразить границы изменения параметров преобразованного гнезда циклов, записать код.

Общий способ генерации кода преобразованных вложенных гнезд циклов будет рассмотрен в разделе 2.5

Определение (степень параллелизма). *Число параллельных циклов называется степенью параллелизма вложенного гнезда циклов.*

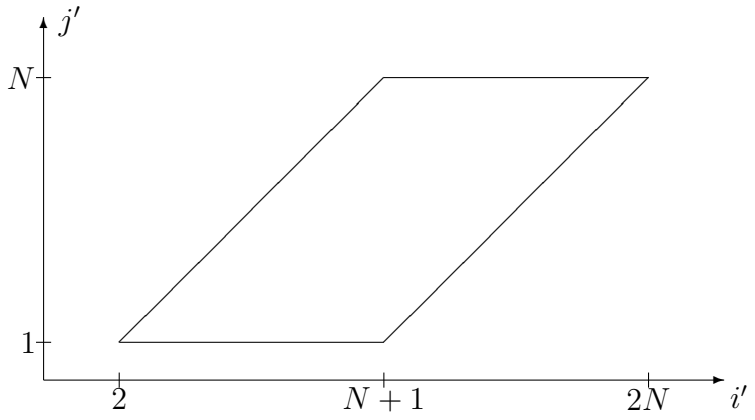

Рис. 1.13

Применение векторных таймирующих функций, удовлетворяющих условиям (1.14) и (1.15), приводит к гнездам циклов со степенью параллелизма $n - d$ и d соответственно.

1.4.2 Применение к не тесно вложенным циклам

Рассмотрим случай гнезда вложенных, но не обязательно тесно, циклов. Будем предполагать, что координаты векторных t-функций имеют вид (1.10), причем при фиксированных ξ векторы $\tau^{(\beta,\xi)}$ одинаковы для всех операторов, расположенных на одном уровне вложенности:

$$\begin{gathered} t_{\xi}^{(\beta)}(J) = \tau_{1:d}^{(\xi)} J + a_{\beta,\xi}, \\ J \in V_\beta \subset \mathbf{Z}^d, \quad \tau_{1:d}^{(\xi)} = (\tau_1^{(\xi)}, \ldots, \tau_d^{(\xi)}) \in \mathbf{Z}^d, \quad a_{\beta,\xi} \in \mathbf{Z}, \\ d = n_\beta, \quad 1 \leqslant \beta \leqslant K, \quad 1 \leqslant \xi \leqslant n. \end{gathered} \tag{1.19}$$

В матрично-векторной форме записи равенства (1.19) имеют вид

$$\bar{t}^{(\beta)}(J) = T_{1:d} J + a^{(\beta)}, \quad J \in V_\beta \subset \mathbf{Z}^d, \quad d = n_\beta, \quad 1 \leqslant \beta \leqslant K,$$

где $T_{1:d} = T^{(\beta)}$ — матрица, строки которой составлены из векторов $\tau_{1:d}^{(1)}, \ldots, \tau_{1:d}^{(n)}$, J — вектор-столбец, $a^{(\beta)} = \begin{pmatrix} a_{\beta,1} \\ \cdots \\ a_{\beta,n} \end{pmatrix}$.

Обозначим $\bar{t}_{1:d}^{(\beta)} = (t_1^{(\beta)}, \cdots, t_d^{(\beta)})$, $G_0(\bar{t}_{1:d})$ — остовный подграф редуцированного графа зависимостей, в котором оставлены только такие

дуги (v_α, v_β), что существуют зависимые операции $S_\alpha(I)$, $S_\beta(J)$, для которых $\bar{t}^{(\alpha)}_{1:d}(I) = \bar{t}^{(\beta)}_{1:d}(J)$. Граф $G_0(\bar{t}_{1:d})$ выражает внутриитерационные зависимости по d первым циклам преобразованного гнезда циклов, поэтому на уровне вложенности d операторы следует упорядочивать в соответствии с графом $G_0(\bar{t}_{1:d})$. Теорема 1.1 в рассматриваемом случае остается справедливой (равенства в условиях (1.14) и (1.15) понимаются как лексикографические).

Рассматриваемые в этом разделе аффинные преобразования вложенных циклов часто применимы и для гнезд циклов с более сложной структурой вложенности. Применение преобразований к не вложенным циклам можно рассматривать как способ получения вложенных циклов. Приведем иллюстрационный пример слияния циклов.

Рассмотрим два не вложенных цикла:

$$\begin{array}{l}\text{do}\ \ i = 1, N\\ \quad S_1:\ x(i) = a(i)\\ \text{enddo}\\ \text{do}\ \ j = 1, N\\ \quad S_2:\ x(j) = x(j) + b(j)\\ \text{enddo}\end{array}$$

Можно показать (см. пример в разделе 2.2), что функции $t^{(1)}(i) = i$, $t^{(2)}(i) = i$ являются f-функциями. Из теоремы 1.1 следует, что порождаемый этими функциями цикл является параллельным:

$$\begin{array}{l}\text{dopar}\ \ i = 1, N\\ \quad S_1:\ x(i) = a(i)\\ \quad S_2:\ x(i) = x(i) + b(i)\\ \text{enddo}\end{array}$$

Глава 2

ПОЛУЧЕНИЕ ТАЙМИРУЮЩИХ ФУНКЦИЙ И ПРЕОБРАЗОВАНИЙ ЦИКЛОВ

Эта глава посвящена методам получения таймирующих функций и преобразований гнезд циклов, приводящих к параллельным циклам.

2.1 Распараллеливание, использующее уровни зависимостей

В этом разделе будет рассмотрен алгоритм Аллена и Кеннеди распараллеливания гнезд тесно вложенных циклов.

Приведем сначала пример, иллюстрирующий использование анализа уровней зависимостей для выявления координатного параллелизма.

Рассмотрим однородное гнездо циклов (алгоритм (1.2) умножения матрицы на вектор, записанный в виде гнезда тесно вложенных циклов)

do $i = 1, N$
  do $j = 0, N$
    S_1 : if $(j = 0)$ then $c(i) = 0$ else
    S_2 : $c(i) = c(i) + a(i, j)b(j)$
  enddo
enddo

и его редуцированный граф зависимостей, помеченный векторами зависимостей (рис. 2.1).

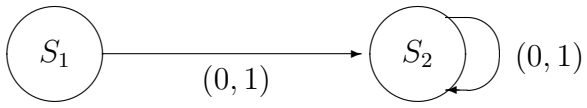

Рис. 2.1

Так как цикл с параметром i (цикл уровня 1) не порождает зависимостей, то его можно записать как параллельный; цикл с парамет-

ром j (цикл уровня 2) порождает зависимости, поэтому он последовательный:

dopar $i = 1, N$
  doseq $j = 0, N$
    S_1 : if $(j = 0)$ then $c(i) = 0$ else
    S_2 : $c(i) = c(i) + a(i, j)b(j)$
  enddo
enddo

Запишем теперь алгоритм умножения матрицы на вектор в следующем виде:

do $j = 0, N$
  do $i = 1, N$
    S_1 : if $(j = 0)$ then $c(i) = 0$ else
    S_2 : $c(i) = c(i) + a(i, j)b(j)$
  enddo
enddo

Изобразим редуцированный граф зависимостей (рис. 2.2).

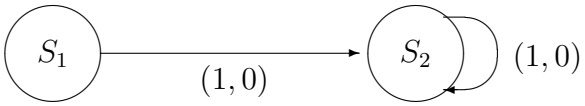

Рис. 2.2

Так как цикл уровня 1 порождает зависимости, то он последовательный; так как все зависимые операции упорядочены в соответствии с циклом уровня 1, то цикл уровня 2 параллельный:

doseq $j = 0, N$
  dopar $i = 1, N$
    S_1 : if $(j = 0)$ then $c(i) = 0$ else
    S_2 : $c(i) = c(i) + a(i, j)b(j)$
  enddo
enddo

Пример показывает, что информации об отсутствии зависимостей по циклу бывает достаточно как для получения параллельных множеств (параллельный внешний цикл), так и для получения параллельных форм алгоритмов (параллельный внутренний цикл).

Алгоритм Аллена и Кеннеди основывается на двух фактах:

1) самый внешний цикл является параллельным, если нет зависимостей, порождаемых циклом, т. е. нет зависимостей уровня 1;

2) все итерации оператора S_α могут быть выполнены перед любой итерацией оператора S_β, если в редуцированном графе зависимостей нет пути из v_β в v_α.

Первое свойство позволяет пометить цикл как dopar или doseq. Из второго свойства следует, что поиск параллелизма можно осуществлять независимо в каждой сильно связной компоненте редуцированного графа зависимостей[1].

Алгоритм 2.1 (распараллеливание, использующее уровни зависимостей). *Пусть G — редуцированный граф зависимостей, дуги которого помечены уровнями зависимостей. Сначала положить $k = 1$.*

1. Удалить из G все дуги уровня меньше k. Полученный граф обозначить $G(k)$.

2. Разложить $G(k)$ на сильно связные компоненты $G_i(k)$ и упорядочить их топологически.

3. Переписать код так, чтобы каждый $G_i(k)$ соответствовал разным гнездам циклов на уровне большем или равном k, и чтобы очередность $G_i(k)$ соответствовала бы шагу 2 (распределение циклов уровня большего или равного k и переупорядочение операторов).

4. Для каждого $G_i(k)$ пометить цикл уровня k как dopar, если $G_i(k)$ не содержит дуг уровня k. В противном случае пометить цикл как doseq.

5. Для каждого $G_i(k)$ обозначить $G = G_i(k)$, увеличить k на 1 и перейти к следующей рекурсии алгоритма, т. е. перейти к шагу 1.

В алгоритме 2.1 выявление параллелизма осуществляется с помощью анализа уровней зависимостей и распределения циклов. Основная идея алгоритма — использовать распределение циклов так, чтобы уменьшить число операторов внутри цикла, и тем самым потенциально уменьшить число зависимостей по циклу.

Известно, что алгоритм 2.1 является оптимальным относительно входных данных. Это означает, что любой алгоритм распараллелива-

[1]Ориентированный граф (подграф) называется сильно связным, если в нем существует замкнутый путь, проходящий через все вершины.

ния, входными данными которого является редуцированный граф зависимостей с дугами, помеченными уровнями зависимостей, не может найти больше параллелизма, чем алгоритм 2.1.

Пример 3 (продолжение). Применим алгоритм 2.1 к гнезду циклов (1.18). Построим редуцированный граф зависимостей (рис. 2.3).

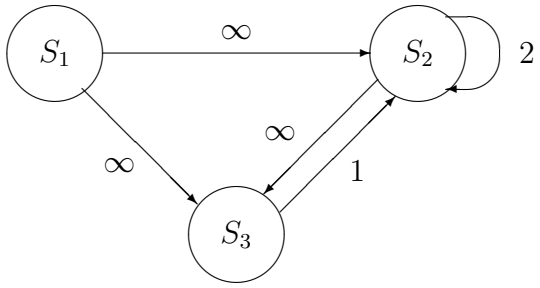

Рис. 2.3

Положим $k = 1$. Граф $G(1)$ совпадает с графом G и содержит две сильно связные компоненты; $G_1(1)$ содержит вершину, соответствующую оператору S_1, $G_2(1)$ содержит вершину, соответствующую S_2 и S_3 (рис. 2.4).

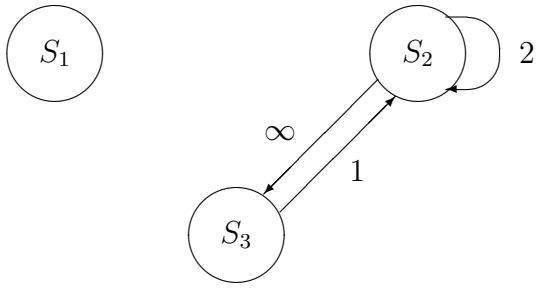

Рис. 2.4

Теперь следует переписать код так, чтобы $G_1(1)$ и $G_2(1)$ соответствовали разным гнездам циклов. Граф $G_1(1)$ не содержит дуг, поэтому соответствующие ему операторы цикла пометим как dopar; граф $G_2(1)$ содержит дугу уровня 1, поэтому оператор цикла do $i = 1, N$ пометим как doseq:

dopar $i = 1, N$
  dopar $j = 1, N$
    $S_1:\ a(i,j) = i$
  enddo
enddo

doseq $i = 1, N$
  do $j = 1, N$
    $S_2:\ b(i,j) = b(i,j-1) + a(i,j)c(i-1,j)$
    $S_3:\ c(i,j) = 2b(i,j) + a(i,j)$
  enddo
enddo

Далее положим $k = 2$ и перейдем к шагу 1 для графа $G_2(1)$. Удалив из $G_2(1)$ дугу уровня 1, получим граф $G(2)$ с двумя сильно связными компонентами (рис. 2.5).

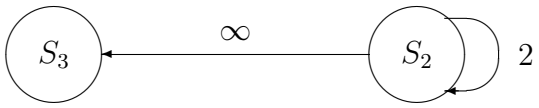

Рис. 2.5

Все итерации оператора S_2 (вычисления, соответствующие компоненте $G_1(2)$) могут быть выполнены раньше любой итерации оператора S_3 (вычисления, соответствующие компоненте $G_2(2)$). Перепишем код, присвоив оператору do $j = 1, N$ тип doseq для случая $G_1(2)$ (так как $G_1(2)$ содержит дугу уровня 2) и тип dopar для случая $G_2(2)$. Окончательно получим:

dopar $i = 1, N$
  dopar $j = 1, N$
    $S_1:\ a(i,j) = i$
  enddo
enddo

doseq $i = 1, N$
  doseq $j = 1, N$
    $S_2:\ b(i,j) = b(i,j-1) + a(i,j)c(i-1,j)$
  enddo

dopar $j = 1, N$
$S_3 :\ c(i,j) = 2b(i,j) + a(i,j)$
enddo
enddo

□

В заключение отметим, что метод Аллена и Кеннеди не выявляет скошенный параллелизм. Он не выявляет также и координатный параллелизм, если для параллелизации циклов требуется их перестановка.

2.2 Получение таймирующих функций для однородных гнезд циклов

Пусть алгоритм задан однородным гнездом тесно вложенных циклов. В этом случае (см. раздел 1.1) зависимости алгоритма выражаются набором векторов зависимостей $\varphi^{(\alpha,\beta)},\ (\alpha,\beta) \in P$. В этом разделе исследуется случай, когда таймирующие функции $t^{(\beta)}(J)$ являются аффинными функциями вида

$$\begin{gathered} t^{(\beta)}(J) = \tau \cdot J + a_\beta, \quad J \in V_\beta, \quad 1 \leqslant \beta \leqslant K, \\ \tau = (\tau_1, \ldots, \tau_n) \in \mathbf{Z}^n,\ a_\beta \in \mathbf{Z}. \end{gathered} \tag{2.1}$$

Введем обозначения:

$\widetilde{\tau} = (\tau_1, \ldots, \tau_n, a_1, \ldots, a_K)$ — вектор параметров функций (2.1);

$\widetilde{\varphi}^{(\alpha,\beta)} = \begin{pmatrix} \varphi^{(\alpha,\beta)} \\ 0 \end{pmatrix} + e^{(n+K)}_{n+\beta} - e^{(n+K)}_{n+\alpha}$, где $e^{(n+K)}_l$ — вектор размера $n+K$, у которого координата с номером l равна 1, а остальные координаты нулевые;

$E^{(l)}$ — единичная матрица порядка l;

D — матрица, столбцы которой составлены из векторов $\widetilde{\varphi}^{(\alpha,\beta)}$; $D \in \mathbf{Z}^{(n+K)\times m}$, где m — число векторов зависимостей $\varphi^{(\alpha,\beta)}$ (число элементов в множестве P). Матрица D называется матрицей зависимостей.

Теорема 2.1 *Пусть над строками матрицы* $(E^{(n+K)}|D)$ *совершена какая-либо последовательность элементарных преобразований*[2],

[2]Элементарными преобразованиями являются сложение, вычитание, перестановка строк и столбцов матрицы, умножение строк и столбцов на ненулевое число.

и в результате получена матрица $(P|B)$, $P \in \mathbf{Z}^{(n+K)\times(n+K)}$, $B \in \mathbf{Z}^{(n+K)\times m}$.

1. Если какая-либо из строк матрицы B *содержит только неотрицательные элементы, то соответствующая строка матрицы* P *определяет вектор параметров t-функций вида (2.1).*

2. Если какая-либо из строк матрицы B *является нулевой, то соответствующая строка матрицы* P *определяет вектор параметров f-функций вида (2.1).*

3. Если какая-либо из строк матрицы B *содержит только положительные элементы, то соответствующая строка матрицы* P *определяет вектор параметров строгих t-функций вида (2.1).*

Доказательство. Докажем справедливость первого утверждения теоремы. Преобразуем условия сохранения зависимостей (1.7) (с учетом того, что $\Phi_{\alpha,\beta} = E^{(n)}$): $t^{(\beta)}(J) - t^{(\alpha)}(J - \varphi^{(\alpha,\beta)}) \geqslant 0$, $\tau J + a_\beta - - \tau(J - \varphi^{(\alpha,\beta)}) - a_\alpha \geqslant 0$, $\tau\varphi^{(\alpha,\beta)} + a_\beta - a_\alpha \geqslant 0$,

$$\widetilde{\tau}\widetilde{\varphi}^{(\alpha,\beta)} \geqslant 0, \quad (\alpha,\beta) \in P. \tag{2.2}$$

Таким образом, для того чтобы функция вида (2.1) являлась таймирующей, необходимо и достаточно выполнения неравенств (2.2), или, в другой форме записи, матрично-векторного неравенства

$$\widetilde{\tau}D \geqslant 0. \tag{2.3}$$

Решение неравенства вида (2.3) сведем к решению вспомогательного уравнения

$$(\widetilde{\tau}|z)\begin{pmatrix} D \\ -E^{(m)} \end{pmatrix} = 0. \tag{2.4}$$

Если найдено такое решение $(\widetilde{\tau}|z)$ уравнения (2.4), что z — вектор, у которого все координаты неотрицательны, то вектор $\widetilde{\tau}$ есть решение неравенства (2.3). Это очевидно, если (2.4) записать в виде $\widetilde{\tau}D = zE^{(m)}$.

Уравнению (2.4) удовлетворяют строки матрицы $(E^{(n+K)}|D)$ (так как $(E^{(n+K)}|D)\begin{pmatrix} D \\ -E^{(m)} \end{pmatrix} = E^{(n+K)}D - DE^{(m)} = D - D = 0$). Строки матрицы $(P|B)$ есть линейные комбинации строк матрицы $(E^{(n+K)}|D)$, поэтому тоже удовлетворяют (2.4). Если какая-либо строка матрицы

B содержит только неотрицательные элементы, то соответствующая строка матрицы P удовлетворяет неравенству (2.3), т. е. определяет вектор параметров t-функции. Первое утверждение теоремы доказано.

Второе утверждение теоремы доказывается аналогично первому, только вместо нестрогих неравенств (2.2) используются равенства $\widetilde{\tau}\widetilde{\varphi}^{(\alpha,\beta)} = 0$, а от вектора z из уравнения (2.4) требуется быть нулевым. Для доказательства третьего утверждения теоремы используются строгие неравенства $\widetilde{\tau}\widetilde{\varphi}^{(\alpha,\beta)} > 0$, а у вектора z из (2.4) должны быть положительные координаты. □

Следствие 2.1 *Для любого однородного гнезда тесно вложенных циклов с глубиной вложенности n существует n таймирований (2.1) с линейно-независимыми векторами τ.*

Действительно, первые ненулевые координаты векторов зависимостей $\varphi^{(\alpha,\beta)}$ положительны, поэтому прибавлением к строкам с большими номерами строк с меньшими номерами можно добиться неотрицательности первых n строк матрицы B. Тогда первые n строк матрицы P будут порождать n t-функций вида (2.1) с линейно-независимыми векторами τ и $a_1 = \ldots = a_K = 0$ (эти строки получены невырожденными преобразованиями n первых строк матрицы $E^{(n+K)}$).

Отметим, что если не учитывать внутриитерационные зависимости, то для любого однородного гнезда тесно вложенных циклов существует строгая t-функция вида (2.1). Действительно, если не принимать во внимание нулевые векторы $\varphi^{(\alpha,\beta)}$, то прибавлением к строкам с большими номерами строк с меньшими номерами можно добиться положительности всех элементов, по крайней мере, на месте последней строки матрицы B (и тем самым показать существование строгой t-функции).

На практике матрицу D лучше преобразовывать таким образом, чтобы элементы матрицы B были как можно ближе к нулю. Как следует из доказательства теоремы 2.1, меньшим значениям элементов матрицы B соответсттвуют меньшие значения величин $t^{(\beta)}(J) - - t^{(\alpha)}(\Phi_{\alpha,\beta}J - \varphi^{(\alpha,\beta)})$. Меньшие значения этих величин приводят к выполнению зависимых операций на более близких итерациях, что важно во многих схемах распараллеливания и организации обмена данными.

Далее нам понадобится понятие эрмитовой нормальной формы матрицы.

Определение (эрмитова нормальная форма матрицы). *Пусть D есть матрица с вещественными элементами, ρ — ранг матрицы D. Тогда элементарными строчными преобразованиями матрицу D можно свести к ее эрмитовой нормальной форме — к матрице H следующей структуры:*

— первые ρ строк матрицы H ненулевые, все элементы остальных строк равны нулю;

— первый ненулевой элемент в i-й ($i = 1, 2, \ldots, \rho$) строке матрицы H равен 1; пусть этот элемент входит в столбец с номером c_i;

— $c_1 < c_2 < \ldots < c_\rho$;

— в столбце с номером c_i все элементы, кроме входящего в i-ю строку, равны нулю.

Матрица вида эрмитовой нормальной формы является в некотором смысле обобщением единичной матрицы, если рассматривать произвольные, а не квадратные невырожденные матрицы.

Теперь можно предложить следующий алгоритм получения параметров таймирующих функций вида (2.1).

Алгоритм 2.2 (получение таймирующих функций для однородных гнезд циклов).

1. Сформировать матрицу D.

2. Элементарными строчными преобразованиями матрицы $(E^{(n+K)}|D)$ получить на месте матрицы D ее эрмитову нормальную форму (можно с точностью до перестановки строк и столбцов и с точностью до положительных множителей строк матрицы).

3. Сложением строк получить на месте матрицы D матрицу, любой столбец которой состоит только из неотрицательных элементов.

4. Обозначить преобразованную матрицу через $(P|B)$. Если какая-либо строка матрицы P содержит не целые элементы, то умножить строку на такое наименьшее натуральное число, чтобы все элементы стали целыми. Для получения t-функций и f-функций воспользоваться первым и вторым утверждением теоремы 2.1. Для получения строгих t-функций рассмотреть любой такой набор из строк матрицы B, чтобы в сумме эти строки порождали вектор с положительными координатами. Сумма этих строк в матрице P приводит к строгой t-функции.

Рассмотрим пример.

