А К А Д Е М И Я Н А У К С С С Р
СИБИРСКОЕ ОТДЕЛЕНИЕ
А В Т О М Е Т Р И Я

№ 6 1983

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

УДК 519.68 : 681.32

М. К. ВАЛИЕВ, А. И. МИШИН
(Новосибирск)

ОРГАНИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ НА СИСТЕМЕ С ЛОКАЛЬНЫМИ ВЗАИМОДЕЙСТВИЯМИ ЭЛЕМЕНТОВ

Введение. В работах [1, 2] предложена асинхронная вычислительная система, содержащая кольцевую систему из процессоров, каждый из которых связан с кольцевой системой из запоминающих элементов. В этой системе, называемой в дальнейшем кольцевой вычислительной системой (КВС), одновременно может существовать несколько информационных потоков, используемых для доставки операндов и управляющих сигналов к процессорам; один из этих потоков движется по процессорному кольцу, а другие — по отдельным запоминающим кольцам (ЗК). Важным средством управления работой системы являются метки, записываемые в управляющие разряды слов, позволяющие управлять запуском и остановкой потоков, а также осуществлять выделение ключевых для обработки элементов по ассоциативному принципу (без прямой адресации процессоров и элементов памяти). Все взаимодействия как информационные, так и управляющие между элементами (процессорными и запоминающими) осуществляются по мере их готовности к взаимодействию. Таким образом, все виды длинных связей в КВС исключены, и время исполнения каждой элементарной операции определяется только собственными характеристиками элементов системы. В отличие от этого в системах, использующих глобальные взаимодействия процессорных или запоминающих элементов (например, общее тактирование или прямой доступ к памяти), время исполнения элементарной операции зависит также от размеров системы, и, следовательно, для определения фактического времени исполнения алгоритма требуется количество тактов работы умножить на весовой коэффициент, зависящий от размеров системы. Поэтому при оценке временной сложности алгоритмов обработки матриц порядка n известные оценки [3] нужно умножить на cn, где c — некоторая константа, а не на $\log n$, как это было при использовании логарифмического веса операций [3]. Аналогичные уточнения нужно делать и для параллельных алгоритмов [4]. Необходимость учета времени глобальных взаимодействий подчеркивается также в [5, 6].

В работе [1] приведено несколько примеров матричных алгоритмов, достаточно эффективно реализуемых на КВС. При этом для записи алгоритмов было использовано расширение языка АЛГОЛ операторами по взаимодействию процессов. Их семантика близка к механизму рандеву в языке АДА или взаимодействующих последовательных процессов Хоара [7]. Однако этот язык неудобен по нескольким причинам: 1) явное использование операторов по взаимодействию сильно усложняет программирование, понимаемость программ и проверку их корректности (уже в довольно простых программах из [1] имеются некоторые неточности, а для программ с более сложными взаимодействиями

это кажется неизбежным); 2) в программах существенно используется соединение процессоров в кольцо (в этом смысле язык машинно-ориентирован); 3) интерпретация языка на КВС затруднена, так как он ориентирован на память с произвольным доступом. В нашей работе предлагается язык МАК (матричный ассоциативный координатный), ориентированный на описание программ по матричной обработке в терминах глобальных (векторных, матричных) операторов, что позволяет исключить применение операций по взаимодействию и упрощает программирование (проблемы, связанные с корректностью взаимодействий, переносятся на этап интерпретации или компиляции). При этом ассоциативный принцип управления вычислениями реализован в языке введением шаблонов (обобщенных селекторов), позволяющих выделять фрагменты матриц, обрабатываемые особым образом. Шаблоны интерпретируются на КВС посредством вышеупомянутых управляющих меток. Предлагаемый язык близок к языку АПЛ, однако представляется нам концептуально более ясным, целенаправленным и удобным для программирования алгоритмов матричной обработки. Кажется также, что понятие шаблона может оказаться полезным и при разработке языков, работающих с более богатыми структурами данных.

