УДК 681.3.06:51

# ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ УМНОЖЕНИЯ МАТРИЦ

В.П.Толстых

В работе обсуждаются способы умножения матриц, которые могут быть эффективно реализованы на многопроцессорных вычислительных системах. Приводятся некоторые сравнительные характеристики рассмотренных алгоритмов.

Запись алгоритмов производится на языке высокого уровня APL [1], удобном для описания операций над векторами, матрицами, деревьями и другими крупными величинами информации.

Скалярные величины будем обозначать строчными латинскими буквами, векторы - подчеркнутыми строчными, а матрицы - подчеркнутыми прописными. Вектор-строка матрицы $\underline{A}$ обозначается $\underline{a}_{i\cdot}$, вектор-столбец - $\underline{a}_{\cdot j}$. Размерность вектора $\underline{a}$ обозначается $\nu(\underline{a})$, размерность матрицы $\underline{A}$ по строкам - $\mu(\underline{A})$, по столбцам - $\nu(\underline{A})$. Все допустимые бинарные операции над векторами и матрицами понимаются как покомпонентные. Дополнительные сведения об операциях и специальных величинах языка APL будут вводиться по мере необходимости. Отметим, что на языке APL операция умножения матриц записывается как частный случай о б о б щ е н н о г о м а т р и ч н о г о п р о и з в е д е н и я и выглядит следующим образом:

$$\underline{C} \leftarrow \underline{A} \overset{+}{\times} \underline{B} \longleftrightarrow c_{ij} \leftarrow +/(\underline{a}_{i\cdot} \times \underline{b}_{\cdot j}),$$

$$i = 1,2,\ldots,\mu(\underline{A}),\ j = 1,2,\ldots,\nu(\underline{B}),\ \nu(\underline{A}) = \mu(\underline{B}),$$

где $\underline{C}$ - результирующая матрица, $(i,j)$-й элемент которой определяется в результате покомпонентного умножения $i$-го вектора-строки матрицы $\underline{A}$ на $j$-й вектор-столбец матрицы

$\underline{B}$ и применения к полученному вектору операции р е д у к т и в н о г о  с л о ж е н и я .

Редуктивная операция выполняется путем последовательного применения ее ко всем компонентам вектора

$$c \leftarrow +/\underline{a} \leftrightarrow c \leftarrow (\ldots((a_1+a_2)+a_3)+\ldots+a_\nu).$$

Редуктивное сложение матрицы по строкам (столбцам) дает в результате вектор, компоненты которого получаются редуктивным сложением строк (столбцов) матрицы:

1) редуктивное сложение матрицы по строкам

$$\underline{c} \leftarrow +/\underline{A} \leftrightarrow c_i \leftarrow +/\underline{a}_{i\cdot}\ ,\ \nu(\underline{c}) = \mu(\underline{A}),$$

2) редуктивное сложение матрицы по столбцам

$$\underline{c} \leftarrow +//\underline{A} \leftrightarrow c_j \leftarrow +/\underline{a}_{\cdot j}\ ,\ \nu(\underline{c}) = \nu(\underline{A}).$$

В данной работе алгоритмы будут записываться в виде последовательности операций языка APL , которая удобна для реализации на многопроцессорной вычислительной системе. В дальнейшем для простоты изложения будем считать, что количество процессоров в системе позволяет производить умножение матриц, не прибегая к разбиению их на подматрицы.

I$^0$. В [2] описан способ умножения матриц, который используется в вычислительной системе ILLIAC $\overline{\text{IV}}$ . Этот способ реализуется на процессорах, настроенных в виде линейной вычислительной системы.

Для нахождения $i$-й строки результирующей матрицы $\underline{C}$ выбирается $i$-я строка матрицы $\underline{A}$ , ( $1 \leqslant i \leqslant \mu(\underline{A})$ ). Затем поочередно выбирается каждый $j$-й элемент этой строки ( $1 \leqslant j \leqslant \nu(\underline{A})$ ), и с помощью операции р а с п р о с т р а н е н и я из него формируется вектор-строка размерности $\nu(\underline{B})$ , каждая компонента которого равна выбранному элементу. Полученный вектор покомпонентно умножается на соответствующую $j$-ю строку матрицы $\underline{B}$ ( $1 \leqslant j \leqslant \nu(\underline{B})$ ). Образующиеся в результате каждой такой операции векторы покомпонентно складываются между собой. После обработки $\nu(\underline{A})$-го элемента выбранной строки матрицы $\underline{A}$ образуется редуктивная сумма столбцов матрицы, полученной в результате покомпонентного умножения $i$-й строки матрицы $\underline{A}$ на все столбцы матрицы $\underline{B}$ , т.е. $i$-я строка результирующей матрицы $\underline{C}$ . Описанная процедура выполняется до тех пор, пока не будет обработана

последняя строка матрицы $\underline{A}$ .