Запишем алгоритм (1.2) в виде однородного гнезда тесно вложенных циклов:

$$\begin{array}{l} \text{do}\ \ i = 1, N \\ \quad \text{do}\ \ j = 0, N \\ \quad\quad S_1 :\ \text{if}\ (j = 0)\ \text{then}\ c(i) = 0\ \text{else} \\ \quad\quad S_2 :\ \ c(i) = c(i) + a(i,j)b(j) \\ \quad \text{enddo} \\ \text{enddo} \end{array} \tag{2.5}$$

Развернутый граф зависимостей алгоритма (1.2), приведенный в разделе 1.1, можно рассматривать как граф алгоритма (2.5), если считать, что вершины, соответствующие оператору S_1, расположены в точках $(i, 0)$. Зависимости алгоритма (2.5) можно задать векторами $\varphi^{(1,2)} = (0, 1)$, $\varphi^{(2,2)} = (0, 1)$. Имеем: $n + K = 2 + 2 = 4$, $\widetilde{\tau} = (\tau_1, \tau_2, a_1, a_2)$, $\widetilde{\varphi}^{(1,2)} = (0, 1, -1, 1)$, $\widetilde{\varphi}^{(2,2)} = (0, 1, 0, 0)$, $(E^{(4)}|D) = \left(\begin{array}{cccc|cc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right)$. Вычтем из второй строки четвертую и прибавим к третьей строке четвертую; получим $(P|B) = \left(\begin{array}{cccc|cc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right)$. Из теоремы 2.1 следует, что векторы $\widetilde{\tau} = (1, 0, 0, 0)$ и $\widetilde{\tau} = (0, 0, 1, 1)$ являются векторами параметров f-функций (f-функции $t^{(1)}(i, j) = 1,\ t^{(2)}(i, j) = 1$, определяемые вектором $\widetilde{\tau} = (0, 0, 1, 1)$, задают тривиальное разбиение алгоритма на одну часть), а векторы $\widetilde{\tau} = (0, 1, 0, -1)$ и $\widetilde{\tau} = (0, 0, 0, 1)$ являются векторами параметров t-функций (вектор $\widetilde{\tau} = (0, 0, 0, 1)$ определяет вырожденные t-функции $t^{(1)}(i, j) = 0,\ t^{(2)}(i, j) = 1$). Если в матрице $(P|B)$ сложить вторую и четвертую строки, то придем к вектору $\widetilde{\tau} = (0, 1, 0, 0)$, определяющему строгие t-функции $t^{(1)}(i, j) = j,\ t^{(2)}(i, j) = j$; эти функции можно было получить и сразу из матрицы $(E^{(4)}|D)$.

Пример 3 (продолжение). Получим таймирования для алгоритма (1.18). Пометим дуги редуцированного графа зависимостей векторами зависимостей $\varphi^{(1,2)} = (0, 0)$, $\varphi^{(1,3)} = (0, 0)$, $\varphi^{(2,3)} = (0, 0)$, $\varphi^{(2,2)} = (0, 1)$, $\varphi^{(3,2)} = (1, 0)$ (рис. 2.6).

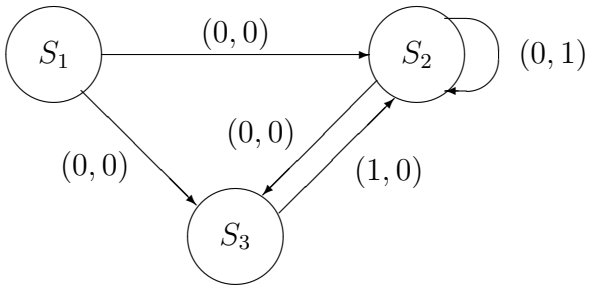

Рис. 2.6

Для получения векторных таймирующих функций используем алгоритм 2.2. Имеем: $n + K = 2 + 3 = 5$, $\widetilde{\tau} = (\tau_1, \tau_2, a_1, a_2, a_3)$, $\widetilde{\varphi}^{(1,2)} = (0, 0, -1, 1, 0)$, $\widetilde{\varphi}^{(1,3)} = (0, 0, -1, 0, 1)$, $\widetilde{\varphi}^{(2,2)} = (0, 1, 0, 0, 0)$, $\widetilde{\varphi}^{(2,3)} = (0, 0, 0, -1, 1)$, $\widetilde{\varphi}^{(3,2)} = (1, 0, 0, 1, -1)$,

$(E^{(5)}|D) = \left(\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & -1 \end{array}\right)$. Вычтем из четвертой строки первую, прибавим к пятой строке первую, к третьей строке четвертую и пятую, к четвертой — пятую; получим $(P|B) =$

$= \left(\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{array}\right)$. Векторы $\widetilde{\tau} = (1, 0, 0, 0, 0)$, $\widetilde{\tau} = (0, 1, 0, 0, 0)$, $\widetilde{\tau} = (1, 0, 0, 0, 1)$ являются векторами параметров невырожденных t-функций. Вектор $\widetilde{\tau} = (0, 0, 1, 1, 1)$ задает вырожденную f-функцию. Вектор $\widetilde{\tau} = (0, 0, 0, 1, 1)$ задает вырожденную t-функцию. □

Приведем пример применения алгоритма 2.2 к не вложенным циклам.

Рассмотрим два цикла:

do $i = 1, N$
  $S_1 : \; x(i) = a(i)$
enddo
do $j = 1, N$
  $S_2 : \; x(j) = x(j) + b(j)$
enddo

Произведем раскрутку циклов при $N = 3$:

$$S_1(1):\ x(1) = a(1)$$
$$S_1(2):\ x(2) = a(2)$$
$$S_1(3):\ x(3) = a(3)$$

$$S_2(1):\ x(1) = x(1) + b(1)$$
$$S_2(2):\ x(2) = x(2) + b(2)$$
$$S_2(3):\ x(3) = x(3) + b(3)$$

Видно, что имеют место зависимости $S_1(i) \to S_2(j)$, $i = j$. Таким образом, зависимости определяются вектором $\varphi^{(1,2)} = 0$. Имеем: $n+K = 1+$ $+ 2 = 3$, $\widetilde{\tau} = (\tau_1, a_1, a_2)$, $\widetilde{\varphi}^{(1,2)} = (0, -1, 1)$, $(E^{(3)}|D) = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 \end{array}\right)$.

Из теоремы 2.1 следует, что вектор $\widetilde{\tau} = (1, 0, 0)$ определяет f-функции $t^{(1)}(i) = i$, $t^{(2)}(i) = i$.

Замечание. *Введем понятие эрмитовой целочисленной нормальной формы матрицы. Пусть D — матрица с целочисленными элементами, ρ — ранг матрицы. Тогда сложением, вычитанием и перестановкой строк можно матрицу D свести к целочисленной матрице H следующей структуры:*

— первые ρ строк матрицы H ненулевые, все элементы остальных строк равны нулю;

— первый ненулевой элемент в i-й ($i = 1, 2, \ldots, \rho$) строке матрицы H положителен; пусть первый ненулевой элемент в i-й ($i =$ $= 1, 2, \ldots, \rho$) строке входит в столбец с номером c_i; тогда $c_1 < c_2 <$ $< \ldots < c_\rho$;

— в столбце с номером c_i все элементы неотрицательны, наибольшим является элемент в i-й строке, в строках с номерами, большими i, находятся нули.

В алгоритме (2.2), как и в последующих алгоритмах получения таймирующих функций, можно на шаге 2 приводить матрицу D к ее эрмитовой целочисленной нормальной форме (с точностью до перестановки строк и столбцов). Тогда гарантирована целочисленность элементов матрицы P.

2.3 Необходимые и достаточные условия для сохранения зависимостей

В этом разделе исследуются условия (1.7)–(1.9), которым должны удовлетворять аффинные таймирующие функции вида

$$\begin{gathered} t^{(\beta)}(J) = \tau^{(\beta)} \cdot J + a_\beta, \quad J \in V_\beta, \quad 1 \leqslant \beta \leqslant K, \\ \tau^{(\beta)} = \tau_{1:n_\beta} = (\tau_1, \ldots, \tau_{n_\beta}) \in \mathbf{Z}^{n_\beta}, \quad a_\beta \in \mathbf{Z}. \end{gathered} \tag{2.6}$$

Векторы $\tau^{(\beta)}$ одинаковы для всех операторов, расположенных на одном уровне вложенности. Если $n_\beta = 0$, примем $\tau^{(\beta)} = 0$.

Введем обозначения:

$n = \max\limits_{1 \leqslant \beta \leqslant K} n_\beta$,

$\widetilde{\tau} = (\tau_1, \ldots, \tau_n, a_1, \ldots, a_K)$ — вектор размера $n + K$, составленный из параметров функций (2.6);

$0^{i \times j}$ — нулевая матрица размера $i \times j$;

$\widetilde{\Phi}_{\alpha,\beta} = \begin{pmatrix} E^{(n_\beta)} \\ 0^{(n-n_\beta+K)\times n_\beta} \end{pmatrix} - \begin{pmatrix} \Phi_{\alpha,\beta} \\ 0^{(n-n_\alpha+K)\times n_\beta} \end{pmatrix}$ — матрица размера $(n + K) \times n_\beta$; если $n_\alpha = 0$, то вычитаемая матрица является нулевой (напомним, $\Phi_{\alpha,\beta} \in \mathbf{Z}^{n_\alpha \times n_\beta}$);

$0^{(i)}$ — нулевой вектор-столбец размера i;

$\widetilde{\varphi}^{(\alpha,\beta)} = \begin{pmatrix} \varphi^{(\alpha,\beta)} \\ 0^{(n-n_\alpha+K)} \end{pmatrix} + e^{(n+K)}_{n+\beta} - e^{(n+K)}_{n+\alpha}$; если $n_\alpha = 0$, то $\widetilde{\varphi}^{(\alpha,\beta)} = e^{(n+K)}_{n+\beta} - e^{(n+K)}_{n+\alpha}$.

Исследуем сначала строгие условия сохранения зависимостей.

Преобразуем строгие условия сохранения зависимостей (1.9): $\tau^{(\beta)} J + a_\beta > \tau^{(\alpha)}(\Phi_{\alpha,\beta} J - \varphi^{(\alpha,\beta)}) + a_\alpha$, $(\tau^{(\beta)} - \tau^{(\alpha)}\Phi_{\alpha,\beta})J + a_\beta - a_\alpha + \tau^{(\alpha)}\varphi^{(\alpha,\beta)} > 0$,

$$\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta} J + \widetilde{\tau}\widetilde{\varphi}^{(\alpha,\beta)} > 0, \quad J \in V_{\alpha,\beta}, \ (\alpha,\beta) \in P. \tag{2.7}$$

Неравенства (2.7) задают ограничения, которым должны удовлетворять аффинные функции вида (2.6), чтобы быть строгими таймирующими. Число неравенств в ограничениях (2.7) большое и зависит от числа точек J в множествах $V_{\alpha,\beta}$. Исключение составляют случаи $\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta} = 0$, в частности, рассмотренный в разделе 2.2 случай $\tau^{(\alpha)} = \tau^{(\beta)}$, $\Phi_{\alpha,\beta}$ — единичная матрица. Для практического использования ограничений (2.7) следует уменьшить число неравенств.

Рассмотрим необходимые и достаточные условия сохранения зависимостей для двух часто встречающихся случаев. Для обозначения k-той координаты векторов наряду с обозначением типа v_k будем использовать обозначение $(v)_k$. Запись типа $(\Phi)_k$ обозначает k-й столбец матрицы Φ. Неравенство векторов понимается как неравенство всех соответствующих координат векторов.

Теорема 2.2 *Пусть $p^{(\alpha,\beta)}$ — такой вектор, что $p^{(\alpha,\beta)} \leqslant J$ для всех $J \in V_{\alpha,\beta}$. Для того чтобы для данной пары $(\alpha,\beta) \in P$ строгие условия сохранения зависимостей выполнялись при любых значениях внешних переменных, достаточно выполнения условий*

$$\widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta}p^{(\alpha,\beta)} + \widetilde{\varphi}^{(\alpha,\beta)}) > 0,\ \widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta} \geqslant 0. \tag{2.8}$$

Условия (2.8) являются необходимыми, если $p^{(\alpha,\beta)} \in V_{\alpha,\beta}$, вектор $p^{(\alpha,\beta)}$ и функции $\overline{\Phi}_{\alpha,\beta}$, $t^{(\alpha)}$, $t^{(\beta)}$ не зависят от внешних переменных, и для любого натурального N существуют такие значения внешних переменных, что $p^{(\alpha,\beta)} + Ne_i^{(n_\beta)} \in V_{\alpha,\beta},\ 1 \leqslant i \leqslant n_\beta$.

Доказательство. Строгие условия сохранения зависимостей (2.7) запишем в виде

$$\begin{gathered}\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta}(J - p^{(\alpha,\beta)}) + \widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta}p^{(\alpha,\beta)} + \widetilde{\varphi}^{(\alpha,\beta)}) > 0,\\ J \in V_{\alpha,\beta},\ (\alpha,\beta) \in P.\end{gathered} \tag{2.9}$$

Для доказательства достаточности заметим, что из справедливости условий (2.8) для данной пары (α,β) следует справедливость условия (2.9), а значит и условия сохранения зависимостей (2.7).

Докажем необходимость. Предположим, что условия (2.8) не выполняются для какой-либо пары (α,β). Если в (2.8) не выполняется первое неравенство, то (2.7) не выполняется при $J = p^{(\alpha,\beta)} \in V_{\alpha,\beta}$. Пусть не выполняется второе неравенство, например $(\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta})_1 < 0$. Выберем $J = p^{(\alpha,\beta)} + Ne_1^{(n_\beta)}$, где N — произвольное фиксированное натуральное число. Согласно предположениям теоремы выбранное J принадлежит области $V_{\alpha,\beta}$. Тогда $\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta}(J - p^{(\alpha,\beta)}) = N(\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta})_1$, для достаточно большого N условие (2.9) $N(\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta})_1 + \widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta}p^{(\alpha,\beta)} + \widetilde{\varphi}^{(\alpha,\beta)}) > 0$, а значит и условие (2.7), не выполняются. $\square$

Теорема 2.3 *Пусть для всех* $J = (J_1, \ldots, J_{n_\beta}) \in V_{\alpha,\beta}$ *выполняется* $J_{k_1} \leqslant J_{k_i} + q_i^{(\alpha,\beta)}$, $i = 2, \ldots, m_{\alpha,\beta}$, *где* $k_1, k_2, \ldots, k_{m_{\alpha,\beta}}$ — *фиксированные индексы,* $q_i^{(\alpha,\beta)} \in \mathbf{Z}$, $m_{\alpha,\beta} \in \{2, \ldots, n_\beta\}$. *Пусть* $p^{(\alpha,\beta)} = (p_1^{(\alpha,\beta)}, \ldots, p_{n_\beta}^{(\alpha,\beta)})$ — *такой вектор, что* $p^{(\alpha,\beta)} \leqslant J$ *для всех* $J \in V_{\alpha,\beta}$, *причем* $p_{k_1}^{(\alpha,\beta)} = p_{k_i}^{(\alpha,\beta)} + q_i^{(\alpha,\beta)}$, $i = 2, \ldots, m_{\alpha,\beta}$. *Для того чтобы для данной пары* $(\alpha, \beta) \in P$ *строгие условия сохранения зависимостей выполнялись при любых значениях внешних переменных, достаточно выполнения условий*

$$\begin{gathered} \widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta} p^{(\alpha,\beta)} + \widetilde{\varphi}^{(\alpha,\beta)}) > 0, \\ \widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta})_k \geqslant 0,\ k \neq k_1,\ \widetilde{\tau} \sum_{i=1}^{m_{\alpha,\beta}} (\widetilde{\Phi}_{\alpha,\beta})_{k_i} \geqslant 0. \end{gathered} \tag{2.10}$$

Условия(2.10) являются необходимыми, если $p^{(\alpha,\beta)} \in V_{\alpha,\beta}$, *вектор* $p^{(\alpha,\beta)}$ *и функции* $\overline{\Phi}_{\alpha,\beta}$, $t^{(\alpha)}$, $t^{(\beta)}$ *не зависят от внешних переменных, и для любого натурального* N *существуют такие значения внешних переменных, что* $p^{(\alpha,\beta)} + N e_k^{(n_\beta)} \in V_{\alpha,\beta}$, $k \neq k_1$, *и выполняется* $p^{(\alpha,\beta)} + N \sum_{i=1}^{m_{\alpha,\beta}} e_{k_i}^{(n_\beta)} \in V_{\alpha,\beta}$.

Доказательство. Для доказательства достаточности покажем справедливость неравенства (2.9). Обозначим $S_{k_1,\ldots,k_{m_{\alpha,\beta}}} = \sum_{i=1}^{m_{\alpha,\beta}} (\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta})_{k_i}(J_{k_i} - p_{k_i}^{(\alpha,\beta)})$, тогда

$$\begin{gathered} \widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta}(J - p^{(\alpha,\beta)}) = \\ = \sum_{k \neq k_i,\ i=1,\ldots,m_{\alpha,\beta}} (\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta})_k (J_k - p_k^{(\alpha,\beta)}) + S_{k_1,\ldots,k_{m_{\alpha,\beta}}}. \end{gathered} \tag{2.11}$$

Перепишем $S_{k_1,\ldots,k_{m_{\alpha,\beta}}}$ следующим образом:

$S_{k_1,\ldots,k_{m_{\alpha,\beta}}} = (\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta})_{k_1}(J_{k_1} - p_{k_1}^{(\alpha,\beta)}) + \sum_{i=2}^{m_{\alpha,\beta}} (\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta})_{k_i}$ $\Big((J_{k_1} - p_{k_1}^{(\alpha,\beta)}) + (J_{k_i} - J_{k_1} + q_i^{(\alpha,\beta)}) + (p_{k_1}^{(\alpha,\beta)} - p_{k_i}^{(\alpha,\beta)} - q_i^{(\alpha,\beta)})\Big) =$ $= (J_{k_1} - p_{k_1}^{(\alpha,\beta)}) \sum_{i=1}^{m_{\alpha,\beta}} (\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta})_{k_i} + \sum_{i=2}^{m_{\alpha,\beta}} (\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta})_{k_i}(J_{k_i} - J_{k_1} + q_i^{(\alpha,\beta)})$. Если условия (2.10) выполняются, то $S_{k_1,\ldots,k_{m_{\alpha,\beta}}} \geqslant 0$, правая часть равенства (2.11) неотрицательна (т. е. $\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta}(J - p^{(\alpha,\beta)}) \geqslant 0$), неравенство (2.9) выполняется.

Докажем необходимость. Предположим, что условия (2.10) не выполняются. Если в (2.10) не выполняется первое неравенство, то (2.7) не выполняется при $J = p^{(\alpha,\beta)} \in V_{\alpha,\beta}$. Пусть не выполняется второе неравенство, т. е. для некоторого $k_0 \neq k_1$, имеет место $\widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta})_{k_0} < 0$. Выберем $J = p^{(\alpha,\beta)} + Ne_{k_0}^{(n_\beta)}$, где N — произвольное фиксированное натуральное число; такое J принадлежит области $V_{\alpha,\beta}$ согласно предположениям теоремы. При выбранном J получим $\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta}(J - p^{(\alpha,\beta)}) = N\widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta})_{k_0}$. Для достаточно большого N условие (2.9) $N\widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta})_{k_0} + \widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta}p^{(\alpha,\beta)} + \widetilde{\varphi}^{(\alpha,\beta)}) > 0$, а значит и условие (2.7), не выполняются. Предположим, что в (2.10) не выполняется третье неравенство, т. е. $\widetilde{\tau}\sum\limits_{i=1}^{m_{\alpha,\beta}}(\widetilde{\Phi}_{\alpha,\beta})_{k_i} < 0$. Выберем $J = p^{(\alpha,\beta)} + N\sum\limits_{i=1}^{m_{\alpha,\beta}} e_{k_i}^{(n_\beta)}$, где N — произвольное фиксированное натуральное число; выбранное J принадлежит области $V_{\alpha,\beta}$. При таком J получим $S_{k_1,\ldots,k_{m_{\alpha,\beta}}} = N\widetilde{\tau}\sum\limits_{i=1}^{m_{\alpha,\beta}}(\widetilde{\Phi}_{\alpha,\beta})_{k_i}$ и, следовательно, $\widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta}(J - p^{(\alpha,\beta)}) = N\widetilde{\tau}\sum\limits_{i=1}^{m_{\alpha,\beta}}(\widetilde{\Phi}_{\alpha,\beta})_{k_i}$. Для достаточно большого N условие (2.9) $N\widetilde{\tau}\sum\limits_{i=1}^{m_{\alpha,\beta}}(\widetilde{\Phi}_{\alpha,\beta})_{k_i} + \widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta}p^{(\alpha,\beta)} + \widetilde{\varphi}^{(\alpha,\beta)}) > 0$, а значит и условие (2.7), не выполняются. □

Таким образом, для получения строгих таймирующих функций можно воспользоваться условиями (2.8) и (2.10). Какое конкретно выбрать условие для пары (α, β) зависит от вида $V_{\alpha,\beta}$. Если $V_{\alpha,\beta}$ — "треугольная" или "пирамидальная" область, то следует использовать условия (2.10), если $V_{\alpha,\beta}$ — прямоугольный параллелепипед, то можно воспользоваться болеее слабыми условиями (2.8).

Для получения не обязательно строгих таймирующих функций можно воспользоваться аналогами условий (2.8)

$$\widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta}p^{(\alpha,\beta)} + \widetilde{\varphi}^{(\alpha,\beta)}) \geqslant 0, \quad \widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta} \geqslant 0 \tag{2.12}$$

или аналогами условий (2.10)

$$\begin{gathered}\widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta}p^{(\alpha,\beta)} + \widetilde{\varphi}^{(\alpha,\beta)}) \geqslant 0,\\ \widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta})_k \geqslant 0,\ k \neq k_1,\ \widetilde{\tau}\sum\limits_{i=1}^{m_{\alpha,\beta}}(\widetilde{\Phi}_{\alpha,\beta})_{k_i} \geqslant 0.\end{gathered} \tag{2.13}$$

Все утверждения относительно необходимости и достаточности этих условий остаются такими же, как в теоремах 2.2 и 2.3.

Для получения расщепляющих функций можно воспользоваться условиями

$$\widetilde{\tau}(\widetilde{\Phi}_{\alpha,\beta}p^{(\alpha,\beta)} + \widetilde{\varphi}^{(\alpha,\beta)}) = 0, \quad \widetilde{\tau}\widetilde{\Phi}_{\alpha,\beta} = 0. \tag{2.14}$$

Пример 2 (продолжение). Напомним, что для алгоритма (1.2) $\Phi_{1,2} = (1\ 0)$, $\varphi^{(1,2)} = 0$, $V_{1,2} = \{(i,1) \in \mathbf{Z}^2 |\ 1 \leqslant i \leqslant N\}$, $\Phi_{2,2} = E^{(2)}$, $\varphi^{(2,2)} = (0,1)$, $V_{2,2} = \{(i,j) \in \mathbf{Z}^2 |\ 1 \leqslant i \leqslant N,\ 2 \leqslant j \leqslant N\}$.