Для интерпретации МАК на КВС используется промежуточный язык, предназначенный, как и язык из [1], для программирования отдельных процессоров, однако являющийся безадресным и расширенным средствами для работы с ЗК. Операторы по взаимодействию этого языка в отличие от средств языка АДА или Хоара [7] допускают весьма простую аппаратную реализацию (в частности, так как язык не содержит недетерминированных конструкций, снимаются проблемы с планированием выбора условий (guard sheduling)). Вообще говоря, для эффективной реализации МАК в полном объеме более приспособлена система с параллельным доступом к памяти и двунаправленной передачей как по строкам, так и по столбцам [2]. Однако и в простейшем варианте КВС может быть эффективно реализована большая часть языка. Заметим также, что на КВС эффективно выполняются и некоторые нечисленные алгоритмы (в частности, сортировка слиянием).

I. **Краткое описание КВС.** В работе используется вариант КВС, в котором каждый процессор связан с несколькими ЗК. Блок-схема процессорного элемента представлена на рис. 1. Он функционирует следующим образом. Из памяти программ (ПП) устройство управления (УУ) считывает очередную команду и в соответствии с ней устанавливает элемент либо на обработку информации, хранящейся в оперативном запоминающем устройстве (ОЗУ), арифметико-логическим устройством (АЛУ), либо на передачу (прием) информации посредством блоков $?i$ и $!i$ ($i = 1, \ldots, k$, где k — число соседних элементов) управления передачей (?) и приемом (!) информации. Один из регистров (R) в АЛУ выделен для

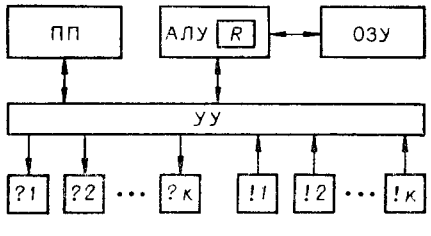

Рис. 1.

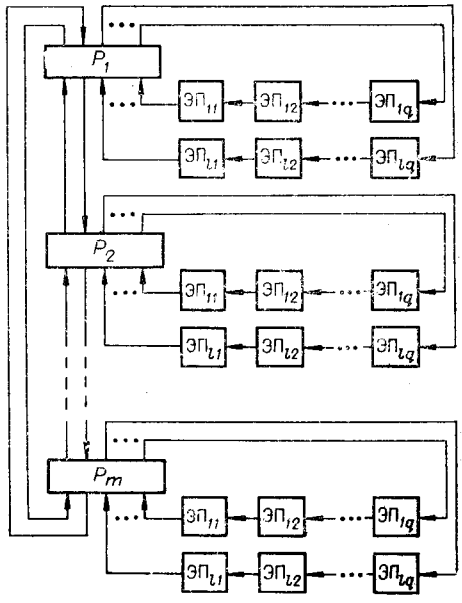

Рис. 2.

взаимодействий с соседними элементами (приема и передачи информации). Работа блоков $?i$ и $!i$ описывается соответственно программами

begin $\alpha_i := 1$; **waituntil** $\beta_i = 1$; $\alpha_i := 0$; **waituntil** $\beta = 0$; **end**

begin waituntil $\alpha_i = 1$; $R := R^{(i)}$; $\beta_i := 1$; **waituntil** $\alpha = 0$; $\beta := 0$; **end**.

Здесь R и $R^{(i)}$ — регистры для взаимодействия процессоров P и $P^{(i)}$, связанных i-м каналом, а команда „waituntil Q" означает ожидание выполнения условия Q. Предполагается, что в начале работы значения сигналов α_i и β_i для всех i равны 0.

Отметим, что для каждой команды приема (передачи) по i-му каналу компоненты программы, исполняемой процессором P, должна быть соответствующая команда передачи (приема) в компоненте программы, исполняемой процессором, связанным с P i-м каналом.

Элемент памяти ВС отличается от процессора тем, что он не содержит блоков управления и обработки, а имеет регистр, блок приема и блок передачи. Вход каждого из этих блоков соединен с выходом другого; предусмотрена также возможность начальной установки элемента на прием сигналом из процессора.