Выделение строк и столбцов матриц, а также выделение элементов векторов будем описывать с помощью операции с ж а т и е .

В результате сжатия произвольного вектора $\underline{a}$ логическим вектором $\underline{u}$ получается вектор $\underline{c}$ , содержащий только те компоненты вектора $\underline{a}$ , которые соответствуют $u_i = 1$ . Сжатие записывается как $\underline{c} \leftarrow \underline{u}/\underline{a}$ , где $\nu(\underline{u}) = \nu(\underline{a})$. Размерность результирующего вектора $\underline{c}$ равна числу единиц в векторе $\underline{u}$ .

Сжатие матрицы $\underline{A}$ логическим вектором $\underline{u}$ определяется

1) по строкам: $\underline{C} \leftarrow \underline{u}/\underline{A} \longleftrightarrow \underline{c}_{i\cdot} \leftarrow \underline{u}/\underline{a}_{i\cdot}$ $(i = 1,2,\ldots,\mu(\underline{A}))$,

т.е. результирующая матрица $\underline{C}$ содержит столбцы $\underline{A}$ , соответствующие $u_j = 1$ , а $\nu(\underline{C}) = +/\underline{u}$ ;

2) по столбцам: $\underline{C} \leftarrow \underline{u}\,//\underline{A} \longleftrightarrow \underline{c}_{\cdot j} \leftarrow \underline{u}/\underline{a}_{\cdot j}$ $(i = 1,2,\ldots,\nu(\underline{A}))$,

т.е. результирующая матрица $\underline{C}$ содержит строки $\underline{A}$ , соответствующие $u_i = 1$ , а $\mu(\underline{C}) = +/\underline{u}$ .

В качестве сжимающего логического вектора будет использоваться специальный логический вектор $\underline{\varepsilon}^i(n)$ , $i$-я компонента которого равна единице, а остальные $(n-1)$ компонент – нули.

Операцию распространения будем записывать:

1) для скаляра по строке $\underline{c}_{i\cdot} \leftarrow \Theta(n)\,a$ ,

2) для скаляра по столбцу $\underline{c}_{\cdot j} \leftarrow \Phi(n)\,a$ ,

3) для вектора по строкам $\underline{C} \leftarrow \Theta(n)\,\underline{a}_{\cdot j}$ ,

4) для вектора по столбцам $\underline{C} \leftarrow \Phi(n)\,\underline{a}_{i\cdot}$ ,

где $n$ – размерность результирующей величины по соответствующей координате.

С помощью введенных операций алгоритм описанного выше способа умножения матриц можно записать на языке APL следующим образом:

```
                  ⟨1⟩  i ← 1
                  ⟨2⟩  j ← 1
от ⟨12⟩ ⟶ ⟨3⟩  a_i· ← ε^i(μ(A))//A
от ⟨10⟩ ⟶ ⟨4⟩  b_j· ← ε^j(μ(B))//B
                  ⟨5⟩  a_ij ← ε^j(ν(A))/a_i·
```

```
                  ⟨6⟩  d ← θ(ν(B))a_ij
                  ⟨7⟩  e ← b_i. × d
                  ⟨8⟩  c_i. ← c_i. + e
                  ⟨9⟩  j ← j+1
к ⟨4⟩ ←           ⟨10⟩ j : ν(A)
                  ⟨11⟩ i ← i+1
к ⟨3⟩ ←           ⟨12⟩ i : μ(A)
                  ⟨13⟩     конец
```

Результирующая матрица $C$ формируется в ячейках $c_{1.} \div c_{n.}$.

В рассмотренном способе умножения матриц параллельной обработке подвергаются только векторные величины. При достаточно большом количестве процессоров или незначительных размерностях перемножаемых матриц более эффективными могут оказаться операции над матричными величинами.