Воспользуемся теоремой 2.2. Имеем:
$t^{(1)}(i) = \tau_1 i + a_1$, $t^{(2)}(i,j) = (\tau_1,\tau_2)(i,j) + a_2$, $\widetilde{\tau} = (\tau_1,\tau_2,a_1,a_2)$, $\widetilde{\Phi}_{1,2} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$, $\widetilde{\varphi}^{(1,2)} = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 1 \end{pmatrix}$, $\widetilde{\Phi}_{2,2} = 0$, $\widetilde{\varphi}^{(2,2)} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$, $p^{(1,2)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $p^{(2,2)} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$, $\widetilde{\Phi}_{1,2}p^{(1,2)} + \widetilde{\varphi}^{(1,2)} = (0,1,-1,1)$, $\widetilde{\Phi}_{2,2}p^{(2,2)} + \widetilde{\varphi}^{(2,2)} = (0,1,0,0)$. Условия (2.8) для $(\alpha,\beta) = (1,2)$ и $(\alpha,\beta) = (2,2)$ имеют вид:

$$\widetilde{\tau}\begin{pmatrix} 0 \\ 1 \\ -1 \\ 1 \end{pmatrix} > 0, \quad \widetilde{\tau}\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} > 0. \tag{2.15}$$

Таким образом, для выполнения строгих условий сохранения зависимостей необходимо и достаточно, чтобы вектор параметров таймирующих функций удовлетворял неравенствам (2.15). □

Пример применения теоремы 2.3 будет приведен в разделе 2.5.

2.4 Получение таймирующих функций для аффинных гнезд циклов

2.4.1 Условия и алгоритм получения t-функций

Применим результаты раздела 2.3 для получения таймирующих функций вида (2.6).

Пусть D_1 — матрица, столбцы которой составлены из ненулевых векторов $\widetilde{\Phi}_{\alpha,\beta}p^{(\alpha,\beta)} + \widetilde{\varphi}^{(\alpha,\beta)}$, D_2 — матрица, столбцы которой составлены из ненулевых столбцов матриц $\widetilde{\Phi}_{\alpha,\beta}$. Если какой-либо столбец матрицы D_1 есть сумма других столбцов или совпадает с другим столбцом, то

его можно опустить[3]. Если какой-либо столбец матрицы D_2 есть сумма других столбцов матриц D_1, D_2 или совпадает с другим столбцом, то его можно опустить.

Матрицы D_1 и D_2 сформированы исходя из условий (2.8) и их аналогов (2.12), (2.14). Если исходить из условий (2.10), (2.13), то следует вместо столбца $(\widetilde{\Phi}_{\alpha,\beta})_{k_1}$ использовать $\sum\limits_{i=1}^{m_{\alpha,\beta}} (\widetilde{\Phi}_{\alpha,\beta})_{k_i}$.

Обозначим $D = (D_1|D_2),\ D \in \mathbf{Z}^{(n+K)\times(\mu_1+\mu_2)}$, где μ_1 и μ_2 — число столбцов матриц D_1 и D_2 соответственно.

Теорема 2.4 *Пусть над строками матрицы $(E^{(n+K)}|D)$ совершена последовательность элементарных преобразований и получена матрица $(P|B),\ P \in \mathbf{Z}^{(n+K)\times(n+K)},\ B \in \mathbf{Z}^{(n+K)\times(\mu_1+\mu_2)}$.*

1. Если первые μ_1 элементов какой-либо строки матрицы B положительны, а следующие μ_2 элементов строки неотрицательны, то соответствующая строка матрицы P определяет вектор параметров $\widetilde{\tau}$ строгих t-функций.

2. Если какая-либо из строк матрицы B является нулевой, то соответствующая строка матрицы P определяет вектор параметров f-функций.

3. Если какая-либо из строк матрицы B содержит только неотрицательные элементы, то соответствующая строка матрицы P определяет вектор параметров t-функций.

Доказательство. Докажем справедливость первого утверждения теоремы. Запишем неравенства (2.8) в виде системы неравенств

$$\widetilde{\tau}D_1 > 0, \quad \widetilde{\tau}D_2 \geqslant 0. \tag{2.16}$$

Решение этой системы сводится к решению системы уравнений

$$(\widetilde{\tau}|z)\begin{pmatrix} D \\ -E^{(\mu_1+\mu_2)} \end{pmatrix} = 0, \tag{2.17}$$

где первые μ_1 координат вектора $z \in \mathbf{Z}^{\mu_1+\mu_2}$ должны быть положительны, а следующие μ_2 координат — неотрицательны. Такое требование к координатам вектора z следует из записи системы (2.17) в виде

[3]Объяснение этому такое: если неравенство в системе (2.16) является следствием других неравенств, то его можно не учитывать.

$\widetilde{\tau}(D_1|D_2) = zE^{(\mu_1+\mu_2)}$. Уравнению (2.17) удовлетворяют строки матрицы $(E^{(n+K)}|D)$, а следовательно и линейные комбинации этих строк, в частности, строки матрицы $(P|B)$. Если первые μ_1 элементов какой-либо строки матрицы B положительны, а следующие μ_2 элементов строки неотрицательны, то вектор, составленный из соответствующей строки матрицы P, удовлетворяет неравенству (2.16), т. е. определяет вектор параметров строгих t-функций. Первое утверждение теоремы доказано.

Второе утверждение теоремы доказывается аналогично первому, только вместо неравенств (2.16) используются равенства $\widetilde{\tau}D_1 = 0$, $\widetilde{\tau}D_1 = 0$, а от вектора z из уравнения (2.17) требуется быть нулевым. Для доказательства третьего утверждения теоремы используются нестрогие неравенства $\widetilde{\tau}D_1 \geqslant 0$, $\widetilde{\tau}D_1 \geqslant 0$, а у вектора z из (2.17) должны быть неотрицательные координаты. □

Следующий алгоритм основан на теореме 2.4.

Алгоритм 2.3 (получение таймирующих функций для аффинных гнезд циклов).

1. Сформировать матрицу $D = (D_1|D_2)$.

2. Элементарными строчными преобразованиями матрицы $(E^{(n+K)}|D)$ *получить на месте матрицы* D *ее эрмитову нормальную форму (можно с точностью до перестановки строк и столбцов и с точностью до положительных множителей строк матрицы).*

3. Сложением строк получить на месте матрицы D *матрицу, состоящую только из неотрицательных элементов. Если этого добиться не удалось, то строки с отрицательными элементами далее не рассматривать.*

4. Обозначить преобразованную матрицу через $(P|B)$. *Если какая-либо строка матрицы* P *содержит не целые элементы, то умножить строку на такое наименьшее натуральное число, чтобы все элементы стали целыми. Для получения f-функций и t-функций воспользоваться вторым и третьим утверждением теоремы 2.4. Для получения строгих t-функций рассмотреть любой такой набор из строк матрицы* B, *чтобы в сумме эти строки порождали вектор, у которого первые* μ_1 *координат положительны, а следующие* μ_2 *координат неотрицательны. Сумма этих строк в матрице* P *приводит к строгим t-функциям.*

Пример 2 (продолжение). С учетом примера 2, рассмотренного в разделе 2.3, имеем: $\mu_1 = 2$, $\mu_2 = 0$, $(E^{(4)}|D) = \left(\begin{array}{cccc|cc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right)$. Прибавив ко второй строке третью, к третьей строке четвертую, получим $(P|B) = \left(\begin{array}{cccc|cc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right)$. Из теоремы 2.4 следует, что векторы $\widetilde{\tau} = (1, 0, 0, 0)$ и $\widetilde{\tau} = (0, 0, 1, 1)$ являются векторами параметров f-функций (f-функции $t^{(1)}(i) = 1,\ t^{(2)}(i, j) = 1$, определяемые вектором $\widetilde{\tau} = (0, 0, 1, 1)$, задают тривиальное разбиение алгоритма на одну часть), а векторы $\widetilde{\tau} = (0, 1, 1, 0)$ и $\widetilde{\tau} = (0, 0, 0, 1)$ являются векторами параметров t-функций (вектор $\widetilde{\tau} = (0, 0, 0, 1)$ определяет вырожденные t-функции $t^{(1)}(i) = 0,\ t^{(2)}(i, j) = 1$). Если в матрице $(P|B)$ сложить вторую и четвертую строки, то придем к вектору $\widetilde{\tau} = (0, 1, 1, 1)$, определяющему строгие t-функции $t^{(1)}(i) = 1,\ t^{(2)}(i, j) = j + 1$. □

2.4.2 Получение наибольшего числа независимых таймирований

Теоретически возможны случаи, в которых применение алгоритма 2.3 не приводит к таймирующим функциям. Следующий алгоритм гарантирует получение t-функций, если только при формировании матрицы D использовались необходимые и достаточные условия сохранения зависимостей.

Алгоритм 2.4 (получение наибольшего числа линейно-независимых решений системы $\widetilde{\tau}D \geqslant 0$, получение таймирующих функций для аффинных гнезд циклов).

1. Элементарными строчными преобразованиями и перестановкой столбцов привести матрицу D к виду $\left(\begin{array}{cc} E & Q \\ 0 & 0 \end{array}\right)$, *где E — единичная матрица или матрица, в каждой строке и каждом столбце которой по одному ненулевому, причем положительному, элементу, Q — матрица, состоящая из рациональных чисел, 0 — нулевые матрицы; матрицы Q, 0 могут отсутствовать. Нулевые строки исключить из рассмотрения.*

2. Если Q состоит только из неотрицательных чисел или отсутствует, то перейти к шагу 6 (к шагу 7, если совершается первая рекурсия алгоритма). Иначе преобразованиями вида $R_i + \lambda R_j$, $\lambda > 0$, (прибавление к строке с номером i строки с номером j, умноженной на положительное число) обнулить отрицательные элементы и получить хотя бы одну строку с неотрицательными элементами. При затруднении найти такую строку следует решить оптимизационную задачу

$$\min\{\lambda_1 + \ldots + \lambda_m \mid \sum_{i=1}^{m} \lambda_i R_i \geqslant 0,\ \sum_{i=1}^{m} \lambda_i \geqslant 1,\ \lambda_i \geqslant 0,\ 1 \leqslant i \leqslant m\},$$

где m — число строк в матрице Q. Если такую строку получить невозможно, то строки матрицы Q исключить из рассмотрения и перейти к шагу 6. Если матрица Q свелась к матрице с неотрицательными элементами, то перейти к шагу 6.

3. Перестановкой строк получить матрицу $\begin{pmatrix} H \\ Q^- \end{pmatrix}$, где строки матрицы H состоят только из неотрицательных элементов, в каждой строке матрицы Q^- есть хотя бы один отрицательный элемент.

4. Если в матрице H нет нулевых столбцов, то перейти к шагу 6. Иначе перестановкой столбцов получить из матрицы $\begin{pmatrix} H \\ Q^- \end{pmatrix}$ матрицу вида $\begin{pmatrix} H_1 & 0 \\ Q_1 & Q_2 \end{pmatrix}$.

5. Перейти к следующей рекурсии процедуры: перейти к шагу 1 применительно к матрице Q_2, полученной на шаге 4 (совершая в дальнейшем над матрицей Q_1 те же самые строчные преобразования, что и над матрицей Q_2).

6. Если вся преобразованная матрица D, кроме строк, исключенных из рассмотрения на шаге 2, состоит только из неотрицательных элементов, то сразу перейти к шагу 7. Если в каких-либо строках матриц Q_1 есть отрицательные элементы (в других строках их быть не может), то обнулить эти элементы, прибавляя к строкам матриц Q_1 строки матриц H_1, умноженные на соответствующие коэффициенты.

7. Все строчные преобразования, которые совершались на всех

шагах и всех рекурсиях над матрицей D, совершить над единичной матрицей $E^{(n+K)}$. Обозначить преобразованную матрицу $(E^{(n+K)}|D)$ через $(P|B)$. Если какая-либо строка матрицы P содержит не целые элементы, то умножить строку на такое наименьшее натуральное число, чтобы все элементы стали целыми. Рассмотреть все такие строки матрицы P, что среди первых n элементов есть ненулевые и соответствующие строки матрицы B имеют только неотрицательные элементы. Эти строки матрицы P порождают t-функции с наибольшим возможным числом линейно-независимых векторов $\widetilde{\tau}$.

Заметим, что в рассматриваемом случае алгоритмов с аффинными зависимостями, в отличие от случая алгоритмов с однородными зависимостями, может не существовать n независимых таймирований с таймирующими функциями вида (2.6).

Рассмотрим пример. Применим алгоритм 2.4 для получения таймирований следующего аффинного гнезда тесно вложенных циклов:

do $i = 1, N$
  do $j = 1, N$
    do $k = 1, N$
      $S_1:\ a(i,j,k) = a(i,j,k-1) + b(i,j-1,k) + b(i,j-1,k+i)$
      $S_2:\ b(i,j,k) = a(i-1,j,k) + b(i,j,k-1)$
    enddo
  enddo
enddo

Зависимости описываются функциями вида

$$\overline{\Phi}_{\alpha,\beta}(J) = F^{-1}_{l,\alpha,p}F_{l,\beta,q}J + F^{-1}_{l,\alpha,p}(f^{(l,\beta,q)} - f^{(l,\alpha,p)})$$

(см. подраздел 1.1.3): $\overline{\Phi}_{1,1}(i,j,k) = \overline{\Phi}_{2,2}(i,j,k) = E^{(3)}(i\ j\ k)^{\mathrm{T}} - (0\ 0\ 1)^{\mathrm{T}}$, $\overline{\Phi}_{1,2}(i,j,k) = E^{(3)}(i\ j\ k)^{\mathrm{T}} - (1\ 0\ 0)^{\mathrm{T}}$, $\overline{\Phi}_{(2,1)_1}(i,j,k) = E^{(3)}(i\ j\ k)^{\mathrm{T}} - (0\ 1\ 0)^{\mathrm{T}}$, $\overline{\Phi}_{(2,1)_2}(i,j,k) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}(i\ j\ k)^{\mathrm{T}} - (0\ 1\ 0)^{\mathrm{T}}$, $V_{(2,1)_2} = \{(i,j,k) \in \mathbf{Z}^3 |\ 1 \leqslant i \leqslant N,\ 1 \leqslant j \leqslant N,\ 1 \leqslant j-1 \leqslant N,\ 1 \leqslant k \leqslant N,\ 1 \leqslant i+k \leqslant N\} = \{(i,j,k) \in \mathbf{Z}^3 |\ 1 \leqslant i \leqslant N-1,\ 2 \leqslant j \leqslant N,\ 1 \leqslant k \leqslant N-i\}$. Изобразим развернутый граф

зависимостей (рис. 2.7, $N = 3$); зависимости, задаваемые функциями $\overline{\Phi}_{(2,1)_2}(i,j,k)$ (или, что то же самое, векторами $(0,1,-i)$), не изображены.

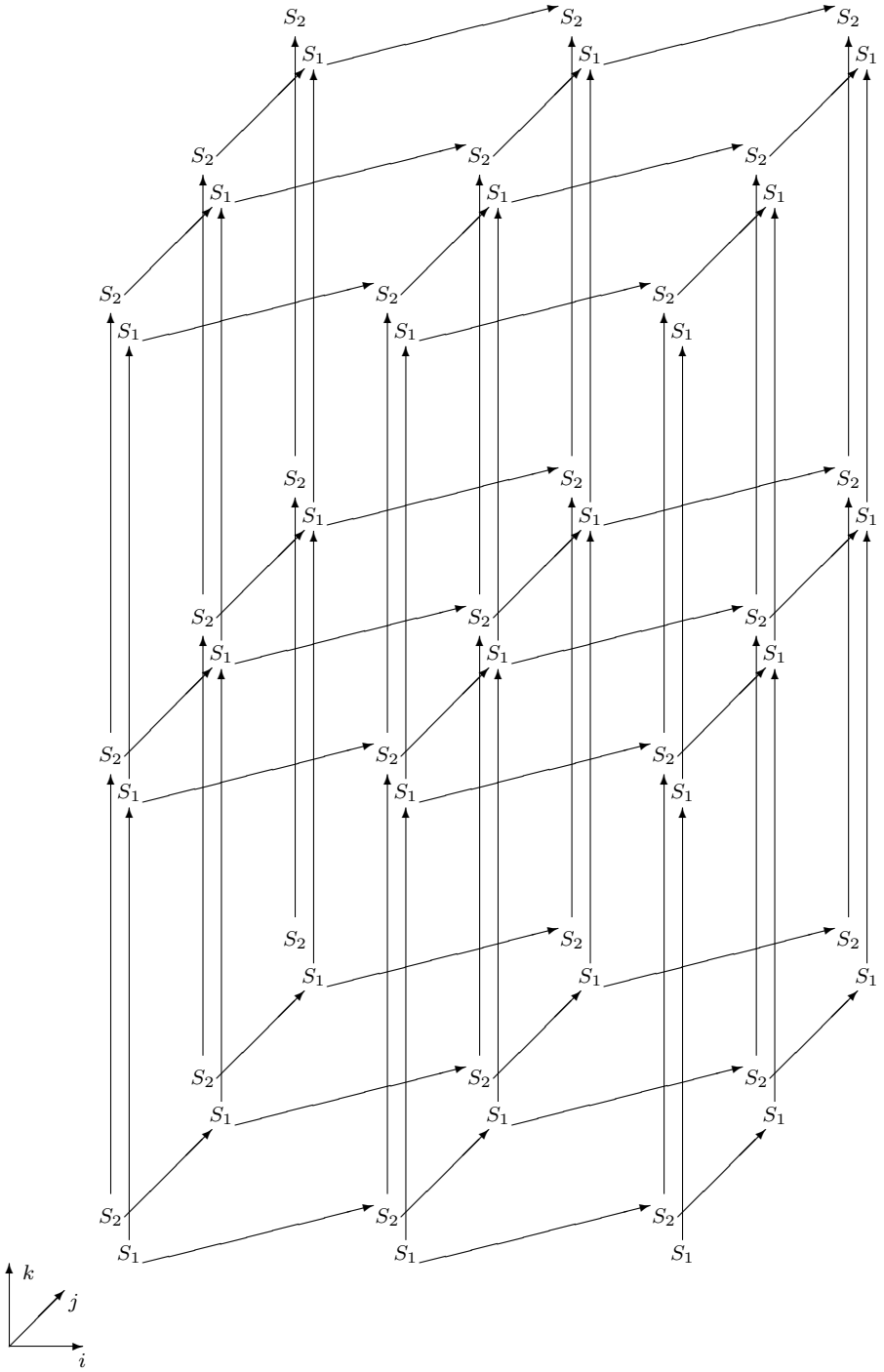

Рис. 2.7

Далее, $t^{(1)}(i,j,k) = (\tau_1,\tau_2,\tau_3)(i,j,k) + a_1$, $t^{(2)}(i,j,k) = (\tau_1,\tau_2,\tau_3)(i,j,k) + a_2$, $\widetilde{\tau} = (\tau_1,\tau_2,\tau_3,a_1,a_2)$, $\widetilde{\Phi}_{1,1} = \widetilde{\Phi}_{1,2} = \widetilde{\Phi}_{(2,1)_1} = \widetilde{\Phi}_{2,2} = 0$, $\widetilde{\Phi}_{(2,1)_2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, $\widetilde{\Phi}_{1,1}p^{(1,1)} + \widetilde{\varphi}^{(1,1)} = \widetilde{\varphi}^{(1,1)} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$,

$\widetilde{\Phi}_{2,2}p^{(2,2)} + \widetilde{\varphi}^{(2,2)} = \widetilde{\varphi}^{(2,2)} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$, $\widetilde{\Phi}_{1,2}p^{(1,2)} + \widetilde{\varphi}^{(1,2)} = \widetilde{\varphi}^{(1,2)} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ -1 \\ 1 \end{pmatrix}$,

$\widetilde{\Phi}_{(2,1)_1}p^{(2,1)_1} + \widetilde{\varphi}^{(2,1)_1} = \widetilde{\varphi}^{(2,1)_1} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ -1 \end{pmatrix}$, $\widetilde{\varphi}^{(2,1)_2} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ -1 \end{pmatrix}$, $p^{(2,1)_2} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$,

$\widetilde{\Phi}_{(2,1)_2}p^{(2,1)_2} + \widetilde{\varphi}^{(2,1)_2} = \begin{pmatrix} 0 \\ 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}$, $D = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & -1 & -1 \\ 0 & -1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 & 0 \end{pmatrix}$.

При формировании матрицы D использовались условия сохранения зависимостей (2.12).

На шаге 1 алгоритма 2.4, совершив преобразования $R_5 + R_4$ (прибавление к пятой строке четвертой строки), $R_4 + R_1$, $R_4 - R_2$, получим матрицу $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$. Матрица $Q = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ -1 & -1 \end{pmatrix}$ содержит два отрицательных элемента, один из которых можно обнулить преобразованием $R_3 + R_2$ (шаг 2). Получим матрицу, требуемую на шаге 3. После перестановки первого и четвертого столбцов получим на шаге 4 матрицу $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}$, $Q_2 = (1,-1)$.

Далее следует перейти ко второй рекурсии алгоритма применительно к матрице Q_2. Матрица Q_2 уже имеет вид, требуемый на шаге 1, $Q = (-1)$. Так как матрица Q состоит только из отрицательных

элементов, то сразу перейдем к шагу 7. После преобразований $R_5 + R_4$, $R_4 + R_1$, $R_4 - R_2$, $R_3 + R_2$ матрица $(E^{(5)}|D)$ переходит в матрицу $(P|B) = \left(\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & -1 \\ 1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$. Первая, вторая и четвертая строки матрицы P порождают векторы параметров невырожденных t-функций, причем четвертая строка приводит к расщепляющей функции.

2.5 Генерация кода

Пусть имеется гнездо вложеных (не обязательно тесно) циклов. Пусть к гнезду циклов применено аффинное преобразование, задаваемое векторными t-функции вида (1.19):

$$\begin{array}{c} t_\xi^{(\beta)}(J) = \tau_{1:n_\beta}^{(\xi)} J + a_{\beta,\xi}, \\ J \in V_\beta \subset \mathbf{Z}^{n_\beta}, \quad \tau_{1:n_\beta}^{(\xi)} = (\tau_1^{(\xi)}, \ldots, \tau_{n_\beta}^{(\xi)}) \in \mathbf{Z}^{n_\beta}, \quad a_{\beta,\xi} \in \mathbf{Z}, \\ 1 \leqslant \beta \leqslant K, \quad 1 \leqslant \xi \leqslant n. \end{array} \tag{2.18}$$

В матрично-векторной форме записи равенства (2.18) имеют вид

$$\bar{t}^{(\beta)}(J) = T^{(\beta)} J + a^{(\beta)}, \quad J \in V_\beta \subset \mathbf{Z}^{n_\beta}, \quad 1 \leqslant \beta \leqslant K,$$

где $T^{(\beta)}$ — матрица, строки которой составлены из векторов $\tau_{1:n_\beta}^{(1)}, \ldots, \tau_{1:n_\beta}^{(n)}$, J — вектор-столбец, $a^{(\beta)} = \begin{pmatrix} a_{\beta,1} \\ \cdots \\ a_{\beta,n} \end{pmatrix}$.