Структурная схема системы представлена на рис. 2, на котором для простоты показаны только информационные связи. КВС содержит m процессорных элементов $P_1, P_2, \ldots, P_m$, соединенных в кольцо. Процессор имеет l запоминающих колец (ЗК), содержащих цепочку из q элементов памяти $ЭП_{ij}$, выход которой через процессор соединен со входом. В общем случае разные ЗК могут иметь разную длину. По процессорному кольцу информация может передаваться в двух направлениях, а по ЗК — в одном. Каждое ЗК представляет собой циклический асинхронный конвейер по доставке данных (слов, записанных в элементах памяти, находящихся в состоянии передачи) к процессору. Взаимодействие между информационными потоками, движущимися по ЗК разных процессоров, осуществляется с помощью информационного потока, перемещающегося по процессорному кольцу. Например, вычитание строки (матрицы), хранящейся в ЗК процессора P_1, из всех остальных строк, записанных в ЗК других процессоров, можно выполнить следующим образом. Процессор P_1 принимает последовательно элементы a_{1j} из своего ЗК и передает их соседнему (например, правому) процессору. Процессор P_i, $i > 1$, после приема элемента a_{1j} от левого соседа принимает элемент a_{ij} из своего ЗК, передает $a_{ij} - a_{1j}$ в свое ЗК и элемент a_{1j} соседнему справа процессору (левый с P_1 процессор осуществляет передачу только в свое ЗК). Таким образом, рассматриваемая операция с последовательной временной сложностью n^2 выполняется за время $2n$.

КВС наиболее приспособлена для реализации алгоритмов, в которых данные разбиваются на линейные массивы (такие, как строки или столбцы матриц), обрабатываемые в их естественном порядке. При этом ЗК может содержать один или несколько таких массивов. Работу системы удобно представлять в виде циклов, в каждом из которых выполняется некоторая глобальная операция, связанная с однократным просмотром и обработкой содержимого одного или нескольких ЗК (например, исключение переменной в алгоритме Гаусса). Ясно, что, как правило, и внутри циклов имеются слова, обрабатываемые процессором различным образом. Для идентификации таких слов используются их управляющие разряды. Запись «1» в i-й разряд интерпретируется как пометка слова i-й меткой. В частности, для пометки последнего записанного в ЗК слова используется особая метка Δ. Заметим, что в некоторых случаях управляющие разряды слов удобней записывать в особых ЗК. Заметим также, что, если в каких-то случаях такого ассоциативного способа адресации окажется недостаточно, на КВС всегда можно промоделировать и прямую адресацию путем записи адресов в элементы памяти, однако это требует дополнительной памяти и снижает быстродействие.

Эффективность работы системы в значительной степени зависит от скорости получения процессором информации из ЗК. Для матричных алгоритмов эффективность работы с ЗК можно грубо оценивать суммар-

ным временем ожидания, которое тратит процессор на последовательный прием всех слов из ЗК и их передачу в ЗК после обработки. Пусть n — число слов, записанных в ЗК длиной q, δ — верхняя граница времени задержки элемента памяти, τ — нижняя граница задержки процессора, причем выполнено условие $2 \leqslant q/n \leqslant \tau/\delta$. Тогда довольно легко понять, что при первом цикле работы с ЗК (этап загрузки) слова в ЗК распределятся таким образом, что каждая операция обмена процессора с ЗК (передача или прием) будет выполняться без потери времени на ожидание (для передачи это обеспечивается условием $2 \leqslant q/n$, для приема — другой частью неравенства). Поэтому даже при фиксированной настройке длин ЗК КВС может решать задачи, довольно сильно отличающиеся по размерности, без потери эффективности работы с ЗК (так как τ значительно больше, чем δ, даже если не учитывать, что задержка процессора может быть вызвана выполнением длинных участков программ без обращения к ЗК).

Можно предложить также некоторые варианты или обобщения КВС с целью повышения ее эффективности на том или ином классе задач, в частности для случая решения систем дифференциальных уравнений методом расщепления [2].

II. МАК — матричный ассоциативно-координатный язык. 1. Приведем неформальное описание только некоторой базовой части языка, которая, однако, достаточно представительна и для многих практически важных матричных алгоритмов позволяет получить достаточно ясные и компактные программы (в конце раздела будет приведена программа решения систем линейных уравнений методом главных элементов; соответствующие программы на других известных языках, в том числе на АПЛ и функциональных языках, получаются довольно громоздкими). Кратко будут упомянуты также основные средства расширения языка.