$2^0$. В работе [3] предлагается структура вычислительной системы, реализующей операции языка APL, в которой предусмотрено специальное устройство для выполнения операций над матрицами. Умножение матриц в этой системе предлагается выполнять способом, аналогичным предыдущему, но используя операции над матричными величинами. Особенность этого метода заключается в том, что выбранная $i$-я строка матрицы $A$ транспонируется и распространяется на $\nu(B)$ столбцов. Полученная матрица покомпонентно умножается на матрицу $B$, а результат редуктивно складывается по столбцам, образуя $i$-ю строку матрицы произведения. Запишем алгоритм этого метода.

```
                  ⟨1⟩  i ← 1
                  ⟨2⟩  a_i. ← ε^i(μ(A))/A
                  ⟨3⟩  a_.j ← a_i.
                  ⟨4⟩  D ← θ(ν(B))a_.j
                  ⟨5⟩  F ← B × D
                  ⟨6⟩  c_i. ← +/F
                  ⟨7⟩  i ← i+1
                  ⟨8⟩  i : μ(A)
                  ⟨9⟩     конец
```

$3^0$. С целью экономии памяти вычислительной системы и в тех случаях, когда обращение к памяти в процессе выполнения алгоритма нежелательно, можно применять способ умножения матриц, при котором перемножаемые матрицы целиком находятся в параллельном матричном процессоре в течение всего времени их обработки. В данном случае необходимо произвести некоторые предварительные преобразования исходных величин. Если перемножаются матрицы, которые в общем случае имеют неквадратные формы, то их надо дополнить нулевыми строками или столбцами до квадратных матриц одинаковой размерности $(n \times n)$, причем

$$n = \max \{ \nu(\underline{A}), \mu(\underline{A}), \nu(\underline{B}), \mu(\underline{B}) \} .$$

Преобразованные таким образом матрицы будем обозначать $\underline{A}_0$ и $\underline{B}_0$. После этого матрица $\underline{A}_0$ транспонируется относительно главной диагонали. Покомпонентно умножив транспонированную матрицу $\underline{A}_0$ на матрицу $\underline{B}_0$ и проведя редуктивное сложение по столбцам, получим элементы главной диагонали результирующей матрицы $\underline{C}$. Затем производится ц и к л и ч е с к и й с д в и г матрицы $\underline{B}_0$ на одну позицию влево по строкам и выполняется следующий шаг алгоритма. Операция циклического сдвига в данном случае заключается в том, что все столбцы матрицы сдвигаются на одну позицию влево, причем крайний левый столбец занимает место крайнего правого. При таком способе выполнения умножения матриц элементы результирующей матрицы будут определяться в следующей последовательности:

1-й шаг – $C_{11}\ C_{22}\ C_{33}\ \ldots\ C_{(n-1)(n-1)}\ C_{nn}$

2-й шаг – $C_{12}\ C_{23}\ C_{34}\ \ldots\ C_{(n-1)n}\ \ C_{n1}$

3-й шаг – $C_{13}\ C_{24}\ C_{35}\ \ldots\ C_{(n-1)1}\ \ C_{n2}$

. . . . . .

$n$-й шаг – $C_{1n}\ C_{21}\ C_{32}\ \ldots\ C_{(n-1)(n-2)}\ C_{n(n-1)}$

Для того, чтобы после окончания операции элементы результирующей матрицы расположились в необходимом порядке, можно полученный на каждом шаге вектор элементов записывать в главную диагональ матрицы, в которой формируется результат, и после выполнения очередного шага производить циклический сдвиг содержимого этой матрицы на одну позицию влево по строкам.

Для описания рассматриваемого способа умножения матриц необходимы специальные векторы $\underline{\varepsilon}(n)$ – полный единичный

вектор размерности $n$, каждая компонента которого равна единице, и $\underline{\iota}^{0}(n)$ - интервальный вектор, представляющий собой отрезок натурального ряда чисел от 0 до $(n-1)$. Операция транспонирования матрицы относительно главной диагонали обозначается $\underline{C} \leftarrow \hat{\underline{A}}$. Операции циклического сдвига матриц записываются:

1) по строкам вправо - $\underline{C} \leftarrow \underline{k} \downarrow \underline{A}$,

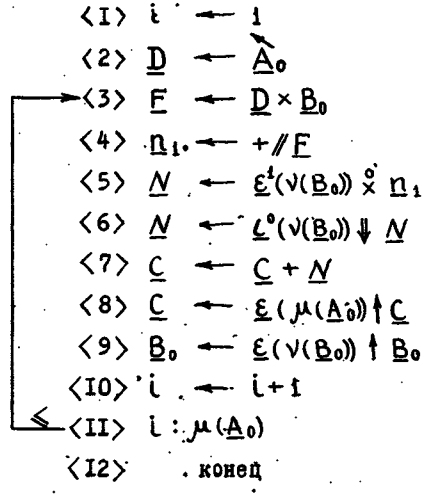
2) по строкам влево - $\underline{C} \leftarrow \underline{k} \uparrow \underline{A}$,

3) по столбцам вниз - $\underline{C} \leftarrow \underline{k} \Downarrow \underline{A}$,

4) по столбцам вверх - $\underline{C} \leftarrow \underline{k} \Uparrow \underline{A}$,