Записать результат аффинного преобразования можно с помощью следующей процедуры.

Алгоритм 2.5 (генерация кода преобразованного гнезда циклов).

1. Для каждого β рассмотреть аффинное преобразование $J' = T^{(\beta)} J + a^{(\beta)}$, $J \in V_\beta$. Если $n_\beta = n$, то выразить параметры исходного гнезда циклов:

$$J = \left(T^{(\beta)}\right)^{-1} (J' - a^{(\beta)}). \tag{2.19}$$

Если $n_\beta < n$, то составить из n_β строк матрицы $T^{(\beta)}$ невырожденную матрицу $T_{n_\beta}^{(\beta)}$, затем сложением, вычитанием и перестановкой

строк привести преобразование $J' = T^{(\beta)}J + a^{(\beta)}$ *к виду* $\begin{pmatrix} J'_{n_\beta} \\ J'_0 \end{pmatrix} = \begin{pmatrix} T^{(\beta)}_{n_\beta} \\ 0 \end{pmatrix} J + \begin{pmatrix} a^{(\beta)}_{n_\beta} \\ a^{(\beta)}_0 \end{pmatrix}$ *и получить представление*

$$J = \left(T^{(\beta)}_{n_\beta}\right)^{-1} (J'_{n_\beta} - a^{(\beta)}_{n_\beta}),\ J'_0 = a^{(\beta)}_0. \tag{2.20}$$

2. Подставить в неравенства, задающие многогранники V_β, *параметры гнезда циклов* J, *представленные в виде (2.19) или (2.20). Для каждого оператора* S_β *выразить границы изменения параметров преобразованного гнезда циклов:*

$$l_1 \leqslant J'_1 \leqslant u_1,$$
$$l_2(J'_1) \leqslant J'_2 \leqslant u_2(J'_1),$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$l_n(J'_1, \cdots, J'_{n-1}) \leqslant J'_n \leqslant u_n(J'_1, \cdots, J'_{n-1}).$$

Эти неравенства задают многогранники, определяющие преобразованные индексные области оператора S_β.

3. Записать код преобразованного гнезда циклов. Если не все векторы $a^{(\beta)}$ *являются нулевыми, то при записи кода использовать операторы if. Если требуется, избавиться от операторов if посредством расщепления пространства итераций и вынесения за пределы цикла нескольких первых или последних итераций. На одном уровне вложенности операторы упорядочиваются в соответствии с внутриитерационными зависимостями.*

В алгоритме 2.5 предполагается, что матрицы $T^{(\beta)}_{n_\beta}$ ($T^{(\beta)}$, если $n_\beta = n$) являются унимодулярными, т. е. матрицами с единичными по модулю определителями.

На шаге 2 для определения границ изменения параметра J'_i можно использовать процесс исключения параметров $J'_{i+1}, \cdots, J'_n$. Теоретическим обоснованием процесса является следующее утверждение.

Теорема Фурье-Моткина. *Пусть имеется система неравенств с неизвестными* $y_1, \cdots, y_k$. *Тогда проекцию этой системы на плоскость* $y_1 = 0$ *задает система неравенств, полученная следующим образом:*

— взять все пары неравенств, с коэффициентами при y_1 противоположных знаков и для каждой пары получить новое неравенство с исключенным y_1;

— добавить из исходной системы неравенств все неравенства, не содержащие y_1.

Для получения границ изменения параметра J'_i можно рассмотреть систему неравенств, содержащих не только параметры $J'_1, \cdots, J'_{i-1}$, последовательно исключить $J'_{i+1}, \cdots, J'_n$ из неравенств, содержащих J'_i, задав тем самым проекцию этих неравенств на пересечение плоскостей $J'_{i+1} = 0, \cdots, J'_n = 0$, из которой и определить границы изменения параметра J'_i.

Пример 4. Пусть A — левая треугольная матрица порядка N с диагональными элементами, равными единице. Рассмотрим алгоритм решения системы линейных алгебраических уравнений $Ax = b$ методом обратной подстановки:

$$\begin{array}{l} S_1: \ x(1) = b(1) \\ \text{do} \ \ i = 2, N \\ \quad S_2: \ x(i) = b(i) \\ \quad \text{do} \ \ j = 1, i-1 \\ \quad\quad S_3: \ x(i) = x(i) - a(i,j)x(j) \\ \quad \text{enddo} \\ \text{enddo} \end{array} \tag{2.21}$$

Гнездо циклов содержит три оператора S_1, S_2, S_3 и элементы трех массивов x, b, a; $n_1 = 0$, $n_2 = 1$, $n_3 = 2$, $\nu_1 = 1$, $\nu_2 = 1$, $\nu_3 = 2$, $V_1 = \{(1)\}$, $V_2 = \{(i) \in \mathbf{Z} |\ 2 \leqslant i \leqslant N\}$, $V_3 = \{(i,j) \in \mathbf{Z}^2 |\ 2 \leqslant i \leqslant N, 1 \leqslant j \leqslant i-1\}$, $W_1 = W_2 = \{(i) \in \mathbf{Z} |\ 1 \leqslant i \leqslant N\}$, $W_3 = V_3$; $\overline{F}_{1,1,1}(1) = 1$, $\overline{F}_{1,2,1}(i) = E^{(1)}(i)$, $\overline{F}_{1,3,1}(i,j) = \overline{F}_{1,3,2}(i,j) = (1\ 0)(i\ j)^{\mathrm{T}}$, $\overline{F}_{1,3,3}(i,j) = (0\ 1)(i\ j)^{\mathrm{T}}$, $\overline{F}_{2,1,1}(1) = 1$, $\overline{F}_{2,2,1}(i) = E^{(1)}(i)$, $\overline{F}_{3,3,1}(i,j) = E^{(2)}(i\ j)^{\mathrm{T}}$.

Можно показать, что зависимости для гнезда циклов (2.21) описываются функциями $\overline{\Phi}_{1,3}(i,1) = 1$, $(i,1) \in V_{1,3} = \{(i,1) \in \mathbf{Z}^2 |\ 2 \leqslant i \leqslant N\}$, $\overline{\Phi}_{2,3}(i,1) = (1\ 0)(i\ 1)^{\mathrm{T}}$, $(i,1) \in V_{2,3} = V_{1,3}$, $\overline{\Phi}_{(3,3)_1}(i,j) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} (i\ j)^{\mathrm{T}} - (0\ 1)^{\mathrm{T}}$, $(i,j) \in V_{(3,3)_1} = \{(i,j) \in \mathbf{Z}^2 |\ 3 \leqslant i \leqslant N,\ 2 \leqslant j \leqslant i-1\}$, $\overline{\Phi}_{(3,3)_2}(i,j) = E^{(2)}(i\ j)^{\mathrm{T}} - (0\ 1)^{\mathrm{T}}$, $(i,j) \in V_{(3,3)_2} = V_{(3,3)_1}$. Заметим, что $(3,3)_1 = (3,3)_{1,1,3}$ (зависимость

порождается первым массивом, т. е. массивом x, первым и третьим его вхождениями в оператор), $(3,3)_2 = (3,3)_{1,1,2}$ (зависимость порождается первым массивом, первым и вторым его вхождениями в оператор, если зависимость считать истинной). Изобразим развернутый граф зависимостей (рис. 2.8, $N = 5$), вершины помечены операциями и необходимыми для выполнения операций элементами массивов).

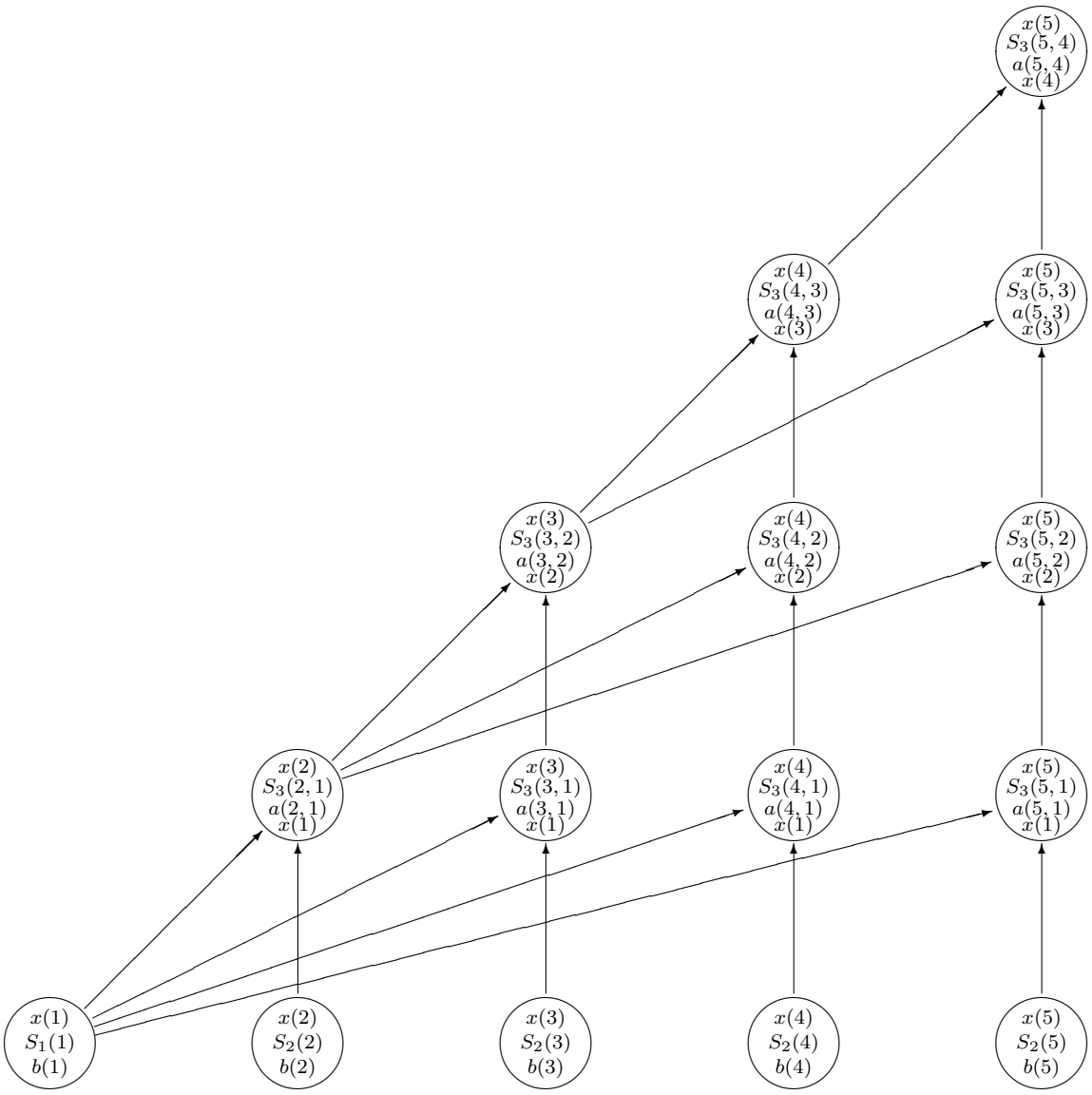

Рис. 2.8

Имеем: $t^{(1)}(1) = a_1$, $t^{(2)}(i) = \tau_1 i + a_2$, $t^{(3)}(i,j) = (\tau_1, \tau_2)(i,j) + a_3$, $\widetilde{\tau} = (\tau_1, \tau_2, a_1, a_2, a_3)$,

$$\widetilde{\Phi}_{1,3} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad \widetilde{\varphi}^{(1,3)} = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 1 \end{pmatrix} \text{ (учтено, что } n_\alpha = 0),$$

$p^{(1,3)} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$,

$$\widetilde{\Phi}_{2,3} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad \widetilde{\varphi}^{(2,3)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 1 \end{pmatrix}, \quad p^{(2,3)} = \begin{pmatrix} 2 \\ 1 \end{pmatrix},$$

$$\widetilde{\Phi}_{(3,3)_1} = \begin{pmatrix} 1 & -1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad \widetilde{\varphi}^{(3,3)_1} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad p^{(3,3)_1} = \begin{pmatrix} 3 \\ 2 \end{pmatrix},$$

$$\widetilde{\Phi}_{(3,3)_2} = 0, \quad \widetilde{\varphi}^{(3,3)_2} = \widetilde{\varphi}^{(3,3)_1}, \quad p^{(3,3)_2} = \begin{pmatrix} 3 \\ 2 \end{pmatrix},$$

$$\widetilde{\Phi}_{1,3} p^{(1,3)} + \widetilde{\varphi}^{(1,3)} = \begin{pmatrix} 2 \\ 1 \\ -1 \\ 0 \\ 1 \end{pmatrix}, \quad \widetilde{\Phi}_{2,3} p^{(2,3)} + \widetilde{\varphi}^{(2,3)} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ -1 \\ 1 \end{pmatrix},$$

$$\widetilde{\Phi}_{(3,3)_1} p^{(3,3)_1} + \widetilde{\varphi}^{(3,3)_1} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \widetilde{\Phi}_{(3,3)_2} p^{(3,3)_2} + \widetilde{\varphi}^{(3,3)_2} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

При формировании матрицы D учтем, что столбцы в D_2, соответствующие матрице $\widetilde{\Phi}_{(3,3)_1}$, следует составлять согласно теореме 2.3 (область $V_{(3,3)_1}$ имеет "треугольный" вид): $D = \begin{pmatrix} 2 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$. Совершив над матрицей $(E^{(5)}|D)$ строчные преобразования $R_1 + 2R_3$ (прибавление к первой строке удвоенной третьей строки), $R_2 + R_3$, $R_5 + R_3$, $-R_3$ (изменение знака элементов третьей строки), $R_5 + R_4$, $R_2 + R_4$, $-R_4$,

получим $\left(\begin{array}{ccccc|ccccc} 1 & 0 & 2 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$. Из теоремы 2.4 следует, что векторы $\widetilde{\tau} = (1,0,2,0,0)$ и $\widetilde{\tau} = (0,1,1,1,0)$ являются векторами параметров невырожденных t-функций.

Рассмотрим преобразование циклов, определяемое полученными t-функциями $t_1^{(1)}(1) = 1$, $t_1^{(2)}(i) = 1$, $t_1^{(3)}(i,j) = j$, $t_2^{(1)}(1) = 2$, $t_2^{(2)}(i) = i$, $t_2^{(3)}(i,j) = i$, $2 \leqslant i \leqslant N,\ 1 \leqslant j \leqslant i-1$.

Для генерации кода применим алгоритм 2.5.

Шаг 1. Для $\beta = 1:\ i' = 1,\ j' = 2$. Для $\beta = 2:\ i' = 1,\ j' = i;\ i = j'$. Для $\beta = 3:\ i' = j,\ j' = i;\ \ i = j',\ j = i'$.

Шаг 2. Для оператора $S_1:\ \ i' = 1,\ j' = 2$. Для оператора $S_2:$ $i' = 1,\ 2 \leqslant j' \leqslant N$.

Для оператора $S_3:\quad 2 \leqslant j' \leqslant N,\ 1 \leqslant i' \leqslant j' - 1$. Чтобы определить границы изменения параметра i', исключим в системе неравенств $j' \geqslant 2,\ N \geqslant j',\ i' \geqslant 1,\ j'-1 \geqslant i'$ параметр j' из неравенств, содержащих i'. Для этого сложим неравенства $N - j' \geqslant 0,\ j' - i' - 1 \geqslant 0$; получим $N - i' - 1 \geqslant 0$. Из неравенств $i' \geqslant 1,\ i' \leqslant N - 1$ получим границы изменения параметра i': $1 \leqslant i' \leqslant N - 1$. Из оставшихся неравенств $j' \geqslant 2,\ j' \leqslant N,\ j' \geqslant i' + 1$ получим границы изменения параметра j': $\max(2, i' + 1) \leqslant j' \leqslant N$.

Гнездо циклов (2.21) после преобразования примет вид

```
do  i' = 1, N − 1
   do  j' = 2, N
      if  i' = 1, j' = 2  then
         S1 :  x(1) = b(1)
      endif
      if  i' = 1,   then
         S2 :  x(j') = b(j')
      endif
      if  j' ⩾ i' + 1  then
         S3 :  x(j') = x(j') − a(j', i')x(i')
      endif
   enddo
enddo
```

После избавления от операторов if окончательно получим

$S_1 :\ x(1) = b(1)$
do $\ j' = 2, N$
  $S_2 :\ x(j') = b(j')$
  $S_3' :\ x(j') = x(j') - a(j', 1)x(1)$
enddo
do $\ i' = 2, N - 1$
  do $\ j' = i' + 1, N$
    $S_3 :\ x(j') = x(j') - a(j', i')x(i')$
  enddo
enddo

□

Глава 3

РАСПРЕДЕЛЕНИЕ ОПЕРАЦИЙ И ДАННЫХ МЕЖДУ ПРОЦЕССОРАМИ. ЛОКАЛИЗАЦИЯ ДАННЫХ

В этой главе рассматриваются методы отображения операций и данных алгоритмов, задаваемых гнездами циклов, в пространство виртуальных процессоров. Основная цель — получить параллельные версии алгоритмов и минимизировать коммутационные затраты. Рассматриваются также некоторые способы оптимизации использования иерархической памяти процессоров, разбиения вычислений для реализации на целевом параллельном компьютере.

3.1 Пространственно-временное отображение. Обобщенный конвейерный параллелизм

Пусть алгоритм задан гнездом вложенных циклов с глубиной вложенности n. Под задачей пространственно-временного отображения понимается задача распараллеливания алгоритмов для реализации на виртуальных процессорах, вложенных в пространство меньшей, чем n, размерности. Рассматриваемые в этом разделе способы получения пространственно-временных отображений позволяют преобразовать исходное гнездо циклов в гнезда циклов, записываемые единым образом для каждого из виртуальных процессоров.

Пусть имеется n-мерное таймирование $\bar{t}$. Будем считать, что r первых координат таймирования $t_\xi = (t_\xi^{(1)}, \ldots, t_\xi^{(K)})$, $1 \leqslant \beta \leqslant K$, $1 \leqslant \xi \leqslant r < n$, задают пространственное отображение операций алгоритма в r-мерное пространство виртуальных процессоров, а оставшиеся $n - r$ координат — упорядочение (выполнения в лексикографическом порядке) вычислений, выполняемых процессорами. Таким образом, значения параметров r внешних циклов преобразованного с

помощью таймирования $\bar{t}$ гнезда циклов определяют координаты процессора, а $n-r$ внутренних циклов — итерации, которые на нем должны выполняться.

Поставим в соответствие итерациям преобразованного гнезда циклов точки (не обязательно все) n-мерного параллелепипеда $\{(t_1,\ldots,t_n) \in \mathbf{Z}^n \mid m_\xi \leqslant t_\xi \leqslant M_\xi,\ 1 \leqslant \xi \leqslant n\}$, где $m_\xi = \min\limits_{1\leqslant\beta\leqslant K,\ J\in V_\beta} t_\xi^{(\beta)}(J)$, $M_\xi = \max\limits_{1\leqslant\beta\leqslant K,\ J\in V_\beta} t_\xi^{(\beta)}(J)$. Положим $\overline{M}_\xi = M_\xi - m_\xi + 1$, $\overline{M}_\xi > 1$; $\overline{M}_\xi$ — числа, характеризующие количество итераций цикла уровня ξ.

Теорема 3.1 *Пусть первые d компонент многомерного таймирования $\bar{t}$ являются таймирующими функциями, а $(d+1)$-я — не является, $d \geqslant r$. Тогда после преобразования гнезда циклов, определяемого векторной функцией $\bar{t}$, $\overline{M}_1 \times \cdots \times \overline{M}_r$ виртуальных процессоров могут реализовать алгоритм за $\sum\limits_{\xi=1}^{r} \sum\limits_{p_\xi=m_\xi+1}^{M_\xi} k_\xi^{(p_\xi)} \prod\limits_{i=d+2}^{n} \overline{M}_i + \prod\limits_{i=r+1}^{n} \overline{M}_i$ параллельных итерационных шагов, где $k_\xi^{(p_\xi)}$ — некоторые константы, $1 \leqslant k_\xi^{(p_\xi)} \leqslant \overline{M}_{d+1}$, $\prod\limits_{i=m}^{n} \overline{M}_i = 1$ при $m > n$, $k_\xi^{(p_\xi)} = 1$ при $d = n$.*

Доказательство. Зафиксируем ξ, $1 \leqslant \xi \leqslant r$. Так как компонента с номером ξ многомерного таймирования является таймированием (ввиду $\xi \leqslant d$), то алгоритм можно разбить на части, отнеся к одной части операции алгоритма с одним и тем же значением функций $t_\xi^{(1)},\ldots,t_\xi^{(K)}$; эти части алгоритма можно выполнять последовательно друг за другом в порядке возрастания значений функций (см. раздел 1.3). Поэтому любую операцию из части с некоторым номером $p_\xi + 1$, не зависящую от других операций из этой части, можно выполнять сразу после выполнения всех тех операций из частей с номерами меньшими или равными p_ξ, от которых зависит эта операция.

Если все виртуальные процессоры выполняют итерации преобразованного гнезда циклов независимо друг от друга, то на выполнение всех итераций потребуется $\overline{M}_{r+1} \times \cdots \times \overline{M}_n$ параллельных итераций. Пусть в преобразованном гнезде циклов существует зависимость $S_\alpha(p_1,\ldots,p_r,I_{r+1},\ldots,I_n) \to S_\beta(q_1,\ldots,q_r,J_{r+1},\ldots,J_n)$. Так как первые d компонент многомерного таймирования являются таймирующими

функциями, а $(d+1)$-я — не является, то выполняется $q_1 \geqslant p_1, \ldots, q_r \geqslant p_r$, $J_{r+1} \geqslant I_{r+1}, \ldots, J_d \geqslant I_d$, но возможно и $I_{d+1} > J_{d+1}$. Следовательно, наличие зависимости может вызвать задержку в вычислениях виртуального процессора с координатами $(q_1, \ldots, q_r)$ по сравнению с процессором с координатами $(p_1, \ldots, p_r)$. Допустим худшую возможность с точки зрения минимальности числа параллельных итераций, требуемых для реализации алгоритма: при увеличении любого p_ξ на единицу имеет место $J_{d+1} = m_{d+1}, \ldots, J_n = m_n$, так что задержка в вычислениях равна числу итерационных шагов между итерациями $(m_{d+1}, \ldots, m_n)$ и $(I_{d+1}, \ldots, I_n)$ и равна $k_\xi^{(p_\xi)} \prod\limits_{i=d+2}^{n} \overline{M}_i$ итераций; если $d = n-1$, то задержка в вычислениях равна $k_\xi^{(p_\xi)}$; если $d = n$, то задержка равна 1. Суммарная задержка по всем p_ξ и всем ξ может для процессора с координатами $(M_1, \ldots, M_r)$ достичь $\sum\limits_{i=1}^{r} \sum\limits_{p_\xi = m_\xi + 1}^{M_\xi} k_\xi^{(p_\xi)} \prod\limits_{i=d+2}^{n} \overline{M}_i$ итераций. □

Следствие 3.1 *Чем больше первых компонент многомерного таймирования являются таймирующими функциями, тем быстрее, вообще говоря, может быть реализован алгоритм виртуальными процессорами. Если число итераций циклов всех уровней одинаково и равно* $\overline{M}$, *то в худшем случае* $(k_\xi^{(p_\xi)} = \overline{M})$ *потребуется* $r\overline{M}^{n-d+1} - r\overline{M}^{n-d} + \overline{M}^{n-r}$ *параллельных итераций. Если первые* r *компонент многомерного таймирования являются расщепляющими функциями, то потребуется* $\overline{M}^{n-r}$ *итераций.*

Если хотя бы $r+1$ первых компонент многомерного таймирования $\bar{t}$ являются одномерными t-функциями, то можно получить максимально возможную степень параллелизма без явного указания параллельных циклов: $O(\overline{M}^{n-r})$ виртуальных процессоров могут реализовать $O(\overline{M}^{n})$ операций алгоритма за $O(\overline{M}^{r})$ параллельных итерационных шагов. Кроме того, в этом случае возможны регулярный код, упрощенная синхронизация, применение блокинга. Обмен данными между любыми двумя процессорами сводится к передаче данных от одного и того же процессора другому процессору. Получаемый способ обработки данных можно рассматривать как обобщение классической конвейерной обработки.