В рассматриваемом варианте языка имеются три типа объектов: булевы значения (истина и ложь), шаблоны (конечные множества пар натуральных чисел) и массивы, определяемые как отображения шаблонов во множество действительных чисел. Таким образом, под массивом понимается любой фрагмент обычного двумерного массива. Такое расширение понятия массива вызвано тем, что при обработке матриц приходится выделять довольно разнообразные их фрагменты, обрабатываемые особым образом (например, исключаемую строку или столбец, диагональные, ненулевые или максимальные элементы и т. п.), которые удобно выделять с помощью шаблонов, представляющих по существу обобщенные селекторы. При этом значительная часть программного управления связывается непосредственно с данными. Эта связь весьма просто реализуется в КВС путем записи меток в управляющие разряды слов.

Заметим, что шаблоны можно понимать так же, как множества клеток целочисленной двумерной решетки $N \times N$ ($N = \{1, 2, \ldots\}$), первая строка и первый столбец которой образуют координатные оси с направлениями слева направо и сверху вниз. Скаляры (действительные числа) и векторы рассматриваются как частные случаи матриц (т. е. массивов, определенных на прямоугольных шаблонах вида $\{1, \ldots, m\} \times \{1, \ldots, n\}$), хотя можно ввести и соответствующие типы объектов.

В языке используются два вида переменных: над шаблонами с описаниями вида S (schablon) и над матрицами с описаниями вида M (matrix $[1{:}m, 1{:}n]$).

2. Синтаксис языка довольно прост и содержит стандартные управляющие средства структурного программирования: условные операторы, циклы while-do, until-do, блоки, нерекурсивные процедуры.

Используются два основных вида оператора присваивания:

1) $S := \text{exp}$, где S — шаблонная переменная, exp — выражение, задающее шаблоны, с естественно определяемой семантикой;

2) $M[\text{exp}_1] := \text{exp}_2$, где M — матричная переменная, exp_1 и exp_2 — выражения, задающие соответственно шаблоны Ш и массивы A (при этом предполагается, что текущее значение Ш выражения exp_1 содер-

жится в области определения текущего значения A выражения exp_2); выполнение присваивания состоит в том, что фрагмент матрицы M, выделяемый шаблоном Ш, заменяется соответствующим фрагментом массива A.

Присваивания такого рода, как «всем элементам массива присвоить фиксированное значение» или «массиву A присвоить с соответствующим сдвигом массив, область определения которого совпадает по форме с A», легко выражаются через операторы вида «2», но их, конечно, можно ввести и явно. Другая напрашивающаяся возможность расширения языка — это введение типов натуральных чисел и множеств натуральных чисел и связанных с этими типами операторов цикла **for** и групповых присваиваний такого рода, как «каждому элементу $A[i, j]$ (i-й строке или столбцу) некоторого множества присвоить значение выражения $exp(i, j)$ (или $exp(i)$)». Такие присваивания иногда более прямо отражают наше понимание алгоритма и имеют более привычный вид; однако если не ограничивать их вида, то возникают некоторые проблемы с их реализацией. Кроме того, в практически важных случаях такие операторы достаточно просто моделируются операторами выбранного здесь вида.

3. Выбор используемых стандартных функций, конечно, также неоднозначен, и приводимый в приложении список следует в основном рассматривать как ориентировочный.

III. **Интерпретация МАК на КВС.** 1. МАК может быть реализован на КВС как путем интерпретации, так и (исходя в основном из тех же принципов) путем компиляции. Отметим, что фактически интерпретация отдельных операторов сводится к их компиляции на некоторый промежуточный язык программирования работы отдельных процессоров. Однако для изложения выбрано описание интерпретации, так как оно проще, и, кроме того, для матричных алгоритмов компиляция не дает такого существенного, как обычно, повышения эффективности по сравнению с интерпретацией. При этом ограничимся приведением общих принципов интерпретации и нескольких показательных примеров интерпретации отдельных операторов.