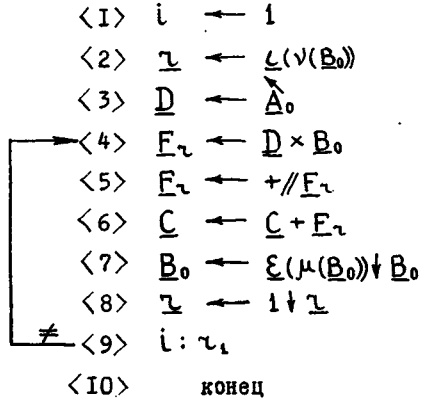
где $\underline{k}$ - целочисленный вектор, каждая компонента которого указывает, на какое число позиций сдвигается соответствующая строка (столбец) матрицы.

Алгоритм данного способа умножения матриц можно записать в виде такой последовательности операций:




Операции ⟨5⟩ ÷ ⟨8⟩ в приведённом алгоритме можно устранить, если снабдить параллельный матричный процессор индекс-регистром, состоящим из соединенной в кольцо цепочки ячеек, каждая из которых управляет соответствующим столбцом процессоров и может хранить числа от I до $\nu(\underline{B}_0)$. Число, хранящееся в ячейке индекс-регистра, показывает, в каком процессоре соответствующего столбца должен формироваться результат редуктив-

ного сложения на данном шаге алгоритма. Перед началом операции в индекс-регистр заносится интервальный вектор $\iota(\nu(B))$, начинающийся с единицы. Поэтому на первом шаге алгоритма значения элементов результирующей матрицы формируются в процессорах, находящихся на главной диагонали. В таком же порядке эти элементы переписываются в матрицу результата с помощью покомпонентной операции сложения. Затем производится кольцевой сдвиг содержимого индекс-регистра на одну позицию влево и выполняется следующий шаг алгоритма. Элементы результирующей матрицы в соответствии с показаниями индекс-регистра будут занимать правильные места в матрице результата. Такая модификация позволяет существенно упростить алгоритм рассматриваемого способа умножения матриц. Теперь он будет выглядеть следующим образом:



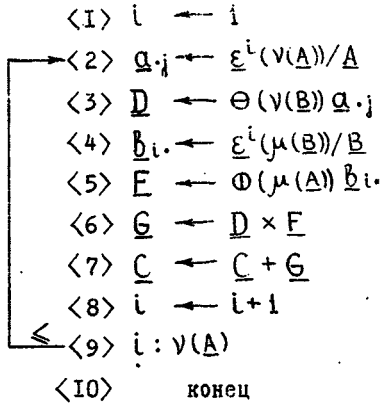

В приведенной записи алгоритма $\tau$ означает индекс-регистр, $F_\tau$ - управляемый матричный процессор.

$4^0$. Рассмотрим еще один способ умножения матриц, который заключается в последовательном накоплении значений элементов результирующей матрицы. Пусть $\nu(A) = \mu(B) = n$. Тогда каждый элемент результирующей матрицы находится как сумма произведений соответствующих пар элементов перемножаемых матриц:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad . \qquad (I)$$

На каждом шаге алгоритма этого способа умножения матриц определяются очередные слагаемые для всех элементов результирующей

матрицы, которые складываются с соответствующими ранее накопленными суммами. Для того, чтобы найти все слагаемые $k$-го шага, выбираются $k$-й столбец матрицы $\underline{A}$ и $k$-я строка матрицы $\underline{B}$. Первый из этих векторов распространяется на $\nu(\underline{B})$ позиций по строкам, а второй на $\mu(\underline{A})$ позиций по столбцам. Полученные матрицы покомпонентно перемножаются, в результате чего образуется матрица $k$-х членов для всех сумм вида (1). Алгоритм этого способа можно записать в виде последовательности операций:




В заключение сравним по быстродействию рассмотренные способы умножения матриц. Без большого ущерба для точности оценок будем считать, что перемножаются квадратные матрицы размерности $(n \times n)$. Такую же размерность должны иметь матричные параллельные процессоры. Чтобы сравнивать все алгоритмы при одинаковом количестве оборудования, будем рассматривать линейную вычислительную систему как $n$ линейных групп по $n$ процессоров в каждой, причем на каждой группе процессоров вычисляется одна строка результирующей матрицы. При такой организации линейной вычислительной системы для умножения матриц на каждой группе процессоров обрабатывается одна из строк матрицы $\underline{A}$ и все строки матрицы $\underline{B}$, т.е. внешний цикл алгоритма $1^0$ по $i$ распараллеливается, в результате чего этот способ умножения матриц по существу сводится к способу $4^0$.