Степень параллелизма можно характеризовать числом независимых таймирующих функций; наличие двух и более функций позволяет

(если индексные области не являются вырожденными) организовать параллельные вычисления.

Пример 2 (продолжение). В разделе 2.4 получены таймирующие функции $t_1^{(1)}(i) = i$, $t_1^{(2)}(i,j) = i$, $t_2^{(1)}(i) = 1$, $t_2^{(2)}(i,j) = j$. Именно этим t-функциям соответствует запись алгоритма умножения матрицы на вектор в виде (1.2). Приняв функции $t_1^{(1)}(i) = i$, $t_1^{(2)}(i,j) = i$ за функции размещения, можно записать код единым образом для каждого из N процессоров. Пространственную координату процессора обозначим через p. Переменную цикла, соответствующую таймированию, обозначим через t.

$$\begin{array}{l} \text{if } \ 1 \leqslant p \leqslant N \ \text{ then} \\ \quad S_1: \ c(p) = 0 \\ \quad \text{do } \ t = 1, N \\ \qquad S_2: \ c(p) = c(p) + a(p,t)b(t) \\ \quad \text{enddo} \end{array} \tag{3.1}$$

На практике для записи кода необходимо еще установить режим обмена данными и определить синхронизацию. □

3.2 Функции распределения массивов между процессорами и итерациями

Для реализации алгоритма на параллельном компьютере с распределенной памятью требуется не только распределить операции алгоритма между процессорами и итерациями, обеспечить возможность параллельного выполнения операций, но и распределить данные, организовать обмен данными. Производительность многопроцессорной вычислительной системы с распределенной памятью в значительной степени зависит от оптимальности решения задачи распределения данных. В этом разделе вводятся и исследуются функции, задающие распределение элементов массивов данных, согласованное с заданным распределением операций.

Пусть функции $t_\xi^{(\beta)}$, $1 \leqslant \beta \leqslant K$, $1 \leqslant \xi \leqslant r$, вида (1.10) задают отображение операций алгоритма в r-мерное пространство виртуальных процессоров, а функции $t_\xi^{(\beta)}$, $r+1 \leqslant \xi \leqslant n$, задают итерации, выполняемые процессорами.

Обозначим: $T_s^{(\beta)}$ — матрица, строки которой составлены из век-

торов $\tau^{(\beta,1)},\dots,\tau^{(\beta,r)}$; $T_t^{(\beta)}$ — матрица, строки которой составлены из векторов $\tau^{(\beta,r+1)},\dots,\tau^{(\beta,n)}$; $\rho_{l,\beta,q} = \mathrm{rank}F_{l,\beta,q}$, $\rho^s_{l,\beta,q} = \mathrm{rank}\begin{pmatrix}F_{l,\beta,q}\\ T_s^{(\beta)}\end{pmatrix}$, $\rho^t_{l,\beta,q} = \mathrm{rank}\begin{pmatrix}F_{l,\beta,q}\\ T_t^{(\beta)}\end{pmatrix}$.

Рассмотрим базис пространства $\mathbf{Z}^{n_\beta}$ с базисными векторами $u_i^\perp$, $1 \leqslant i \leqslant \rho_{l,\beta,q}$, u_i, $1 \leqslant i \leqslant n_\beta - \rho_{l,\beta,q}$, где u_i — базисные векторы подпространства $\ker F_{l,\beta,q}$; если $\rho_{l,\beta,q} = n_\beta$, то векторы u_i отсутствуют. Матрицу, столбцы которой составлены из векторов $u_i^\perp$, обозначим $U^\perp_{l,\beta,q}$. Обозначим через u_i^s, $1 \leqslant i \leqslant n_\beta - \rho^s_{l,\beta,q}$, базисные векторы пересечения подпространства $\ker\begin{pmatrix}F_{l,\beta,q}\\ T_s^{(\beta)}\end{pmatrix}$ и $\mathbf{Z}^{n_\beta}$, через u_i^t, $1 \leqslant i \leqslant n_\beta - \rho^t_{l,\beta,q}$, базисные векторы пересечения подпространства $\ker\begin{pmatrix}F_{l,\beta,q}\\ T_t^{(\beta)}\end{pmatrix}$ и $\mathbf{Z}^{n_\beta}$; векторы u_i^s, u_i^t могут отсутствовать. Рассмотрим базисы пространства $\mathbf{Z}^{n_\beta}$ с базисными векторами $u_i^{\perp,s}$, $1 \leqslant i \leqslant \rho^s_{l,\beta,q}$, u_i^s, $1 \leqslant i \leqslant n_\beta - \rho^s_{l,\beta,q}$, и $u_i^{\perp,t}$, $1 \leqslant i \leqslant \rho^t_{l,\beta,q}$, u_i^t, $1 \leqslant i \leqslant n_\beta - \rho^t_{l,\beta,q}$. Матрицу, столбцы которой составлены из векторов $u_i^{\perp,s}$, обозначим $U^{\perp,s}_{l,\beta,q}$; матрицу, столбцы которой составлены из векторов $u_i^{\perp,t}$, обозначим $U^{\perp,t}_{l,\beta,q}$.

Пусть F — элемент множества W_l. Обозначим через $V_{l,\beta,q}(F)$ множество итераций исходного гнезда циклов, на которых на q-м вхождении массива a_l в оператор S_β используется одно и то же данное $a_l(F)$:

$$V_{l,\beta,q}(F) = \{J \in V_\beta \mid \overline{F}_{l,\beta,q}(J) = F\}.$$

Через $\Lambda_{l,\beta,q}(F)$ обозначим множество, задающее координаты проекции множества $V_{l,\beta,q}(F)$ на линейную оболочку векторов u_i :

$$\Lambda_{l,\beta,q}(F) = \{(\lambda_1,...,\lambda_{n_\beta-\rho_{l,\beta,q}})|\ J = J^\perp + \sum_{i=1}^{n_\beta-\rho_{l,\beta,q}} \lambda_i u_i,\ J \in V_{l,\beta,q}(F)\};$$

в случае $\rho_{l,\beta,q} = n_\beta$ множество $\Lambda_{l,\beta,q}(F)$ пустое.

Вхождением (l,β,q) будем называть q-е вхождение массива a_l в оператор S_β. Область изменения индексов элементов массива $a_l(F)$, связанных с вхождением (l,β,q), будем обозначать $W_{l,\beta,q}$.

Введем в рассмотрение аффинные функции вида

$$d_{\xi}^{(l,\beta,q)}(F) = \eta^{(l,\beta,q,\xi)}F + y_{l,\beta,q,\xi}(F),$$
$$F \in W_{l,\beta,q}, \quad \eta^{(l,\beta,q,\xi)} \in \mathbf{Z}^{\nu_l}, \quad y_{l,\beta,q,\xi}(F) \in \mathbf{Z}, \tag{3.2}$$
$$1 \leqslant l \leqslant L, \quad 1 \leqslant \beta \leqslant K, \quad 1 \leqslant \xi \leqslant n.$$

Будем считать, что функции $\overline{d}_s^{(l,\beta,q)} = (d_1^{(l,\beta,q)}, \ldots, d_r^{(l,\beta,q)})$ задают координаты процессоров, в которых используются элементы массивов, связанные с вхождением (l, β, q), а функции $\overline{d}_t^{(l,\beta,q)} = (d_{r+1}^{(l,\beta,q)}, \ldots, d_n^{(l,\beta,q)})$ задают итерации, на которых эти элементы используются. Отметим, что функции $d_{\xi}^{(l,\beta,q)}$ в общем случае многозначные.

Теорема 3.2 *Элемент массива $a_l(F)$, $F \in W_{l,\beta,q}$, используется виртуальными процессорами $(d_1^{(l,\beta,q)}(F), \ldots, d_r^{(l,\beta,q)}(F))$ на итерациях $(d_{r+1}^{(l,\beta,q)}(F), \ldots, d_n^{(l,\beta,q)}(F))$, где функции $d_{\xi}^{(l,\beta,q)}$ задаются формулами (3.2), в которых $\eta^{(l,\beta,q,\xi)}$ — решение системы уравнений*

$$\eta^{(l,\beta,q,\xi)}F_{l,\beta,q}U_{l,\beta,q}^{\perp} = \tau^{(\beta,\xi)}U_{l,\beta,q}^{\perp}, \tag{3.3}$$

а величины $y_{l,\beta,q,\xi}(F)$ определяются равенствами

$$y_{l,\beta,q,\xi}(F) = a_{\beta,\xi} - \eta^{(l,\beta,q,\xi)}f^{(l,\beta,q)} + \sum_{i=1}^{n_\beta - \rho_{l,\beta,q}} \lambda_i \tau^{(\beta,\xi)}u_i,$$
$$(\lambda_1, \ldots, \lambda_{n_\beta - \rho_{l,\beta,q}}) \in \Lambda_{l,\beta,q}(F). \tag{3.4}$$

Доказательство. Пусть F — произвольный фиксированный элемент множества $W_{l,\beta,q}$. По смыслу задания функций $d_{\xi}^{(l,\beta,q)}$ должно выполняться $t_{\xi}^{(\beta)}(J) - d_{\xi}^{(l,\beta,q)}(\overline{F}_{l,\beta,q}(J)) = 0$ для любого $J \in V_{l,\beta,q}(F)$. Представим вектор J в виде $J = J^{\perp} + J^{(0)}$, где $J^{\perp} = \sum\limits_{i=1}^{\rho_{l,\beta,q}} \lambda_i^{\perp}u_i^{\perp}$, $J^{(0)} = \sum\limits_{i=1}^{n_\beta - \rho_{l,\beta,q}} \lambda_i u_i$. Имеем: $t_{\xi}^{(\beta)}(J) - d_{\xi}^{(l,\beta,q)}(\overline{F}_{l,\beta,q}(J)) =$
$= \tau^{(\beta,\xi)}J + a_{\beta,\xi} - (\eta^{(l,\beta,q,\xi)}\overline{F}_{l,\beta,q}(J) + y_{l,\beta,q,\xi}) = \tau^{(\beta,\xi)}J + a_{\beta,\xi} -$
$- \eta^{(l,\beta,q,\xi)}(F_{l,\beta,q}J + f^{(l,\beta,q)}) - y_{l,\beta,q,\xi} = \tau^{(\beta,\xi)}J^{\perp} + \tau^{(\beta,\xi)}J^{(0)} + a_{\beta,\xi} -$
$- \eta^{(l,\beta,q,\xi)}F_{l,\beta,q}J^{\perp} - \eta^{(l,\beta,q,\xi)}f^{(l,\beta,q)} - y_{l,\beta,q,\xi} = (\tau^{(\beta,\xi)} -$
$- \eta^{(l,\beta,q,\xi)}F_{l,\beta,q}) \sum\limits_{i=1}^{\rho_{l,\beta,q}} \lambda_i^{\perp}u_i^{\perp} + \tau^{(\beta,\xi)} \sum\limits_{i=1}^{n_\beta - \rho_{l,\beta,q}} \lambda_i u_i + a_{\beta,\xi} - \eta^{(l,\beta,q,\xi)}f^{(l,\beta,q)} -$
$- y_{l,\beta,q,\xi} = \sum\limits_{i=1}^{\rho_{l,\beta,q}} \lambda_i^{\perp}(\tau^{(\beta,\xi)}u_i^{\perp} - \eta^{(l,\beta,q,\xi)}F_{l,\beta,q}u_i^{\perp}) + \sum\limits_{i=1}^{n_\beta - \rho_{l,\beta,q}} \lambda_i \tau^{(\beta,\xi)}u_i + a_{\beta,\xi} -$
$- \eta^{(l,\beta,q,\xi)}f^{(l,\beta,q)} - y_{l,\beta,q,\xi}.$

Таким образом, равенство $t_\xi^{(\beta)}(J) - d_\xi^{(l,\beta,q)}(\overline{F}_{l,\beta,q}(J)) = 0$ выполняется, если компоненты каждого вектора $\eta^{(l,\beta,q,\xi)}$ определить как решение системы $\rho_{l,\beta,q}$ уравнений с ν_l неизвестными

$$\eta^{(l,\beta,q,\xi)} F_{l,\beta,q} u_i^\perp = \tau^{(\beta,\xi)} u_i^\perp, \quad 1 \leqslant i \leqslant \rho_{l,\beta,q}, \tag{3.5}$$

а функции $y_{l,\beta,q,\xi}(F)$ задать равенствами (3.4). Осталось заметить, что систему уравнений (3.5) можно записать в виде (3.3), и что для каждого фиксированного F векторы $(\lambda_1, ..., \lambda_{n_\beta - \rho_{l,\beta,q}})$ принадлежат множеству $\Lambda_{l,\beta,q}(F)$. □

Следствие 3.2 *Если $\rho_{l,\beta,q} < \rho^s_{l,\beta,q}$, то каждый элемент массива, связанный с вхождением (l, β, q), используется виртуальными процессорами, коммуникации между которыми можно задать векторами $T_s^{(\beta)} u_i$, $1 \leqslant i \leqslant n_\beta - \rho_{l,\beta,q}$; если $\rho_{l,\beta,q} = \rho^s_{l,\beta,q}$, то элемент массива используется только одним процессором.*

Доказательство. Величины $y_{l,\beta,q,\xi}(F)$, $1 \leqslant \xi \leqslant r$, не зависят от F, если $\rho_{l,\beta,q} = \rho^s_{l,\beta,q}$: в этом случае $\ker \begin{pmatrix} F_{l,\beta,q} \\ T_s^{(\beta)} \end{pmatrix} = \ker F_{l,\beta,q}$, $T_s^{(\beta)} u_i = 0$ для всех u_i, сумма в соотношениях (3.4) равна нулю; в особом частном случае, когда $\rho_{l,\beta,q} = n_\beta$, сумма в соотношениях (3.4) отсутствует, множество $\Lambda_{l,\beta,q}(F)$ пустое.

Осталось показать, что для любого $\eta^{(l,\beta,q,\xi)}$, являющегося решением системы (3.3), при фиксированном F величина $\eta^{(l,\beta,q,\xi)} F - \eta^{(l,\beta,q,\xi)} f^{(l,\beta,q)}$ принимает одно и то же значение.

Любой вектор J, для которого $F_{l,\beta,q} J + f^{(l,\beta,q)} = F$, можно представить в виде $J = J^\perp + J^{(0)}$, где $J^{(0)} \in \ker F_{l,\beta,q}$, $J^\perp$ — какое-либо фиксированное решение системы уравнений $F_{l,\beta,q} J = F - f^{(l,\beta,q)}$, $J^\perp = \sum\limits_{i=1}^{\rho_{l,\beta,q}} \lambda_i^\perp u_i^\perp$. Из доказательства теоремы следует $\eta^{(l,\beta,q,\xi)} F - \eta^{(l,\beta,q,\xi)} f^{(l,\beta,q)} = \eta^{(l,\beta,q,\xi)} (F_{l,\beta,q} J + f^{(l,\beta,q)}) - \eta^{(l,\beta,q,\xi)} f^{(l,\beta,q)} = \eta^{(l,\beta,q,\xi)} F_{l,\beta,q} J = \eta^{(l,\beta,q,\xi)} F_{l,\beta,q} (J^\perp + J^{(0)}) = \eta^{(l,\beta,q,\xi)} F_{l,\beta,q} \sum\limits_{i=1}^{\rho_{l,\beta,q}} \lambda_i^\perp u_i^\perp = \sum\limits_{i=1}^{\rho_{l,\beta,q}} \lambda_i^\perp \eta^{(l,\beta,q,\xi)} F_{l,\beta,q} u_i^\perp = \sum\limits_{i=1}^{\rho_{l,\beta,q}} \lambda_i^\perp \tau^{(\beta,\xi)} u_i^\perp$, что является постоянной величиной при фиксированном $J^\perp$. □

Следствие 3.3 *Если* $\rho_{l,\beta,q} < \rho^t_{l,\beta,q}$, *то каждый элемент массива, связанный с вхождением* (l,β,q), *используется на итерациях, отличающихся линейными комбинациями векторов* $T_t^{(\beta)} u_i$, $1 \leqslant i \leqslant n_\beta - \rho_{l,\beta,q}$; *если* $\rho_{l,\beta,q} = \rho^t_{l,\beta,q}$, *то элемент массива используется только на одной итерации.*

С помощью теоремы 3.2 можно определить распределение входных и выходных данных между процессорами, получить информацию об объеме массивов, используемых в каждом процессоре.

Распределение между виртуальными процессорами входных массивов можно задать с помощью функций $\overline{d}_s^{(l,\beta,q)}(F)$. Для этого элемент $a_l(F)$ входного массива a_l, $F \in W_{l,\beta,q}$, следует распределить в процессор $P_{in}(\overline{d}_s^{(l,\beta,q)}(F))$, координаты которого задаются элементом множества $\overline{d}_s^{(l,\beta,q)}(F)$ с лексикографически наименьшим значением. В этом процессоре $a_l(F)$ будет использоваться не позже, чем в других процессорах: не может быть передачи информации от процессора с большим номером к процессору с меньшим номером.

Результаты вычислений — элементы $a_l(F)$ выходного массива — находятся в процессоре $P_{out}(\overline{d}_s^{(l,\beta,q)}(F))$, координаты которого задаются элементом множества $\overline{d}_s^{(l,\beta,q)}(F)$ с лексикографически наибольшим значением.

Функции $\overline{d}_s^{(l,\beta,q)}(F)$ позволяют также получить информацию об объеме массивов, используемых в каждом процессоре. Для того чтобы определить элементы $a_l(F)$ выходного массива a_l, используемые в процессоре с координатами $(t_1,\ldots,t_r)$, следует найти все элементы F множества $W_{l,\beta,q}$, для которых $\overline{d}_s^{(l,\beta,q)}(F) = (t_1,\ldots,t_r)$.

Пример 4 (продолжение). Пусть $r = 1$, распределение операций гнезда циклов (2.21) между процессорами определяется функциями $t_1^{(1)}(1) = 1$, $t_1^{(2)}(i) = i$, $2 \leqslant i \leqslant N$, $t_1^{(3)}(i,j) = i$, $2 \leqslant i \leqslant N$, $1 \leqslant j \leqslant i-1$; итерации, выполняемые процессорами, зададим функциями $t_2^{(1)}(1) = 1$, $t_2^{(2)}(i) = 1$, $t_2^{(3)}(i,j) = j$. Эти функции получены в разделе 2.5 Найдем распределение элементов массивов данных между процессорами.

Получим функции распределения элементов массива a. Элементы массива a связаны с вхождением в третий оператор, $\tau^{(3,1)} = (1,0)$, $\tau^{(3,2)} = (0,1)$, $F_{3,3,1} = E^{(2)}$, $f^{(3,3,1)} = 0$, $\rho_{3,3,1} = \mathrm{rank}E^{(2)} = 2$, $n_3 =$

$= 2$, $\rho^s_{3,3,1} = 2$, $\rho^t_{3,3,1} = 2$, согласно следствиям 3.2 и 3.3 каждый элемент $a(i,j)$ используется только один раз (одним процессором на одной итерации), u_i отсутствуют, $u_1^\perp = (1,0)$, $u_2^\perp = (0,1)$. Система (3.3): $\eta^{(3,3,1,1)} E^{(2)} E^{(2)} = (1\ 0) E^{(2)}$; отсюда $\eta^{(3,3,1,1)} = (1,0)$. Из соотношений (3.4) получим $y_{3,3,1,1} = 0$. Таким образом, $d_1^{(3,3,1)}(i,j) = i$, процессор $P(i)$ приватизирует i-ю строку матрицы A.

Распределение элементов массивов, связанных со вторым оператором, очевидно: $b(i)$ и $x(i)$ распределяются в процессор $P(i)$.

Найдем теперь распределение элементов массива x, связанных с первым и вторым вхождениями в третий оператор. Достаточно рассмотреть только одно из этих вхождений. Имеем: $F_{1,3,1} = (1\ 0)$, $f^{(1,3,1)} = 0$, $\rho^s_{1,3,1} = \rho_{1,3,1} = \mathrm{rank}(1\ 0) = 1$, $n_3 = 2$, $u_i = (0,1)$, $u_1^\perp = (1,0)$; $\eta^{(1,3,1,1)}(1\ 0)\begin{pmatrix}1\\0\end{pmatrix} = (1\ 0)\begin{pmatrix}1\\0\end{pmatrix}$ (система (3.3)), $\eta^{(1,3,1,1)} = 1$; $y_{1,3,1,1} = 0 - 1 \cdot 0 + \lambda_1(1,0)(0,1) = 0$ (соотношения (3.4)); $d_1^{(1,3,1)}(i) = i$, $x(i)$ используется в процессоре $P(i)$.

Осталось найти распределение элементов массива x, связанных с третьим вхождением в третий оператор. В этом случае $F_{1,3,3} = (0\ 1)$, $f^{(1,3,3)} = 0$, $\rho_{1,3,3} = \mathrm{rank}(0\ 1) = 1$, $\rho^s_{1,3,3} = \mathrm{rank} E^{(2)} = 2$, $n_3 = 2$, $u_i = (1,0)$, $u_1^\perp = (0,1)$; $\eta^{(1,3,3,1)}(0\ 1)\begin{pmatrix}0\\1\end{pmatrix} = (1\ 0)\begin{pmatrix}0\\1\end{pmatrix}$ (система (3.3)), $\eta^{(1,3,3,1)} = 0$; $y_{1,3,3,1} = 0 - 0 + \lambda_1(1,0)(1,0) = \lambda_1$, $\lambda_1 \in \Lambda_{1,3,3}(j)$, j фиксировано. Найдем $\Lambda_{1,3,3}(j) : V_{1,3,3}(j) = \{(i,j) \mid j+1 \leqslant i \leqslant N\}$, $u_1 = (1,0)$, $\Lambda_{1,3,3}(j) = \{i \mid j+1 \leqslant i \leqslant N\}$. Можно сделать вывод, что $d_1^{(1,3,3)}(j) = i, j+1 \leqslant i \leqslant N$, $x(j)$ используется в процессорах $P(j+1), ..., P(N)$.