Конечно, интерпретация зависит от структуры КВС (количества процессоров и ЗК) и расположения данных (элементов обрабатываемых матриц) в памяти. Для простоты рассмотрим случай, когда все обрабатываемые матрицы (исключая векторы) $M_1, M_2, \ldots, M_k$ (в том числе вспомогательные и не указанные явно в программе) имеют одинаковую размерность $m \times n$, система содержит m процессоров, каждый из которых связан с $(k+1)$-м ЗК длиной $2n$. i-е ЗК j-го процессора ($i = 1, \ldots, k$; $j = 1, \ldots, m$) используется для записи j-й строки переменной M_i (при этом считается, что разрядность регистров элементов памяти достаточна для хранения значений элементов матриц). Векторы записываются в памяти выделенного (главного) процессора. $(k+1)$-е ЗК процессоров (обозначаемое далее $ЗК_ш$) используется для записи значений шаблонных переменных: l-й разряд j-го слова i-го процессора содержит «1», если пара $\langle i, j \rangle$ принадлежит шаблону S_l. При другом варианте реализации шаблонов их значения записываются прямо в информационные ЗК, но иногда это оказывается невыгодным. Заметим, что для реально встречающихся программ количество как обрабатываемых матриц, так и шаблонов невелико.

Случай, когда число процессоров меньше числа строк матриц, в основном аналогичен, хотя и имеет некоторые особенности. Диагональный вариант расположения данных в памяти более сложен и подробно не рассматривался. Вариант запоминания матриц по столбцам двойствен рассматриваемому.

Следует заметить, что некоторые операторы языка более эффективно реализуются при работе системы по строкам, а другие — при работе по столбцам (например, операция цилиндрификации шаблона по строкам в первом случае выполняется всегда за линейное время, т. е. за один

оборот данных в ЗК, а во втором случае в зависимости от вида шаблона может потребовать от линейного до квадратичного времени). В этом смысле для эффективной реализации всего языка больше подходила бы система, имеющая память с доступом по строкам и столбцам (наиболее существенные преимущества такая система имеет при реализации алгоритмов, часто использующих транспонирование матриц) [2]. Однако для многих важных матричных алгоритмов операция транспонирования не является доминирующей (используется небольшое число раз или вовсе не используется), а другие операции, недостаточно эффективно реализуемые на простейшем варианте КВС, используются только для операндов, содержащихся в строке или столбце матрицы, и в этом случае выполняются эффективно (ниже будет рассмотрен соответствующий пример оператора присваивания $y := \mathrm{col}(x)$).

2. Существуют два варианта интепретации: 1) каждый из процессоров имеет свою копию исполняемой программы и интерпретирует ее сам; 2) интерпретируемая программа и полный интерпретатор содержатся только в одном (главном) процессоре. Рассмотрим второй вариант.

Работа системы разбивается на этапы. В начале каждого этапа главный процессор, анализируя интерпретируемую программу, вырабатывает очередной исполняемый МАК-оператор (присваивание или вычисление значения логического условия; при этом все операторы можно считать простейшими, т. е. не содержащими сложных выражений) и передает его по процессорному кольцу всем остальным (ведомым) процессорам. Каждый ведомый процессор содержит программу интерпретации отдельных МАК-операторов и, получив очередной исполняемый оператор, начинает его интерпретировать. Главный процессор также исполняет свою часть работы по интерпретации рассматриваемого оператора, закончив которую вырабатывает сигнал конца исполнения оператора, передает этот сигнал правому процессору и переходит к ожиданию сигнала «конца» от левого процессора. Получение им этого сигнала указывает на то, что исполнение оператора закончено и можно перейти к следующему этапу. Закончив исполнение оператора, каждый ведомый процессор переходит к ожиданию сигнала об окончании работы от левого процессора, а получив этот сигнал, транслирует его правому процессору и переходит к ожиданию оператора, исполняемого на следующем этапе.

При исполнении оператора проверки условия, наряду с передачей сигнала об окончании исполнения оператора, главный процессор передает правому соседу также сигнал о значении выработанного им локального условия, а каждый ведомый процессор передает своему соседу сигнал о значении условия с учетом принятого от левого соседа.

3. Для формального описания интерпретации операторов МАК полезно ввести некоторый промежуточный язык, предназначенный для описания взаимодействия отдельных процессоров и элементов ЗК. В этот язык введены специальные команды приема (!) и передачи (?), имеющие в основном такой же смысл, что и команды по взаимодействию в языках из [1, 7]. А именно, предполагается, что взаимодействие двух связанных между собой элементов осуществляется при одновременном соблюдении условия, когда один из элементов выполняет команду передачи, а другой — команду приема. В системе эти команды выполняются блоками по приему (!) и передаче (?) информации. Для последовательности команд !ЗК, ?ЗК, соответствующей циклическому сдвигу содержимого ЗК, используется сокращенное обозначение !?ЗК. С каждым шаблоном x сопоставляется некоторый управляющий разряд. Значение этого разряда регистра R обозначается R_x. Соседи слева и справа от процессора P обозначаются соответственно $P_л$ и $P_п$.