В расчетах можно пренебречь временем выполнения вспомогательных операций таких, как распространение, сдвиг, выделение и других потому, что а) количество таких операций во всех

рассмотренных алгоритмах приблизительно равно, б) это очень быстрые операции, в) многие из них могут быть совмещены во времени с основными операциями - умножением и делением.

При введенных ограничениях алгоритмы $I^0$ и $4^0$ будут содержать, без учета вспомогательных операций, $n$ групповых операций умножения и $n$ групповых операций сложения. Каждая из таких операций состоит из $n^2$ одноименных элементарных операций, выполняемых одновременно. Обозначим время выполнения одной групповой операции умножения и сложения соответственно через $t(\times)$ и $t(+)$. Тогда полное время выполнения алгоритмов $I^0$ и $4^0$ будет равно:

$$T_{1,4} = n \cdot t(\times) + n \cdot t(+). \qquad (2)$$

Алгоритмы $2^0$ и $3^0$ также требуют выполнения $n$ аналогичных групповых операций умножения, поэтому время выполнения этих операций во всех алгоритмах одинаково и равно $n \cdot t(\times)$. Для выполнения операций сложения в алгоритмах $2^0$ и $3^0$ требуется больше времени, чем в оценке (2). Это вызвано тем, что на каждом шаге алгоритмов $2^0$ и $3^0$ необходимо выполнить непосредственно редуктивное сложение векторов, которое невозможно свести к одной групповой операции сложения. Эту операцию можно максимально распараллелить путем последовательного разбиения элементов редуктивно складываемых векторов и промежуточных сумм на пары слагаемых. Поскольку после каждого проведенного таким образом сложения число пар слагаемых уменьшается вдвое, то при размерности векторов, равной $n$, вся операция редуктивного сложения векторов потребует $\log_2 n^*$ групповых сложений, где $n^*$ — ближайшая большая или равная $n$ степень 2. Операцию редуктивного сложения векторов необходимо выполнять на каждом шаге алгоритмов $2^0$ и $3^0$. Следовательно, полное время выполнения этих алгоритмов равно:

$$T_{2,3} = n \cdot t(\times) + n(\log_2 n^*)\, t(+). \qquad (3)$$

Исходя из полученных оценок, можно сделать вывод, что наибольшее быстродействие достигается с помощью алгоритмов $I^0$ и $4^0$. Отметим, что в действительности время выполнения алгоритма $3^0$ будет несколько больше приведенной оценки (3), поскольку этот алгоритм требует значительных преобразований исходных матриц перед их умножением этим способом.

Наиболее серьезные практические трудности связаны с обработкой матриц, размерность которых не совпадает с размерностью параллельных процессоров. В этом случае встают такие вопросы, как простота разбиений исходных матриц, удобство обращения к промежуточным результатам, оптимизация загрузки процессоров и т.д. Поэтому основным требованием к алгоритмам становится их гибкость при решении перечисленных вопросов. Из рассмотренных нами алгоритмов несомненными преимуществами в этом отношении обладает первый алгоритм, т.к. линейная вычислительная система легко перестраивается на различное число групп процессоров, позволяя тем самым устранить простой оборудования. Кроме того, если матричные параллельные процессоры позволяют перемножать без разбиений матрицы размерностью $(n \times n)$, то линейная вычислительная система на таком же количестве процессоров позволяет без разбиений получить произведение матриц

$$\underline{C}_{(m \times n^2)} \leftarrow \underline{A}_{(m \times \ell)} \overset{+}{\times} \underline{B}_{(\ell \times n^2)},$$

где размерности $\ell$ и $m$ ограничиваются лишь длиной машинного слова и памятью процессоров. При этом надо учесть, что метод разбиения матриц на подматрицы в случае алгоритма $1^0$ наиболее прост. Однако, в тех случаях, когда операция умножения матриц является лишь одной из многочисленных операций алгоритма решения более сложной задачи, целесообразным может оказаться применение алгоритмов $2^0$ или $3^0$, т.к. каждый шаг этих алгоритмов дает окончательные значения элементов результирующей матрицы, которые можно сразу использовать для дальнейших вычислений.

## Литература


1. Iverson K.E. A programming language. New York,Wiley,1962.
2. McIntyre D.E. An introduction to the ILLIAC-IV computer. -Datamation,1970,16,N4,p.60-67.
3. Thurber K.J.,Myrna J.W. System design of a cellular APL computer. -IEEE Trans. on Comp.,1970,C-19,N4,p.291-301.



Поступила в ред.-изд.отдел
3 апреля 1972 г.