Таким образом, получено распределение входных массивов A, b и массива результатов вычислений x: $a(i,j)$ и $b(i)$ распределяются в процессор $P(i)$, данное $x(i)$ после вычислений находится в процессоре $P(i)$. В каждом процессоре $P(i)$ используются с первого по i-й элементы массива x. □

Определим итерации, на которых фиксированный элемент массива используется в одном виртуальном процессоре.

Пусть P — произвольный фиксированный элемент множества значений функции $(t_1^{(\beta)}(J), \ldots, t_r^{(\beta)}(J))$, $V^s_\beta(P)$ — множество итераций исходного гнезда циклов, на которых выполняется оператор S_β и кото-

рые реализуются виртуальным процессором P : $V^s_\beta(P) = \{J \in V_\beta \mid (t^{(\beta)}_1(J), \ldots, t^{(\beta)}_r(J)) = P\}$. Рассмотрим множество $V_{l,\beta,q}(F) \cap V^s_\beta(P)$ итераций исходного гнезда циклов, на которых на вхождении (l, β, q) используется одно и то же данное $a_l(F)$ и которые реализуются виртуальным процессором P. Через $\Lambda^s_{l,\beta,q}(F, P)$ обозначим множество, задающее координаты проекции множества $V_{l,\beta,q}(F) \cap V^s_\beta(P)$ на линейную оболочку векторов u^s_i :

$$\Lambda^s_{l,\beta,q}(F,P) = \{(\lambda^s_1, ..., \lambda^s_{n_\beta - \rho^s_{l,\beta,q}}) |\ J = J^{\perp,s} + \\ + \sum_{i=1}^{n_\beta - \rho^s_{l,\beta,q}} \lambda^s_i u^s_i,\ J \in V_{l,\beta,q}(F) \cap V^s_\beta(P)\};$$

в случае $\rho^s_{l,\beta,q} = n_\beta$ множество $\Lambda^s_{l,\beta,q}(F, P)$ пустое.

Теорема 3.3 *Элемент массива $a_l(F)$, $F \in W_{l,\beta,q}$, используется в фиксированном процессоре P на итерациях $(d^{(l,\beta,q)}_{r+1}(F), \ldots, d^{(l,\beta,q)}_n(F))$, где $d^{(l,\beta,q)}_\xi$ задаются формулами (3.2), в которых $\eta^{(l,\beta,q,\xi)}$ — решение системы уравнений*

$$\eta^{(l,\beta,q,\xi)} F_{l,\beta,q} U^{\perp,s}_{l,\beta,q} = \tau^{(\beta,\xi)} U^{\perp,s}_{l,\beta,q}, \tag{3.6}$$

а величины $y_{l,\beta,q,\xi}(F)$ определяются равенствами

$$y_{l,\beta,q,\xi}(F) = a_{\beta,\xi} - \eta^{(l,\beta,q,\xi)} f^{(l,\beta,q)} + \sum_{i=1}^{n_\beta - \rho^s_{l,\beta,q}} \lambda^s_i \tau^{(\beta,\xi)} u^s_i, \\ (\lambda^s_1, ..., \lambda^s_{n_\beta - \rho^s_{l,\beta,q}}) \in \Lambda^s_{l,\beta,q}(F,P). \tag{3.7}$$

Доказательство теоремы, по сути, повторяет доказательство теоремы 3.2, следует лишь зафиксировать F и P.

Следствие 3.4 *Если $\rho^s_{l,\beta,q} < n_\beta$, то в одном виртуальном процессоре каждый элемент массива, связанный с вхождением (l, β, q), используется на итерациях, отличающихся линейными комбинациями векторов $T^{(\beta)}_t u^s_i$, $1 \leqslant i \leqslant n_\beta - \rho^s_{l,\beta,q}$; если $\rho^s_{l,\beta,q} = n_\beta$, то элемент массива используется в одном процессоре только на одной итерации.*

В заключение раздела приведем теорему, в которой определяются процессоры, использующие фиксированный элемент массива на одной итерации.

Пусть вектор T является элементом множества значений функции $(t_{r+1}^{(\beta)}(J),\ldots,t_n^{(\beta)}(J))$. Рассмотрим множество

$$V_\beta^t(T)=\{J\in V_\beta \mid (t_{r+1}^{(\beta)}(J),\ldots,t_n^{(\beta)}(J))=T\}$$

итераций исходного гнезда циклов, на которых выполняется оператор S_β и которые реализуются в пространстве виртуальных процессоров на итерации T, и множество $V_{l,\beta,q}(F)\cap V_\beta^t(T)$ итераций исходного гнезда циклов, на которых на вхождении (l,β,q) используется одно и то же данное $a_l(F)$ и которые реализуются в пространстве виртуальных процессоров на итерации T. Через $\Lambda_{l,\beta,q}^t(F,T)$ обозначим множество, задающее координаты проекции множества $V_{l,\beta,q}(F)\cap V_\beta^t(T)$ на линейную оболочку векторов u_i^t :

$$\Lambda_{l,\beta,q}^t(F,T)=\{(\lambda_1^t,...,\lambda_{n_\beta-\rho_{l,\beta,q}^t}^t)|\ J=J^{\perp,t}+ \\ +\sum_{i=1}^{n_\beta-\rho_{l,\beta,q}^t}\lambda_i^t u_i^t,\ J\in V_{l,\beta,q}(F)\cap V_\beta^t(T)\};$$

в случае $\rho_{l,\beta,q}^t=n_\beta$ множество $\Lambda_{l,\beta,q}^t(F,T)$ пустое.

Теорема 3.4 *Элемент массива $a_l(F)$, $F\in W_{l,\beta,q}$, используется на фиксированной итерации T в процессорах $(d_1^{(l,\beta,q)}(F),\ldots,d_r^{(l,\beta,q)}(F))$, где $d_\xi^{(l,\beta,q)}$ задаются формулами (3.2), в которых $\eta^{(l,\beta,q,\xi)}$ — решение системы уравнений*

$$\eta^{(l,\beta,q,\xi)}F_{l,\beta,q}U_{l,\beta,q}^{\perp,t}=\tau^{(\beta,\xi)}U_{l,\beta,q}^{\perp,t}, \tag{3.8}$$

а величины $y_{l,\beta,q,\xi}(F)$ определяются равенствами

$$y_{l,\beta,q,\xi}(F)=a_{\beta,\xi}-\eta^{(l,\beta,q,\xi)}f^{(l,\beta,q)}+\sum_{i=1}^{n_\beta-\rho_{l,\beta,q}^t}\lambda_i^t\tau^{(\beta,\xi)}u_i^t, \\ (\lambda_1^t,...,\lambda_{n_\beta-\rho_{l,\beta,q}^t}^t)\in\Lambda_{l,\beta,q}^t(F,T). \tag{3.9}$$

Следствие 3.5 *Если $\rho_{l,\beta,q}^t<n_\beta$, то на одной итерации каждый элемент массива, связанный с вхождением (l,β,q), используется виртуальными процессорами, коммуникации между которыми можно задать векторами $T_s^{(\beta)}u_i^t$, $1\leqslant i\leqslant n_\beta-\rho_{l,\beta,q}^t$; если $\rho_{l,\beta,q}^t=n_\beta$, то элемент массива используется на одной итерации только в одном процессоре.*

3.3 Организация обмена данными

На практике даже оптимальное начальное распределение данных не исключает необходимости в обмене данными между процессорами, в локальной памяти которых хранятся данные, и процессорами, в которых эти данные вычисляются или используются как аргументы вычислений. Как следует из предыдущего раздела (следствие 3.2), обмен данными требуется для элементов массива a_l, связанных с q-м вхождением массива в оператор S_β, если имеет место $\rho^s_{l,\beta,q} > \rho_{l,\beta,q}$.

Известно, что на параллельных компьютерах с распределенной памятью структурированные коммуникации, такие как бродкаст (broadcast), разброс (scatter), сборка (gather), редукция (reduction), а также трансляция (translation) данных выполняются значительно быстрее, чем большое количество коммуникаций точка-точка (point-to-point) между процессором, в локальной памяти которого хранится данное, и каждым процессором, который использует или вычисляет это данное. Поэтому желательно выявлять возможность организации таких коммуникаций.

В этом разделе будут сформулированы и исследованы условия, позволяющие в некоторых случаях определять возможность организации наиболее часто используемых быстрых коммуникаций — бродкаста и трансляции данных. Бродкаст (одновременное распространение) — это передача данного группе процессоров, в которых данное одновременно (на одной итерации) используется как аргумент. Трансляция — это передача данного от процессора к процессору в случае, если данное используется в разных процессорах по очереди. Коммуникация точка-точка — это передача данного от одного процессора к другому.

3.3.1 Организация бродкаста

Пусть заданы функции вида (3.2), задающие размещение массивов данных в памяти виртуальных процессоров, и n-мерное таймирование с координатами вида (1.10). Как и в разделе 3.2, r первых координат таймирования задают пространственное отображение операций алгоритма в r-мерное пространство виртуальных процессоров, а остальные $n - r$ координат задают итерации, выполняемые процессорами в лексикографическом порядке.

Обозначим через $U^t_{l,\beta,q}$ матрицу, столбцы которой составлены из векторов u^t_i.

Лемма 3.1 *Пусть существуют истинные зависимости, порождаемые q-м вхождением массива a_l в оператор S_β; $\overline{\Phi}_{\alpha,\beta}$ — функция зависимостей. Если выполняется условие*

$$\Phi_{\alpha,\beta} U^t_{l,\beta,q} = 0, \tag{3.10}$$

то между итерациями множества $V_{l,\beta,q}(F) \cap V^t_\beta(T)$ данное $a_l(F)$ не переопределяется.

Доказательство. Пусть $J_1, J_2 \in V_{l,\beta,q}(F) \cap V^t_\beta(T)$, тогда $J_1 - J_2 \in \ker F_{l,\beta,q} \cap \ker T^{(\beta)}_t$. Имеем: $\overline{\Phi}_{\alpha,\beta}(J_1) - \overline{\Phi}_{\alpha,\beta}(J_2) = \Phi_{\alpha,\beta}(J_1 - J_2) = \Phi_{\alpha,\beta} \sum \alpha_i u^t_i = \sum \alpha_i (\Phi_{\alpha,\beta} u^t_i)$, где $\alpha_i \in \mathbf{Z}$. Так как выполняется (3.10), то $\Phi_{\alpha,\beta} u^t_i = 0$, откуда $\overline{\Phi}_{\alpha,\beta}(J_1) = \overline{\Phi}_{\alpha,\beta}(J_2)$. Полученное равенство означает, что данное $a_l(F)$, используемое при выполнении операций $S_\beta(J_1)$ и $S_\beta(J_2)$, $J_1, J_2 \in V_{l,\beta,q}(F) \cap V^t_\beta(T)$, переопределяется на одной и той же операции $S_\alpha(\overline{\Phi}_{\alpha,\beta}(J_1))$. Отсюда и из определения истинной зависимости следует, что между итерациями множества $V_{l,\beta,q}(F) \cap V^t_\beta(T)$ указанное данное не переопределяется. □

Теорема 3.5 *Пусть $\rho^t_{l,\beta,q} < n_\beta$. На итерации T можно организовать бродкаст данного $a_l(F)$ к процессорам $P(\overline{d}^{(l,\beta,q)}_s(F))$, где параметры функций $\overline{d}^{(l,\beta,q)}_s$ задаются согласно (3.8) и (3.9), в одном из следующих случаев:*

а) элементы массива a_l встречаются только в правых частях операторов алгоритма;

б) для истинной зависимости, порожденной вхождением (l, β, q), выполняется условие (3.10).

Доказательство. Из условия теоремы следует, что функция $\overline{F}_{l,\beta,q}$ встречается в правой части оператора S_β, поэтому элементы массива a_l используются на q-м вхождении в оператор S_β в качестве аргументов. Так как $\rho^t_{l,\beta,q} < n_\beta$, то из теоремы 3.4 и следствия из теоремы вытекает, что на одной итерации каждый элемент массива, связанный с вхождением (l, β, q), используется виртуальными процессорами, коммуникации между которыми можно задать векторами $T^{(\beta)}_s u^t_i$,

$1 \leqslant i \leqslant n_\beta - \rho^t_{l,\beta,q}$; координаты таких процессоров задаются функцией $\overline{d}_s^{(l,\beta,q)}(F)$, параметры которой определяются из системы (3.8) и соотношений (3.9). Процессоры используют одно и то же значение данного $a_l(F)$, поскольку между итерациями множества $V_{l,\beta,q}(F) \cap V^t_\beta(T)$ это данное не переопределяется: в случае *а* это данное не переопределяется вообще, в случае *б* невозможность переопределения гарантировано условием (3.10). □

В случае *а* теоремы бродкаст элемента $a_l(F)$ осуществляется от процессора $P_{in}(\overline{d}_s^{(l,\beta,q)}(F))$, в который распределяется $a_l(F)$ (см. раздел 3.2), если не применялась другая схема распределения входных массивов (например, репликация массивов). В случае *б* бродкаст осуществляется от процессора, в котором $a_l(F)$ вычислялся.

Заметим, что при выполнении условий теоремы 3.5 бродкаст может быть вырожденным, если в силу особенностей множества V_β множество $V_{l,\beta,q}(F) \cap V^t_\beta(T)$ содержит только один элемент.

3.3.2 Организация трансляции данных

Сформулируем и докажем теоремы, которые позволяют выявлять возможность организации трансляции данного. В теореме 3.6 исследуется случай, когда данное используется как аргумент на разных итерациях разными процессорами. В этом случае передача данного осуществляется на нескольких итерациях. В теореме 3.7 исследуется случай, когда данное используется как аргумент и переопределяется разными процессорами по очереди на одной итерации. В этом случае передача данного осуществляется на одной итерации, но со сдвигом по времени при выполнении программы.

Пусть $\rho_{l,\beta,q} < n_\beta$. Обозначим через $U_{l,\beta,q}$ матрицу, столбцы которой составлены из базисных векторов u_i, $1 \leqslant i \leqslant n_\beta - \rho_{l,\beta,q}$.

Лемма 3.2 *Пусть существуют истинные зависимости, порождаемые q-м вхождением массива a_l в оператор S_β; $\overline{\Phi}_{\alpha,\beta}$ — функция зависимостей. Если выполняется условие*

$$\Phi_{\alpha,\beta} U_{l,\beta,q} = 0, \tag{3.11}$$

то между итерациями множества $V_{l,\beta,q}(F)$ данное $a_l(F)$ не переопределяется.

Доказательство леммы 3.2 аналогично доказательству леммы 3.1.

Теорема 3.6 *Пусть* $\rho^t_{l,\beta,q} > \rho_{l,\beta,q}$, $\rho^s_{l,\beta,q} > \rho_{l,\beta,q}$, $\rho^t_{l,\beta,q} = n_\beta$. *На итерациях* $\overline{d}^{(l,\beta,q)}_t(F)$ *можно организовать трансляцию данного* $a_l(F)$ *между процессорами* $P(\overline{d}^{(l,\beta,q)}_s(F))$, *где параметры функций* $\overline{d}^{(l,\beta,q)}_s$ *и* $\overline{d}^{(l,\beta,q)}_t$ *задаются согласно (3.3) и (3.4), в одном из следующих случаев:*

а) массив a_l *встречается только в правых частях операторов алгоритма;*

б) для истинной зависимости, порожденной вхождением (l,β,q), *выполняется условие* (3.11).

Доказательство. Функция $\overline{F}_{l,\beta,q}$ встречается в правой части оператора S_β, поэтому элементы массива a_l используются на q-м вхождении в оператор S_β в качестве аргументов. Условие $\rho^t_{l,\beta,q} > \rho_{l,\beta,q}$ означает, что данное $a_l(F)$ во время выполнения программы используется на q-м вхождении массива a_l в оператор S_β на нескольких итерациях (следствие 3.3), а условие $\rho^s_{l,\beta,q} > \rho_{l,\beta,q}$ означает, что $a_l(F)$ используется в нескольких процессорах (следствие 3.2); из условия $\rho^t_{l,\beta,q} = n_\beta$ следует (следствие 3.5), что на фиксированной итерации a_l используется только одним виртуальным процессором.

Таким образом, данное $a_l(F)$ используется на нескольких итерациях $\overline{d}^{(l,\beta,q)}_t(F)$, в нескольких процессорах $P(\overline{d}^{(l,\beta,q)}_s(F))$, и на каждой итерации данное используется одним процессором. Процессоры используют одно и то же данное, поскольку между итерациями множества $V_{l,\beta,q}(F)$ данное $a_l(F)$ не переопределяется: в случае *а* это данное не переопределяется вообще, в случае *б* невозможность переопределения гарантировано условием (3.11). □

Трансляцию следует осуществлять согласно лексикографической упорядоченности векторов $\overline{d}^{(l,\beta,q)}_s(F)$. В случае *а* теоремы 3.6 трансляция данного $a_l(F)$ начинается от процессора $P_{in}(\overline{d}^{(l,\beta,q)}_s(F))$, в который распределяется $a_l(F)$; в случае *б* теоремы до трансляции осуществляется передача $a_l(F)$ от процессора, в котором $a_l(F)$ вычисляется, к процессору $P(\overline{d}^{(l,\beta,q)}_s(F))$ с лексикографически наименьшими координатами.

Теорема 3.7 *Пусть* $\rho^t_{l,\beta,q} < n_\beta$, *существует истинная зависимость, порожденная вхождением* (l,β,p) *в левую часть оператора* S_β *и*

вхождением (l, β, q) *в правую часть оператора. Тогда на итерации* T *можно организовать трансляцию данного* $a_l(F)$ *между процессорами* $P(\overline{d}_s^{(l,\beta,q)}(F))$, *где параметры функций* $\overline{d}_s^{(l,\beta,q)}$ *задаются согласно (3.8) и (3.9).*

Доказательство. Условие $\rho_{l,\beta,q}^t < n_\beta$ означает, что на итерации T оператор S_β выполняется, вообще говоря, более чем в одном процессоре $P(\overline{d}_s^{(l,\beta,q)}(F))$ (следствие 3.5). Так как существует истинная зависимость, порожденная вхождением p массива a_l в левую часть оператора S_β и вхождением q в правую часть, то данное $a_l(F)$ на итерации T используется как аргумент и переопределяется. Сказанное означает, что на итерации T можно организовать трансляцию данного $a_l(F)$ между процессорами $P(\overline{d}_s^{(l,\beta,q)}(F))$. □

При выполнении условий теоремы 3.7 следует до трансляции данного $a_l(F)$ осуществить его передачу от процессора, в котором он вычислялся, к процессору $P(\overline{d}_s^{(l,\beta,q)}(F))$ (где параметры функций $\overline{d}_s^{(l,\beta,q)}$ задаются согласно (3.8) и (3.9)) с лексикографически наименьшими координатами. Трансляцию следует осуществлять согласно лексикографической упорядоченности векторов $\overline{d}_s^{(l,\beta,q)}(F)$.

Как и в случае бродкаста, трансляция может быть вырожденной, если множество $V_{l,\beta,q}(F) \cap V_\beta^t(T)$ содержит только один элемент.

3.3.3 Установление схемы обмена данными

Если $\rho_{l,\beta,q} = \rho_{l,\beta,q}^s$, то каждый элемент массива, связанный с вхождением (l, β, q), используется только одним процессором (следствие 3.2). В случае $\rho_{l,\beta,q} < \rho_{l,\beta,q}^s$ необходима передача элементов массива a_l при использовании этих элементов операциями $S_\beta(J)$ на q-м вхождении элементов массива a_l в оператор S_β.

Пользуясь теоремой 3.5 можно определить группы процессоров, между которыми можно организовать бродкаст. Пользуясь теоремами 3.6, 3.7 можно определить группы процессоров, между которыми можно организовать трансляцию данных. В случаях, неоговоренных теоремами 3.5–3.7, можно организовать коммуникации точка-точка.

Между процессорами каждой группы, для которой установлен вид коммуникации, обмен данными следует организовывать, пользуясь сле-

дующими правилами очередности передачи и приема данного, используемого при выполнении операции $S_\beta(J)$.

Пусть данное передается от процессора, который использует его в качестве результата операции $S_\beta(J)$. Передачу данного следует осуществлять после выполнения операции $S_\beta(J)$.

Пусть данное передается от процессора, который использует его в качестве аргумента операции $S_\beta(J)$ (при трансляции, которую определяет теорема 3.6). Передачу данного следует осуществлять в начале итерации.

Пусть данное передается от процессора, в локальной памяти которого оно хранится. Передачу данного следует осуществлять после того как будут выполнены все операции, которые изменяют значение этого данного и выполняются в исходной программе раньше операции $S_\beta(J)$, и после того как процессор осуществит прием результатов этих операций от других процессоров.

Пусть данное передается процессору, который использует его в качестве аргумента операции $S_\beta(J)$. Прием данного следует осуществлять до выполнения операции $S_\beta(J)$.

Пусть данное передается процессору, в локальной памяти которого оно хранится. Прием данного следует осуществлять до того как будут выполняться все операции, которые используют это данное в качестве аргумента и выполняются в исходной программе после операции $S_\beta(J)$, и до того как процессор осуществит передачу аргументов этих операций другим процессорам.

Пример 4 (продолжение, см. разделы 2.5, 3.2). Рассмотрим два полученных ранее двумерных таймирования и в каждом случае установим схему обмена данными.

Пусть распределение операций между процессорами определяется функциями $t_1^{(1)}(1) = 1$, $t_1^{(2)}(i) = i$, $2 \leqslant i \leqslant N$, $t_1^{(3)}(i,j) = i$, $2 \leqslant i \leqslant N$, $1 \leqslant j \leqslant i-1$; зададим итерации, выполняемые процессорами, функциями $t_2^{(1)}(1) = 1$, $t_2^{(2)}(i) = 1$, $t_2^{(3)}(i,j) = j$. Эти функции получены в разделе 2.5

Запишем соответствующий код (единым образом для каждого процессора), предназначенный для выполнения алгоритма на N процессорах. Символ p обозначает номер процессора. Переменная цикла t соответствует итерациям алгоритма.

```
if p = 1 then
  S1 : x(1) = b(1)
endif
if 2 ⩽ p ⩽N then
  S2 : x(p) = b(p)
  do t = 1, p-1
    S3 : x(p) = x(p) - a(p,t)x(t)
  enddo
endif
```

Равенства $\rho_{l,\beta,q} = \rho^s_{l,\beta,q}$ выполняются для всех вхождений (l, β, q), кроме $(1, 3, 3)$ (см. пример 4 в разделе 3.2). Поэтому при выполнении алгоритма параллельным компьютером для использования элементов массива x на третьем вхождении этого массива в оператор S_3 необходимо осуществлять передачу данных.