4. В качестве примера приведем интерпретацию оператора $G[S] := 0$, проверки условия непустоты шаблона a и оператора $y := \mathrm{col}(x)$, где шаблон x выделяет подмножество некоторой строки.

А. Интерпретация оператора $G[S] := 0$. Все процессоры работают независимо, считывая поочередно слова из $ЗК_ш$ и $ЗК_G$, в котором записа-

на строка матрицы G. В зависимости от наличия единицы в управляющем разряде слова, считанного из $ЗК_ш$ в $ЗК_ш$, происходит запись процессоры работают по программе $Пр_1$, причем i-й процессор проводит проверку условия $a_i \neq \varnothing$ (где a_i есть пересечение a с i-й строкой) для своего $ЗК_ш$ и результат фиксирует в виде значения переменной u_i. На втором этапе главный процессор работает по программе $Пр_2$, а остальные — по программе $Пр_3$. При этом осуществляется последовательное накапливание значения дизъюнкции всех локальных условий, и этап заканчивается приемом главным процессором результирующего сигнала.

Программы $Пр_1$, $Пр_2$, $Пр_3$ имеют следующий вид:

$Пр_1$: **until** $R_\Delta \neq 1$ **do** $!?ЗК_ш$; **if** $R_a = 1$ **then** $u := $ **true else** $u :=$ false; **od**.

$Пр_2$: $R := u$; $?P_п$; $!P_л$.

$Пр_3$: $!P_л$; $R := R \vee u$; $?P_п$.

В. Интерпретация оператора $y := \mathrm{col}(x)$. Этот оператор выполняется в три этапа.

На первом этапе (поиск ведущего) все процессоры работают независимо по программе $Пр_1$, выполняя поиск в шаблонном ЗК слова с единичным значением управляющего разряда, соответствующего шаблону x. Процессор, содержащий такое слово, работает дальше по программе $Пр_2$ (становится ведущим), а остальные — по программе $Пр_3$.

На втором этапе происходит выделение левого соседа (поглотителя) ведущего процессора, который в дальнейшем (на третьем этапе) выполняет программу $Пр_5$.

На третьем (основном) этапе ведущий исполняет программу $Пр_4$, поглотитель — программу $Пр_5$, а остальные процессоры — программу $Пр_6$. Заметим, что все программы $Пр_4$, $Пр_5$ и $Пр_6$ начинаются с установки своих $ЗК_ш$ в стандартное состояние, когда считанное слово содержит метку Δ начала (конца) строки. Во время этого этапа ведущий содержимое своего $ЗК_ш$ передает остальным процессорам. Все остальные процессоры, за исключением поглотителя, приняв слово от левого соседа, транслируют его вправо и в зависимости от значения разряда, соответствующего шаблону x, записывают «1» (при $R_x = 1$) в разряд, соответствующий шаблону y.

Программы $Пр_1 \div Пр_5$ имеют следующий вид:

$Пр_1$: **until** $R_x = 1 \vee R_\Delta = 1$ **do** $!?ЗК_ш$; **od**; **if** $R_x = 1$ **then** $Пр_2$ **else** $Пр_3$.

$Пр_2$: $R :=$ true; $?P_л$; $!P_п$; $Пр_4$.

$Пр_3$: $!P_п$; **if** $R =$ true **then** $R :=$ false; $?P_л$; $Пр_6$ **else** $R :=$ false; $?P_л$; $Пр_5$.

$Пр_4$: **until** $R_\Delta = 1$ **do** $!?ЗК_ш$; **od**; **until** $R_\Delta = 1$ **do** $!?ЗК_ш$; $?P_п$; **od**.

$Пр_5$: **until** $R_\Delta = 1$ **do** $!?ЗК_ш$; **od**; **until** $R_\Delta = 1$ **do** $!ЗК_ш$; $!P_л$; $?P_п$; **if** $R_x = 1$ **then** $R_y := 1$ **else** $R_y := 0$; $?ЗК_ш$.

Программа $Пр_6$ отличается от $Пр_5$ только тем, что исключена команда $?P_п$.