Определим коммуникации, которые можно организовать для обмена элементами массива x. Имеем: $\rho^t_{1,3,3} = \operatorname{rank}\begin{pmatrix} F_{1,3,3} \\ T^{(3)}_t \end{pmatrix} = \operatorname{rank}\begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} = 1$, $n_3 = 2$, $\rho^t_{1,3,3} < n_3$. Истинные зависимости, порожденные третьим вхождением массива x в оператор S_3, задаются функциями $\overline{\Phi}_{1,3}$ и $\overline{\Phi}^{(1)}_{3,3}$. Для обеих функций выполняются условия (3.10): $\Phi_{1,3}U^t_{1,3,3} = (0\ 0)(1\ 0)^{\mathrm{т}} = 0$ и $\Phi^{(1)}_{3,3}U^t_{1,3,3} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = 0$. Следовательно, согласно теореме 3.5 можно осуществить бродкаст данного $x(F)$ к процессорам $P(\eta^{(1,3,3)}F + y_{1,3,3}(F))$, где $\eta^{(1,3,3)} = 0$ ($\ker\begin{pmatrix} F_{1,3,3} \\ T^{(3)}_t \end{pmatrix} = \ker\begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$, $u^{\perp,t}_1 = (0,1)$, $\tau^{(3,1)} = (1,0)$, $\eta^{(1,3,3,1)}(0\ 1)\begin{pmatrix} 0 \\ 1 \end{pmatrix} = (1\ 0)\begin{pmatrix} 0 \\ 1 \end{pmatrix}$), $y_{1,3,3}(F) = i$ ($a_{3,1} = 0$, $u^t_1 = (1,0)$, $V_{1,3,3}(F) = \{(i,F) \mid F+1 \leqslant i \leqslant N\}$, $V^t_3(T) = \{(i,T) \mid T+1 \leqslant i \leqslant N\}$, $V_{1,3,3}(F) \cap V^t_3(T) = \{(i,F) \mid F+1 \leqslant i \leqslant N,\ F = T\}$, $\Lambda^t_{1,3,3}(F,T) = \{i \mid F+1 \leqslant i \leqslant N,\ F = T\}$, $y_{1,3,3}(F) = i(1,0)(1,0) = i$), $F+1 \leqslant i \leqslant N$. Таким образом, на итерации T бродкаст данного $x(F)$, $F = T$, осуществляется от процессора $P(F)$, в котором $x(F)$

вычислялся (см. пример 4 в разделе 3.2), к процессорам $P(i),\ T + 1 \leqslant i \leqslant N$. На итерации $N-1$ бродкаст вырождается в коммуникацию точка-точка от процессора $P(N-1)$ к процессору $P(N)$.

Установим схему обмена данными пользуясь правилами очередности передачи и приема данных. В процессоре $P(p),\ 1 \leqslant p \leqslant N-1$, на итерации p необходимо осуществить передачу данного $x(p)$, которое находится в его локальной памяти, процессорам $P(i),\ p+1 \leqslant i \leqslant N$. На итерации p операции, изменяющие значение данного $x(p)$, не выполняются, поэтому передачу $x(p)$ процессором $P(p)$ можно осуществить в начале итерации. В процессоре $P(p),\ 2 \leqslant p \leqslant N$, на каждой итерации $t,\ 1 \leqslant t \leqslant p-1$, необходимо осуществить прием процессором данного $x(t)$ от процессора $P(t)$ для использования данного операцией $S_3(p,t)$ в качестве аргумента. Прием данного $x(t)$ от процессора $P(t)$ следует осуществить на итерации t до выполнения операции $S_3(p,t)$. Таким образом, согласно установленной схеме обмена данными, можно записать SPMD код алгоритма:

```
if p = 1 then
  S1 : x(1) = b(1)
endif
if 2 ⩽ p ⩽ N then
  S2 : x(p) = b(p)
  do t = 1, p-1
    \\ прием данного:
    Broadcast x(t) from P(t) to P(t+1),...,P(N)
    S3 : x(p) = x(p) - a(p,t)x(t)
  enddo
endif
if 1 ⩽ p ⩽ N-1 then
  \\ передача данного:
  Broadcast x(p) from P(p) to P(p+1),...,P(N)
endif
```

Пусть теперь распределение операций между процессорами определяется функциями $t_1^{(1)}(1) = 1$, $t_1^{(2)}(i) = 1$, $2 \leqslant i \leqslant N$, $t_1^{(3)}(i,j) = j$, $2 \leqslant i \leqslant N$, $1 \leqslant j \leqslant i-1$. Положим $t_2^{(1)}(1) = 2$, $t_2^{(2)}(i) = i$, $t_2^{(3)}(i,j) = i$ (см. раздел 2.5). Запишем код, предназначенный для выполнения алгоритма на $N-1$ процессоре.

```
if p = 1 then
  S1 : x(1) = b(1)
  do t = 2, N
    S2 : x(t) = b(t)
    S3 : x(t) = x(t) - a(t,1)x(1)
  enddo
endif
if 2 ⩽ p ⩽ N-1 then
  do t = p+1, N
    S3 : x(t) = x(t) - a(t,p)x(p)
  enddo
endif
```

Равенства $\rho_{l,\beta,q} = \rho^s_{l,\beta,q}$ выполняются для всех вхождений (l, β, q), кроме $(1,3,1)$ и $(1,3,2)$ $(\rho_{1,3,1} = \rho_{1,3,2} = 1,\ \rho^s_{1,3,1} = \rho^s_{1,3,2} = 2)$. Поэтому при выполнении алгоритма параллельным компьютером для использования элементов массива x на первом и втором вхождениях этого массива в оператор S_3 необходимо осуществлять передачу данных.

Так как $\rho^t_{1,3,1} < n_3$ $(\rho^t_{1,3,1} = 1,\ n_3 = 2)$ и существует истинная зависимость, порожденная вхождениями $(1,3,1)$, $(1,3,2)$ (задается функцией $\overline{\Phi}^{(2)}_{3,3}$), то согласно теореме 3.7 можно организовать трансляцию данного $a_l(F)$. Трансляция осуществляется на итерации T между процессорами $P(\eta^{(1,3,1)}F + y_{1,3,1}(F))$, где (с учетом $u_1^{\perp,t} = (1,0)$, $\tau^{(3,1)} = (0,1)$, $a_{3,1} = 0$, $u_1^t = (0,1)$, $V_{1,3,1}(F) = \{(F,j) \mid 1 \leqslant j \leqslant F - 1\}$, $V_3^t(T) = \{(T,j) \mid 1 \leqslant j \leqslant T-1\}$, $V_{1,3,1}(F) \cap V_3^t(T) = \{(F,j) \mid 1 \leqslant j \leqslant F-1,\ F = T\}$, $\Lambda^t_{1,3,1}(F,T) = \{j \mid 1 \leqslant j \leqslant F - 1,\ F = T\}$) $\eta^{(1,3,1)} = 0$, $y_{1,3,1}(F) = j$, $1 \leqslant j \leqslant F-1,\ F = T$.

Трансляцию следует осуществлять от процессора с меньшим номером к процессору с большим номером. На второй итерации $(F = T = 2)$ для передачи данного $x(2)$ трансляция вырождается в коммуникацию точка-точка от процессора $P(1)$ к процессору $P(2)$.

Установим схему обмена данными пользуясь правилами очередности передачи и приема данных. В процессоре $P(p),\ 1 \leqslant p \leqslant N-2$, на каждой итерации $t,\ p+1 \leqslant t \leqslant N$, необходимо осуществить передачу данного $x(t)$ процессору $P(p+1)$ как результат выполнения операции $S_3(t,p)$. Передачу следует осуществлять после выполнения операции $S_3(t,p)$. В процессоре $P(p),\ 2 \leqslant p \leqslant N-1$, на итерации p необходимо

осуществить прием данного $x(p)$. На итерации p операции, которые используют данное $x(p)$ не выполняются, поэтому прием данного можно осуществить в начале итерации. В процессоре $P(p),\ 2 \leqslant p \leqslant N-1$, на каждой итерации $t,\ p+1 \leqslant t \leqslant N$, необходимо осуществить прием данного $x(t)$ от процессора $P(p-1)$ для использования данного операцией $S_3(t,p)$ в качестве аргумента. Прием следует осуществить до выполнения операции $S_3(t,p)$. Согласно установленной схеме обмена данными, запишем SPMD код алгоритма (предназначенный для выполнения алгоритма на $N-1$ процессоре):

```
if p = 1 then
  S1 : x(1) = b(1)
  do t = 2, N
    S2 : x(t) = b(t)
    S3 : x(t) = x(t) - a(t,1)x(1)
    Send x(t) to P(2)
  enddo
endif
if 2 ⩽ p ⩽ N-2 then
  Receive x(p) from P(p-1)
  do t = p+1, N
    Receive x(t) from P(p-1)
    S3 : x(t) = x(t) - a(t,p)x(p)
    Send x(t) to P(p+1)
  enddo
endif
if  p = N-1 then
  Receive x(N-1) from P(N-2)
  Receive x(N) from P(N-2)
  S3 : x(N) = x(N) - a(N,N-1)x(N-1)
endif
```

Отметим, что на практике для эффективной реализации алгоритма на многопроцессорной вычислительной системе с распределенной памятью необходимо объединить вычисления алгоритма в более крупные зерна.

3.4 Улучшение локальности гнезд циклов

Время решения задачи на современном компьютере определяется не столько скоростью выполнения вычислительных операций, сколько скоростью доступа к памяти. Скорость доступа существенно зависит от места в иерархической памяти (диск, оперативная память, кэш второго уровня, кэш первого уровня, регистры), где хранятся данные. Поэтому для быстрой реализации алгоритма, заданного последовательной программой, задача быстрого (до исчезновения из памяти с быстрым доступом) переиспользования данных является одной из важнейших.

Под улучшением локальности программы понимается такое ее преобразование, которое позволяет процессору быстро переиспользовать данные (элементы массивов) на большем числе итераций, чем до преобразования. Улучшение локальности программы называют также локализацией данных.

Различают задачи улучшения временной локальности и пространственной локальности. Временная локальность требует установить такой порядок выполнения операций, при котором данные использовались бы повторно прежде, чем они будут перемещены из памяти с быстрым доступом в память более низкого уровня. Желательно использовать одни те же данные на всех итерациях самого внутреннего цикла. Пространственная локальность предполагает использование данных, расположенных в памяти близко к недавно использованным, а следовательно, расположенных с большой вероятностью в памяти с быстрым доступом.

В этом разделе исследуется применение аффинных преобразований циклов и техники многомерного таймирования для улучшения локальности. Мы увидим, что математический аппарат, применяемый для распараллеливания алгоритмов, во многом применим и для локализации данных.

Дадим формальное определение локальности данных. Зафиксируем q-е вхождение массива a_l в оператор S_β.

Определение (временная локальность, связанная с выбранным вхождением). *Данные* $a_l(\overline{F}_{l,\beta,q}(J))$, $J(J_1, \ldots, J_{n_\beta}) \in V_\beta$, *называются*

локальными по времени (t-локальными), если функция $a_l(\overline{F}_{l,\beta,q}(J))$, *не зависит от последней координаты* J_{n_β} *вектора* J.

Другими словами, элементы массива, связанные с выбранным вхождением в некоторый оператор, локальны по времени, если индексы этих элементов не зависят от параметра самого внутреннего цикла. Таким образом, при фиксированных значениях параметров внешних циклов каждое t-локальное данное участвует в вычислениях на всех итерациях самого внутреннего цикла.

Заметим, что t-локальность элементов массива не зависит от способа размещения (по строкам, по столбцам, развернутый в одну строку) массива в памяти и сохраняется при переразмещении массива.

Определение (пространственная локальность, связанная с выбранным вхождением). *Данные* $a_l(\overline{F}_{l,\beta,q}(J))$, $J(J_1,\ldots,J_{n_\beta}) \in V_\beta$, *называются локальными пространственно (s-локальными), если данное* $a_l(\overline{F}_{l,\beta,q}(J_1,\ldots,J_{n_\beta}+1))$ *близко следует в памяти за данным* $a_l(\overline{F}_{l,\beta,q}(J_1,\ldots,J_{n_\beta}))$.

Другими словами, элементы массива, связанные с выбранным вхождением в некоторый оператор, локальны пространственно, если при увеличении параметра самого внутреннего цикла на единицу получим элемент массива, близко следующий в памяти за элементом массива до увеличения параметра. Определение s-локальности не является строгим, поскольку требует уточнения понятия близкого расположения в памяти. Будем считать, что элементы $a_l(F_1)$ и $a_l(F_2)$ расположены в памяти близко, если при перемещении $a_l(F_1)$ в память более высокого уровня перемещается также и $a_l(F_2)$.

Из приведенного определения следует, что t-локальность можно считать предельным, наиболее благоприятным случаем s-локальности: итерации внутреннего цикла используют на q-м вхождении в оператор S_β только одно данное.

Пусть имеется зависимость $S_\alpha(I) \to S_\beta(J)$ любого типа. Чем лексикографически ближе величины $\overline{t}^{(\beta)}(J)$ и $\overline{t}^{(\alpha)}(I)$, тем быстрее переиспользование элемента массива, определяющего зависимость. Поэтому для временной локализации данных при поиске функций $\overline{t}^{(\beta)}$, задающих преобразование циклов, необходимо стремиться к выполнению равенства $\overline{t}^{(\beta)}_{1:d}(J) = \overline{t}^{(\alpha)}_{1:d}(I)$ для наибольшего значения d. При этом для всех типов зависимостей, кроме зависимостей по входу, должны выполняться условия (1.12) сохранения зависимостей.

Для пространственной локализации необходимо, чтобы операции алгоритма, которые используют соседние с точки зрения хранения в памяти элементы массива данных, выполнялись бы друг за другом. Если хранение элементов массива осуществляется по столбцам, то близко расположенными в памяти элементами массива являются элементы, отличающиеся только первой координатой в индексных выражениях; если хранение элементов массива осуществляется по строкам, то близко расположенными в памяти элементами массива являются элементы, отличающиеся только последней координатой.

Пространственную локализацию данных будем осуществлять среди операций одного и того же оператора при фиксированном вхождении элемента массива в оператор. Индексы элементов $a_l(\overline{F}_{l,\beta,q}(I))$ и $a_l(\overline{F}_{l,\beta,q}(J))$ l-го массива отличаются только первой (последней) координатой, если векторы $F_{l,\beta,q}I + f^{(l,\beta,q)}$ и $F_{l,\beta,q}J + f^{(l,\beta,q)}$ отличаются только первой (последней) координатой. Отсюда следует, что при фиксированном вхождении q-го элемента массива a_l в оператор S_β на итерациях I и J используются близко расположенные в памяти данные, если $F^{(1)}_{l,\beta,q}I = F^{(1)}_{l,\beta,q}J$, где $F^{(1)}_{l,\beta,q}$ — матрица размера $(\nu_l - 1) \times n_\beta$, составленная из всех, кроме первой (если хранение по столбцам) или последней (если хранение по строкам), строк матрицы $F_{l,\beta,q}$.

Обозначим: $r^{(1)}_{l,\beta,q}$ — ранг матрицы $F^{(1)}_{l,\beta,q}$; $s^{(1)}_{l,\beta,q} = n_\beta - r^{(1)}_{l,\beta,q}$; $U^{(1)}_{l,\beta,q}$ — матрица, столбцы которой составлены из базисных векторов пересечения подпространства $\ker F^{(1)}_{l,\beta,q}$ и $\mathbf{Z}^{n_\beta}$ (предполагается $r^{(1)}_{l,\beta,q} < n_\beta$).

Теорема 3.8 *Пусть многомерное таймирование $(t_1, t_2, \ldots, t_n)$ с координатными функциями вида (1.13) задает преобразование гнезда циклов. Если среди функций $t^{(\beta)}_1, t^{(\beta)}_2, \ldots, t^{(\beta)}_d$, $d < n$, имеется $r^{(1)}_{l,\beta,q}$ функций $t^{(\beta)}_\xi$, $\xi \in \Xi = \{\xi_1, \xi_2, \ldots, \xi_{r^{(1)}_{l,\beta,q}}\}$, удовлетворяющих условиям*

$$T^{(1)}_{l,\beta,q}U^{(1)}_{l,\beta,q} = 0, \tag{3.12}$$

$$\operatorname{rank} T^{(1)}_{l,\beta,q} = r^{(1)}_{l,\beta,q}, \tag{3.13}$$

где $T^{(1)}_{l,\beta,q}$ — матрица, строки которой составлены из векторов $\tau^{(\xi_1)}, \tau^{(\xi_2)}, \ldots, \tau^{(\xi_{r^{(1)}_{l,\beta,q}})}$, то при фиксированных значениях параметров d внешних циклов на q-м вхождении в оператор S_β массива a_l используются элементы только одного столбца (строки) массива.

Доказательство. Достаточно показать, что любые итерации преобразованного гнезда циклов, использующие на q-м вхождении оператора S_β элементы разных столбцов (строк) l-го массива, отличаются значением хотя бы одного из параметров d внешних циклов. Формально докажем следующее: для любых $I, J \in V_\beta$ таких, что $F^{(1)}_{l,\beta,q} I \neq F^{(1)}_{l,\beta,q} J$, существует $\xi' \in \Xi$ такое, что $t^{(\beta)}_{\xi'}(I) \neq t^{(\beta)}_{\xi'}(J)$.

Пусть $I, J \in V_\beta$ такие, что $F^{(1)}_{l,\beta,q} I \neq F^{(1)}_{l,\beta,q} J$. Тогда имеет место равенство $F^{(1)}_{l,\beta,q}(I - J) = s$, $s \in \mathbf{Z}^{\nu_l - 1}$, $s \neq 0$. Существуют такие λ_i, $1 \leqslant i \leqslant s^{(1)}_{l,\beta,q}$, что $I - J = \sum\limits_{i=1}^{s^{(1)}_{l,\beta,q}} \lambda_i u_i + \widetilde{u}$, где u_i — базисные векторы пересечения $\ker F^{(1)}_{l,\beta,q} \cap \mathbf{Z}^{n_\beta}$, $\widetilde{u}$ — решение неоднородной системы линейных уравнений $F^{(1)}_{l,\beta,q} x = s$, векторы $\widetilde{u}$ и u_i, $1 \leqslant i \leqslant s^{(1)}_{l,\beta,q}$, линейно-независимы.

Покажем, что $T^{(1)}_{l,\beta,q} \widetilde{u} \neq 0$. Так как $\operatorname{rank} T^{(1)}_{l,\beta,q} = r^{(1)}_{l,\beta,q}$, то базис подпространства $\ker T^{(1)}_{l,\beta,q}$ содержит $n_\beta - r^{(1)}_{l,\beta,q}$ векторов. Ввиду $T^{(1)}_{l,\beta,q} u_i = 0$, $1 \leqslant i \leqslant s^{(1)}_{l,\beta,q}$, и $s^{(1)}_{l,\beta,q} = n_\beta - r^{(1)}_{l,\beta,q}$, векторы u_i, $1 \leqslant i \leqslant s^{(1)}_{l,\beta,q}$, есть фундаментальная система решений системы $T^{(1)}_{l,\beta,q} x = 0$. Отсюда следует, что $T^{(1)}_{l,\beta,q} \widetilde{u} \neq 0$, в противном случае вектор $\widetilde{u}$ являлся бы линейной комбинацией векторов u_i, что противоречило бы линейной независимости векторов $\widetilde{u}$ и u_i.

Так как $T^{(1)}_{l,\beta,q} \widetilde{u} \neq 0$, то $\tau^{(\xi')} \widetilde{u} \neq 0$ для некоторого $\xi' \in \Xi$. Поэтому

$$t^{(\beta)}_{\xi'}(I) - t^{(\beta)}_{\xi'}(J) = \tau^{(\xi')}(I - J) = \tau^{(\xi')} \left(\sum_{i=1}^{s^{(1)}_{l,\beta,q}} \lambda_i u_i + \widetilde{u} \right) = \sum_{i=1}^{s^{(1)}_{l,\beta,q}} \lambda_i \tau^{(\xi')} u_i +$$

$$+ \tau^{(\xi')} \widetilde{u} = \tau^{(\xi')} \widetilde{u} \neq 0. \qquad \square$$

Таким образом, для улучшения пространственной локальности требуется получить $r^{(1)}_{l,\beta,q}$ линейно независимых векторов $\tau^{(\xi)}$, удовлетворяющих условию (3.12), причем по возможности с наименьшими значениями ξ.

Приведем простейший алгоритм улучшения локальности.

Обозначим:

$\widetilde{\tau}^{(\xi)} = (\tau^{(\xi)}_1, \ldots, \tau^{(\xi)}_n, a_{1,\xi}, \ldots, a_{K,\xi})$ — вектор параметров функций вида (1.13);

D — матрица, характеризующая все зависимости, кроме зависимостей по входу (D определена в разделе 2.4); пусть всего матрица D содержит μ_D столбцов: $D \in \mathbf{Z}^{(n+K)\times\mu_D}$;

D_{in} — матрица, характеризующая зависимости по входу; $D_{in} \in \mathbf{Z}^{(n+K)\times\mu_{D_{in}}}$;

$\widetilde{u}_i = \begin{pmatrix} u_i \\ 0^{(n+K-n_\beta)} \end{pmatrix}$ — вектор размера $n+K$;

D_u — матрица, столбцы которой составлены из векторов $\widetilde{u}_i$; $D_u \in \mathbf{Z}^{(n+K)\times\mu_{D_u}}$.

Алгоритм 3.1 (получение таймирующих функций для улучшение локальности данных).

1. Сформировать матрицу $(D|D_{in}|D_u) \in \mathbf{Z}^{(n+K)\times(\mu_D+\mu_{D_{in}}+\mu_{D_u})}$.

2. Элементарными строчными преобразованиями матрицы $(E^{(n+K)}|D|D_{in}|D_u)$ *получить матрицу* $(P|PD|PD_{in}|PD_u)$, *где* PD — *эрмитова нормальная форма матрицы* D *(с точностью до перестановки строк и столбцов и с точностью до положительных множителей строк матрицы).*

3. Сложением строк получить на месте матрицы D *матрицу, состоящую только из неотрицательных элементов. Если этого добиться не удалось, то строки с отрицательными элементами далее не рассматривать.*

4. Сложением строк получить на месте матриц $(D_{in}|D_u)$ *как можно больше нулей.*

5. Получить n *наборов функций* $t_\xi^{(\beta)}$, *задающих аффинное преобразование гнезда циклов. Для этого в качестве координат векторов* $\tau^{(\xi)}$ *и констант* $a_{\beta,\xi}$ *выбрать такие* n *строк на месте матрицы* $E^{(n+K)}$, *чтобы векторы, составленные из* n *первых координат строк, были линейно-независимыми, а в соответствующих строках на месте матриц* $(D_{in}|D_u)$ *было как можно больше нулей; чем больше в строке нулей, тем меньший номер* ξ *присвоить координате многомерного таймирования, порождаемой строкой.*

Приведенный алгоритм ищет многомерное таймирование, все n координат которого являются одномерными t-функциями. Как уже отмечалось в разделе 2.4, в случае алгоритмов с аффинными зависимо-

стями может не существовать n таймирований вида (1.13) с линейно-независимыми векторами τ.