5. Легко видеть, что все рассмотренные операторы выполняются за линейное время (если длины ЗК удовлетворяют условию из р. 1). Можно показать, что это верно и для остальных операторов из программы „Gaussmain". Так как общее число исполняемых МАК-операторов имеет порядок n, то временная сложность этой программы оценивается как $0(n^2)$. Несколько более точные подсчеты дают верхнюю оценку $20n^2 + cn$. Величина константы c, хотя и зависит от деталей интерпретатора, тоже, конечно, не может быть слишком большой. Кроме того, если вместо интерпретации программы использовать ее компиляцию, то второй член в этой оценке обращается в нуль. Программирование алгоритма непосредственно в машинных командах дает программу с временной сложностью $5n^2$. Такого же рода соотношение между временной сложностью программ

на машинном языке и языке МАК имеется и для ряда других алгоритмов линейной алгебры. Это показывает, что эффективность реализации МАК на КВС достаточно высока.

При решении системы уравнений методом главных элементов существенную часть работы составляют вспомогательные операции, связанные с поиском ведущих элементов. Интуитивно ясно, что параллельный поиск ведущих элементов невозможен. Поиск каждого ведущего элемента занимает порядка n шагов. В связи с этим можно предположить, что вышеприведенная оценка временной сложности этого алгоритма не может быть существенно улучшена даже при использовании большего числа процессоров. Если это предположение верно, то оно показывает, что, несмотря на свою простоту и кажущуюся ограниченность (для решения системы с n неизвестными используется не более n процессоров), КВС при реализации некоторых важных алгоритмов оказывается не менее эффективной, чем любая другая (с произвольным числом процессоров) система. Заметим для сравнения, что клеточный автомат из [5] с n^2 процессорами, вычисляющий LU-разложение матрицы порядка n за линейное время (без учета времени ввода исходной информации), основан на последовательном делении на диагональные элементы и не работает, если хотя бы один из этих элементов оказывается равным 0.

ПРИЛОЖЕНИЕ

Функции над шаблонами.

Булевы ($\cup$, $\cap$, $\setminus$).

Проекции по строкам (prow) и столбцам (pcol): $\mathrm{prow}(S) = \{\langle 1, j\rangle \mid \exists i(\langle i, j\rangle \in S)\}$, $\mathrm{pcol}(S) = \{\langle i, 1\rangle \mid \exists j(\langle i, j\rangle \in S)\}$.

Цилиндрификация по строкам (row) и столбцам (col): $\mathrm{row}(S, M) = \{\langle i, j\rangle \in \mathrm{dom}(M) \mid \exists k(\langle k, j\rangle \in S)\}$. col определяется аналогично. Обычно массив M совпадает с обрабатываемой матрицей наибольшего размера, при этом параметр M будет опускаться. Кроме того, для выделения граничных строк или столбцов рассматриваемой матрицы $M[1{:}m, 1{:}n]$ будут употребляться краткие обозначения: $\mathrm{row}(m)$, $\mathrm{col}(n)$.

Декартово произведение линейных (строчного и столбцевого) шаблонов: $S_1 \times S_2 = \{\langle i, j\rangle \mid \exists k(\langle i, k\rangle \in S_1) \,\&\, \exists l \langle l, j\rangle \in S_2)\}$.

Функции left, right, upper, lower — выделения левых, правых, верхних или нижних элементов шаблона.

Функции shleft, shrigth, shup, shdown — циклических сдвигов шаблона (по модулю размерности матрицы).

Функция выделения диагональных элементов шаблона.

Функции над массивами.

Функция dom, дающая по массиву область его определения.

Циклические сдвиги и сдвиг массива A на шаблон той же формы, что и A.

Покомпонентные арифметические операции над массивами с одинаковой областью определения.

Транспонирование массивов.

Цилиндрификация униформных массивов по строкам (mrow) и столбцам (mcol): если в каждом столбце массива A содержится не более одного элемента, то $\mathrm{mcol}(A, M)$ совпадает с массивом A' таким, что $\mathrm{dom}(A') = \mathrm{col}(\mathrm{dom}\, A, M)$ и элементы столбцов A' совпадают с соответствующими элементами массива A.

Перестановки выделенных строк или столбцов.