Пример 2 (продолжение). Рассмотрим задачу улучшения локальности гнезда циклов (1.2) для случая хранения элементов массива a по столбцам.

Помимо зависимостей, характеризуемых матрицей $D = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ -1 & 0 \\ 1 & 0 \end{pmatrix}$ (см. пример 2 в разделах 1.1, 2.3, 2.4), имеется зависимость по входу $S_2(i-1,j) \to S_2(i,j)$, порождаемая массивом b. Зависимость является однородной, $\Phi_{(2,2)_2} = E^{(2)}$, $\varphi^{(2,2)_2} = (1,0)$, $\widetilde{\Phi}_{(2,2)_2} = 0$, $\widetilde{\varphi}^{(2,2)_2} = (1,0,0,0)$, $D_{in} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$.

Рассмотрим величины, характеризующие пространственную локализацию массива a : $F_{1,2,1} = E^{(2)}$, $F^{(1)}_{1,2,1} = (0\,1)$, $r^{(1)}_{1,2,1} = 1$, $s^{(1)}_{1,2,1} = 1$, $u_1 = (1,0)$, $D_u = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$, $(E^{(5)}|D|D_{in}|D_u) = \left(\begin{array}{cccc|cc|c|c} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{array}\right)$.
Прибавив ко второй строке третью, к третьей строке четвертую, получим $(P|B) = \left(\begin{array}{cccc|cc|c|c} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{array}\right)$.

Вторая и первая строки матрицы P порождают двумерное таймирование $\bar{t}^{(1)}(i) = (1,i)$. $\bar{t}^{(2)}(i,j) = (j,i)$. Соответственно преобразуется гнездо циклов:

```
do  i = 1, N
    S1 :  c(i) = 0
enddo
do  j = 1, N
    do  i = 1, N
        S2 :  c(i) = c(i) + a(i, j)b(j)
    enddo
enddo
```

Время умножения матрицы на вектор на основе ФОРТРАН-реализаций преобразованного гнезда циклов и исходного гнезда циклов (1.2) может отличаться значительно. Например, при $N = 8000$ преобразованный алгоритм выполняется в 10 раз быстрее (на платформе Intel P4). Основной причиной уменьшения времени вычисления является пространственная локализация двумерного массива a. Улучшение пространственной локальности гарантировано наличием нулевого элемента во второй строке на месте матрицы D_u. □

Характеризовать локальность гнезд циклов можно с помощью так называемых векторов локальности.

Пусть a_l — массив размерности большей или равной 2, хранение элементов массива в памяти компьютера осуществляется по строкам. Рассмотрим данные $a_l(\overline{F}_{l,\beta,q}(J)),\ J \in V_\beta$, используемые на q-м вхождении массива a_l в оператор S_β. Определим вектор $\lambda_{l,\beta,q} = (\lambda_0, \lambda_1)$, где λ_0 — число внутренних циклов, на итерациях которых используется элемент массива a_l со всеми фиксированными индексами (т. е. используется только один элемент массива); λ_1 — число внутренних циклов, на итерациях которых используются элементы массива a_l с одним последним переменным индексом (если a_l — двумерный массив, то используется одна строка). Вектор $\lambda_{l,\beta,q}$ называется вектором локальности данных $a_l(\overline{F}_{l,\beta,q}(J))$. Первая компонента вектора характеризует временную локальность, вторая — пространственную.

Для оптимизации работы с памятью желательно, чтобы как можно больше векторов $\lambda_{l,\beta,q}$ были ненулевыми и имели как можно большее значение координат.

Пример. Рассмотрим основную часть ijk-алгоритма перемножения двух квадратных матриц порядка 1000:

```
do  i = 1, 1000
   do  j = 1, 1000
      do  k = 1, 1000
         S1 :  c(i, j) = c(i, j) + a(i, k)b(k, j)
      enddo
   enddo
enddo
```

Перестановкой циклов можно получать различные модификации этого алгоритма. В следующей таблице приведены векторы локальности и

время вычислений для четырех случаев очередности выполнения циклов; хранение элементов массивов осуществляется по строкам (язык программирования — C).

	ijk	ikj	jki	kij
λ^c	$(1,2)$	$(0,2)$	$(0,0)$	$(0,1)$
λ^a	$(0,2)$	$(1,2)$	$(0,0)$	$(1,1)$
λ^b	$(0,0)$	$(0,1)$	$(1,1)$	$(0,2)$
T	89 366	16 814	307 022	17 204

В таблице хорошо видна зависимость времени вычислений от количества ненулевых координат векторов локальности. □

3.5 Блокинг. Тайлинг

Одной из основных проблем, возникающих при отображении вычислительных алгоритмов на параллельные компьютеры, является проблема выбора зерна вычислений. Под зерном вычислений (или тайлом) понимается множество операций алгоритма, выполняемых атомарно: все вычисления, принадлежащие одному зерну, производятся на одном процессоре и не могут прерываться синхронизацией, а обмен данными, связанными с вычислениями одного зерна, не может выполняться на фоне этих вычислений.

Целью тайлинга (т. е. получения зерен вычислений) является уменьшение накладных расходов на обмен данными между процессорами и (или) на использование иерархической памяти одного процессора. Так как время решения задачи на параллельном компьютере во многом определяется скоростью коммуникаций и доступа к памяти, то объединение операций алгоритма в тайлы является важным этапом реализации алгоритмов на многопроцессорных вычислительных системах с распределенной памятью. В этом разделе предлагаются способы выбора зерна вычислений, исследуются задачи, возникающие при выборе зерна вычислений.

3.5.1 Задачи, связанные с разбиением операций алгоритма на зерна вычислений

Стратегии, следуя которым алгоритм отображается на целевой параллельный компьютер, могут быть разными, но при задании зерна вычислений так или иначе необходимо решить следующие задачи.

- **Задача 1**. Выбор формы тайла (геометрически тайл можно представить множеством точек многомерной целочисленной решетки). Наиболее простой для использования является форма прямоугольного параллелепипеда; рассматриваются также тайлы в виде произвольного параллелепипеда. Известны методы оптимизации формы тайла при заданном его объеме и оптимизация формы тайла с целью уменьшения времени простоя процессоров параллельного компьютера.

- **Задача 2**. Выбор размера (числа операций) тайла. Условием, исходя из которого выбираются параметры тайла, может являться минимальность отношения объема коммуникаций к объему вычислений. Большой интерес для практики представляют работы, в которых выбор зерна вычислений исследуется с точки зрения отображения на целевую параллельную архитектуру. Особенно интересны исследования, учитывающие параметры (такие как производительность процессора, время инициализации, пропускная способность каналов связи) параллельного компьютера.

- **Задача 3**. Выбор последовательности выполнения операций, принадлежащих тайлу. Порядок выполнения операций влияет на локальность данных, а следовательно, и на время вычислений. Отметим два пути улучшить локальность. Один путь — повторное разбиение тайлов на тайлы второго уровня; двухуровневое разбиение может улучшить локальность данных ввиду наличия кэшей двух уровней. Другой путь — установление приоритетов выполнения циклов с точки зрения локальности данных и перестановка их в соответствии с приоритетами.

- **Задача 4**. Выбор последовательности выполнения тайлов, реализуемых одним процессором. Порядок выполнения тайлов может влиять на начало вычислений другими процессорами, а следовательно, и на загруженность процессоров. Поэтому естественно стремиться в первую очередь выполнять тайлы, от результатов операций которых зависят вычисления других процессоров.

3.5.2 Разбиение пространств итераций на блоки и тайлы

Обозначим, как и ранее, $n = \max\limits_{1 \leqslant \beta \leqslant K} n_\beta$. Пусть функции $\overline{t}^{(\beta)} = (t_1^{(\beta)}, \ldots, t_n^{(\beta)})$, $1 \leqslant \beta \leqslant K$, с координатными функциями вида $t_\xi^{(\beta)}(J) = \tau^{(\beta,\xi)} J + a_{\beta,\xi}$, $J \in V_\beta$, $\tau^{(\beta,\xi)} \in \mathbf{Z}^{n_\beta}$, $a_{\beta,\xi} \in \mathbf{Z}$, $1 \leqslant \xi \leqslant n$, являются многомерным таймированием алгоритма. Будем считать, что первые r $(r < n)$ координат многомерного таймирования задают распределение операций между виртуальными процессорами, размещенными в r-мерном пространстве. Это означает, что операция $S_\beta(J)$, выполняемая на итерации J исходного гнезда циклов, выполняется виртуальным процессором с координатами $(t_1^{(\beta)}(J), \ldots, t_r^{(\beta)}(J))$.

Пусть целевой компьютер содержит $P_1 \times \ldots \times P_r$ процессоров, где $P_1, \ldots, P_r$ — заданные натуральные числа, превосходящие 1. Обозначим $P = \{(p_1, \ldots, p_r) \in \mathbf{Z}^r |\ 1 \leqslant p_\xi \leqslant P_\xi,\ 1 \leqslant \xi \leqslant r\}$, $m_\xi^{(\beta)} = \min\limits_{J \in V_\beta} t_\xi^{(\beta)}(J)$, $B_\xi^{(\beta)} = \left\lceil \dfrac{\max\limits_{J \in V_\beta} t_\xi^{(\beta)}(J) - m_\xi^{(\beta)}}{P_\xi} \right\rceil$, $1 \leqslant \beta \leqslant K$, $1 \leqslant \xi \leqslant n$. Разобьем каждую область итераций V_β на $P_1 \times \ldots \times P_r$ областей $V_{\overline{p}}^{(\beta)} = \{J \in V_\beta |\ m_\xi^{(\beta)} + (p_\xi - 1)B_\xi^{(\beta)} \leqslant t_\xi^{(\beta)}(J) \leqslant m_\xi^{(\beta)} - 1 + p_\xi B_\xi^{(\beta)}, 1 \leqslant \xi \leqslant r\}$, $\overline{p} \in P$. Области $V_{\overline{p}}^{(\beta)}$ могут быть вырожденными из-за особенностей итерационных областей V_β.

Каждому вектору $\overline{p}$ взаимно-однозначно соответствует процессор целевого компьютера. Выполнение итераций $J \in V_{\overline{p}}^{(\beta)}$ назначается процессору с координатами $\overline{p} = (p_1, \ldots, p_r)$.

Разобьем теперь каждую из областей $V_{\overline{p}}^{(\beta)}$ на $Q_1 \times \ldots \times Q_n$ областей $V_{\overline{p},\overline{q}}^{(\beta)}$, $\overline{q} \in \{(q_1, \ldots, q_n) \in \mathbf{Z}^n |\ 1 \leqslant q_\eta \leqslant Q_\eta,\ 1 \leqslant \eta \leqslant n\}$ и будем считать, что вычисление операций $S_\beta(J)$, $J \in V_{\overline{p},\overline{q}}^{(\beta)}$, не прерывается синхронизацией или обменом данными, требуемыми для выполнения этих операций. Области $V_{\overline{p},\overline{q}}^{(\beta)}$ могут быть вырожденными не только из-за особенностей V_β, но и потому, что некоторые Q_η могут равняться единице.

Области $V_{\overline{p}}^{(\beta)}$ назовем блоками итераций, а области $V_{\overline{p},\overline{q}}^{(\beta)}$ — тайлами итераций. Множество операций $S_\beta(J)$ назовем блоком вычислений,

если $J \in V_{\overline{p}}^{(\beta)}$, и зерном (тайлом) вычислений, или просто тайлом, если $J \in V_{\overline{p},\overline{q}}^{(\beta)}$.

3.5.3 Выбор параметров тайла и локальность данных

От выбора параметров тайла, от изменения последовательности выполнения операций, принадлежащих одному тайлу, может существенно зависеть время выполнения итераций области $V_{\overline{p}}^{(\beta)}$ на одном процессоре (процессоре с координатами $\overline{p}$). Как уже отмечалось (задача 3), изменение порядка выполнения операций влияет на локальность данных, а следовательно, и на время вычислений.

Теоретически при повторном разбиение тайлов на тайлы второго уровня локализация данных должна улучшиться за счет иерархической структуры памяти (наличия кэшей двух уровней — $L1$ и $L2$). Однако численные эксперименты показывают, что при оптимальном с точки зрения локальности размере тайлов повторный тайлинг не приводит к уменьшению времени вычислений. Это можно объяснить малым размером кэша $L1$, который постоянно занят системной информацией и практически не используется для хранения данных. Тем не менее повторный тайлинг может быть полезен в случае когда оптимальный размер тайла с точки зрения отношения объема коммуникаций к объему вычислений и (или) с точки зрения пропускной способности сети обменов данными намного превышает оптимальный размер тайла с точки зрения локальности данных.

Эффективный способ улучшения локальности данных — установление приоритетов очередности выполнения циклов и перестановка их (если это возможно) в соответствии с приоритетами. Определять оптимальный порядок вычисления циклов можно с помощью векторов локальности, введенных в разделе 3.4

3.5.4 Получение зерна вычислений

С учетом изложенного, выделим основные этапы разбиения операций исходного алгоритма на блоки и зерна вычислений.

1. Предварительное преобразование гнезда циклов (этот этап может отсутствовать). С точки зрения преобразования кода, разбиение

пространства итераций на блоки и тайлы есть разбиение каждой циклической конструкции на циклы большей глубины вложенности и перестановка циклов. Поэтому будем считать, что все другие преобразования циклов уже выполнены с помощью векторных функций $\bar{t}^{(\beta)}$.

2. Разбиение итераций алгоритма на блоки. Разбиение на блоки задают первые r $(r < n)$ координат функции $\bar{t}^{(\beta)} = (t_1^{(\beta)}, \ldots, t_n^{(\beta)})$. Размеры блоков $V_{\overline{p}}^{(\beta)}$ определяются размерами области V_β и числом процессоров $P_1 \times \ldots \times P_r$. Если какие-либо из функций $t_1^{(\beta)}, \ldots, t_n^{(\beta)}$ являются расщепляющими, то не требуется коммуникаций в направлении координат, определяемых такими функциями (но возможно потребуется дублирование входной информации).

3. Разбиение блоков на тайлы.

3.1. Положим $Q_1 = \ldots = Q_r = 1$. Это означает, что по первым r координатам размер блока и тайлов совпадают, по этим координатам разбиение блока на тайлы не производится. Если $Q_\eta = 1$ и есть коммуникации, определяемые координатой η, то процессор с координатой $p_\eta + 1$ не будет простаивать в ожидании данных от процессора с координатой p_η. Если координата η не определяет коммуникаций, то полагать $Q_\eta = 1$ не требуется. Если какая-либо из функций $t_\xi^{(\beta)}$, $\xi > r$, не является таймирующей, т. е. не сохраняет всех зависимостей алгоритма, то по координате с номером ξ разбиение произвести невозможно и следует положить $Q_\xi = 1$.

3.2. Установить приоритеты очередности выполнения циклов с помощью векторов локальности, переставить циклы в соответствии с приоритетами. По внутренним циклам размер тайла сделать максимальным. Число таких циклов определить с учетом размера тайла; размер тайла определить экспериментально, если не проводилось специальных исследований (задача 2 в разделе 1). Один из этапов 3.2, 3.3 может отсутствовать.

3.3. Пусть теоретически или экспериментально удалось установить оптимальные размеры тайла $C_1^l, \ldots, C_n^l$ с точки зрения локализации и оптимальные размеры тайла $C_1^e, \ldots, C_n^e$ точки зрения обмена данными. Если существуют такие η, $1 \leqslant \eta \leqslant n$, что $C_\eta^e \geqslant k \cdot C_\eta^l$, где $k \geqslant 2$, то произвести двухуровневое разбиение для этих η. Размер тайла первого уровня выбирается равным C_η^e, второго уровня — C_η^l.

3.5.5 Примеры разбиения циклов

При блокинге и тайлинге необходимо уметь преобразовывать цикл в двумерную циклическую конструкцию. При разбиении всех циклов гнездо циклов глубины n превращается в гнездо циклов глубины $2n$; внешние n циклов управляют порядком, в котором вычисляются тайлы вычислений, а внутренние n циклов управляют порядком, в котором вычисляются операции, принадлежащие одному тайлу.

Для применения такого тайлинга необходимо, чтобы записи гнезда циклов соответствовало n независимых таймирований. Таймирования $t_1, \ldots, t_n$ называются независимыми, если они удовлетворяют условию (1.11). Если все координатные функции многомерного таймирования являются t-функциями, то возможно разбиение всех циклов. Как уже отмечалось в предыдущем подразделе, если какая-либо из координатных функций не является таймирующей, то разбиение соответствующего цикла произвести невозможно.

Рассмотрим разбиение операций алгоритма (1.2) на тайлы размера $1 \times B$ для оператора S_1, и на тайлы размера $B \times B$ для оператора S_2 :

$$
\begin{array}{l}
\text{do } \ b1 = 1, \lceil N/B \rceil \\
\quad \text{do } \ it = (b1-1)B+1, \min(b1 \cdot B, N) \\
\quad\quad S_1 : \ c(it) = 0 \\
\quad \text{enddo} \\
\quad \text{do } \ b2 = 1, \lceil N/B \rceil \\
\quad\quad \text{do } \ it = (b1-1)B+1, \min(b1 \cdot B, N) \\
\quad\quad\quad \text{do } \ jt = (b2-1)B+1, \min(b2 \cdot B, N) \\
\quad\quad\quad\quad S_2 : \ c(it) = c(it) + a(it, jt)b(jt) \\
\quad\quad\quad \text{enddo} \\
\quad\quad \text{enddo} \\
\quad \text{enddo} \\
\text{enddo}
\end{array}
$$

Соответственно можно изобразить разбиение полного графа зависимостей (рис. 3.1, $N = 6$, $B = 2$).

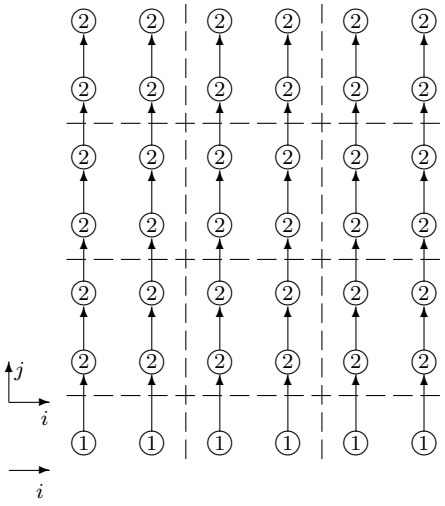

Рис. 3.1

Блочный вариант алгоритма (3.1) можно записать в виде

$$
\begin{array}{l}
\text{if } \ 1 \leqslant p \leqslant \lceil N/B \rceil \ \text{ then} \\
\quad \text{do } \ pt = (p-1)B + 1, \min(p \cdot B, N) \\
\qquad S_1 : \ c(pt) = 0 \\
\quad \text{enddo} \\
\quad \text{do } \ t = 1, \lceil N/B \rceil \\
\qquad \text{do } \ pt = (p-1)B + 1, \min(p \cdot B, N) \\
\qquad\quad \text{do } \ tt = (t-1)B + 1, \min(t \cdot B, N) \\
\qquad\qquad S_2 : \ c(pt) = c(pt) + a(pt, tt)b(tt) \\
\qquad\quad \text{enddo} \\
\qquad \text{enddo} \\
\quad \text{enddo}
\end{array}
$$

ЛИТЕРАТУРА

Воеводин, В. В. Параллельные вычисления / В. В. Воеводин, Вл. В. Воеводин СПб. : БХВ-Петербург, 2002. 600 с.

Антонов, А. С. Эффективная адаптация последовательных программ для современных векторно-конвейерных и массивно-параллельных супер-ЭВМ / А. С. Антонов, Вл. В. Воеводин // Программирование. 1996. № 4. С. 37–51.

Banerjee, U. An introduction to a formal theory of dependence analysis / U. Banerjee // J. Supercomput. 1988. № 2. P. 133–149.

Darte, A. Mathematical tools for loop transformations: from systems of uniform recurrence equations to the polytope model / A. Darte // Algorithms for Parallel Processing, IMA Volumes in Mathematics and its Applications. 1999. Vol. 105. P. 147–183.

Feautrier, P. Some efficient solutions to the affine scheduling problem. Part 1, 2 / P. Feautrier // Int. J. of Parallel Programming. 1992. Vol. 21, № 5, 6. P. 313–348, 389–420.

Lim, A. W. Maximizing parallelism and minimizing synchronization with affine partitions / A. W. Lim, M. S. Lam // Parallel Computing. 1998. Vol. 24, № 3, 4. P. 445–475.

Dion, M. Compiling affine nested loops: how to optimize the residual communications after the alignment phase? / M. Dion, C. Randriamaro, Y. Robert // J. of Parallel and Distrib. Computing. 1996. Vol. 30, № 2. P. 176–187.

Calland, P.-Y. Plugging anti and output dependence removal techniques into loop parallelization algorithms / P.-Y. Calland, A. Darte, Y. Robert, F. Vivien // Parallel Computing. 1997. Vol. 23, № 1, 2. P. 251–266.

Darte, A. On the optimality of Allen and Kennedy's algorithm for parallelism extraction in nested loops / A. Darte, F. Vivien // J. of Parallel Algorithms Applications. 1997. Vol. 12, № 1–3. P. 83–112.

Адуцкевич, Е. В. Необходимые и достаточные условия сохранения зависимостей при распараллеливании алгоритмов / Е. В. Адуцкевич, Н. А. Лиходед // Весці НАН Беларусі. Сер. фіз.-мат. навук. 2004. № 3. С. 100–105.

Лиходед, Н. А. Распределение операций и массивов данных между процессорами / Н. А. Лиходед // Программирование. 2003. № 3. С. 73–80.

Лиходед, Н. А. Получение аффинных преобразований для улучшения локальности гнезд циклов / Н. А. Лиходед, С. В. Баханович, А. В. Жерело // Программирование. 2005. № 5. С. 52–65.

Adutskevich, E. V. Affine transformations of loop nests for parallel execution and distribution of data over processors / E. V. Adutskevich, S. V. Bakhanovich,

N. A. Likhoded. Минск, 2005. 10 с. (Препринт / НАН Беларуси, Ин-т математики; № 3 (574)).

Адуцкевич, Е. В. Оптимизация обмена данными на параллельных компьютерах с распределенной памятью / Е. В. Адуцкевич, Н. А. Лиходед // Кибернетика и системный анализ. 2006. № 2. С. 166–182.

Адуцкевич, Е. В. Согласованное получение конвейерного параллелизма и распределения операций и данных между процессорами / Е. В. Адуцкевич, Н. А. Лиходед // Программирование. 2006. № 3. С. 54–65.

Лиходед, Н. А. Координатные преобразования многомерных циклов / Н. А. Лиходед, Е. В. Кишилов // Докл. НАН Беларуси. 2006. Т. 50, № 6. С. 42–47.