Функции умножения и деления массива на числовую компоненту одноэлементного массива (на действительное число).

Выделение диагонального подмассива.

Выделение фрагмента массива A по шаблону S (обозначаемого $A[S]$ в соответствии с принятым обозначением элементов массива), являющегося сужением A на шаблон S.

Выделение из массива A фрагмента (обозначаемого $A[P]$), состоящего из элементов $A[i, j]$, для которых выполняется предикат $P(i, j, A[i, j], A)$. Не останавливаясь на языке описания предикатов, ограничимся некоторыми примерами.

Если $P(i, j, x, A) \equiv (x \neq 0)$, то $A[P]$ определяет подмассив массива A, состоящий из ненулевых элементов.

Если $P(i, j, x, A) \equiv$ "x является максимальным по абсолютному значению элементом A", то $A[P]$ определяет подмассив, состоящий из максимальных элементов A; в этом случае для шаблона dom $(A[P])$ используется сокращенное обозначение $\max A$.

Зависимость P от i, j также может быть полезной, например, для выделения границ фигур в задачах распознавания образов.

Возможно введение и некоторых операций, ориентированных на обработку матриц в диагональном направлении.

В качестве примера приведем программу решения систем линейных уравнений методом Гаусса — Жордана с выбором для исключения элемента с максимальным значением среди возможных. Следуя языку АДА, комментарии в программе помещены справа за знаком —.

Procedure Gaussmain (G: **matrix** $[1:n, 1:n+1]$):
last, lead, a, x, schablon;
begin — блок приведения к (квази) диагональному виду;
last $:=$ **col** $(n+1)$; — выделяется $(n+1)$-й столбец
$a :=$ **dom**$(G)\backslash$last; — и остальные элементы;
lead $:= \varnothing$; — множество исключенных переменных пусто;
while $a \neq \varnothing$ — цикл прямого хода;
do
$x :=$ **left upper max** $G[a]$; — выделение очередной исключаемой переменной (левой верхней среди возможных) и ее запоминание;
lead $:=$ lead $\cup\, x$;
$G[$**row**$(x)] := G[$**row**$(x)] \div G[x]$;
$G[\neg$**row** $(x)] := G -$ **mcol** $(G[$**row** $(x)]) \times$ **mrow** $(G[$**col** $(x)])$; — исключение переменной: из каждой строки без метки x вычитается строка с — меткой x, помноженная на соответствующий коэффициент;
$a := a\backslash($**row**$(x) \cup$ col$(x))$; — сокращение области поиска
od — исключаемой переменной;
end — конец блока приведения к диагональному виду;
while lead $\neq \varnothing$ — цикл последовательного вывода значений $x_1, \ldots, x_2$;
do
$x :=$ **left**(lead); — выделение;
output$(G[$last $\cap$ **row**$(x)])$; — и вывод значения очередной переменной;
lead $:=$ lead$\backslash x$; — исключение ее из списка;
od
end Gaussmain

При использовании переменных над натуральными числами, циклов вида for и операторов перестановки строк и столбцов можно получить немного более простую программу, в частности, можно исключить шаблоны last, a.

ЛИТЕРАТУРА

1. **Мишин А. И., Седухин С. Г.** Вычислительные системы и параллельные вычисления с локальными взаимодействиями.— В кн.: Вычислительные системы. Новосибирск: изд. ИМ СО АН СССР, 1979, вып. 78.
2. **Мишин А. И.** Параллельные вычислительные среды с локальными взаимодействиями элементов.— Автоматика и телемеханика, 1982, № 10.
3. **Ахо А., Хопкрофт Дж., Ульман Дж.** Построение и анализ вычислительных алгоритмов.— М.: Мир, 1979.
4. **Фадеева В. Н., Фадеев Д. К.** Параллельные вычисления в линейной алгебре.— Кибернетика, 1982, № 3.
5. **Mead C. A., Conway Z. A.** Introduction to VLSI systems.— Addison — Wesley, 1980.
6. **Chazelle B., Monier L.** Optimality in VLSI, VLSI 81, p. 269—278.
7. **Hoare C. A. R.** Communicating sequential processes.— Comm. of the ACM, 1978, vol. 21, N 8, p. 666—677.

Поступила в редакцию 10 октября 1982 г.; окончательный вариант — 3 мая 1983 г.