1991年5月 重庆大学学报 Vol. 14, №. 3
第14卷第3期 JOURNAL OF CHONGQING UNIVERSITY May. 1991

一种只需$4N^2$单元的N阶矩阵快乘法及其实现*

A FAST N×N MATRIX MULTIPLICATION ALGORITHM WHICH ONLY N EEDS $4N^2$ CELLS AND ITS REALIZATION

周六丁 程代杰
Zhou Luding Chen Daijie
(计算机系)

摘 要 本文改进了V. Strassen矩阵快乘算法，在时间复杂性保持相同，但将其空间复杂性从$O(N^{2.81})$降至$4N^2$。文中还给出了改进算法的实现技术。

关键词 算法复杂性；算法分析；数据结构 / 矩阵乘法
中图法分类号 TP301.6

ABSTACT This paper improves V. Strassen' s fast matrix multiplication algorithm, reducing its storage complexity from $O(N^{2.81})$ to $4N^2$, and discuses about realization of improved algorithm.

SUBJECT WORDS algorithm complexity; algotrithm analysis; data structure / matrix multiplication

0 引 言

科学、工程中许多问题可归结为矩阵乘法，因此研究矩阵快乘算法具有重要意义[1,2]。V. Strassen曾提出一个有实用价值且耗时仅为$O(N^{2.81})$的N阶矩阵快乘算法(常规法耗时为$O(N^3)$)。当N很大时，该算法比常规法快许多倍。但它有致命的弱点，即耗费空间太多，它完成N阶矩阵乘需$(11/3)\cdot N^{2.81}-\frac{8}{3}N^2$单元(常规法仅需$3N^2$单元)。本文改进了V. Strassen算法，在耗时几乎相等情况下却将其空间需求量降至$4N^2$单元。为便于实现(这也是V. Strassen算法的弱点之一)，文中给出了实现的技术。值得一提的是本文给出的节省空间的方法具有一定的普遍性。

1 Strassen算法及其时、空复杂性[1]

对于两个2阶矩阵相乘，即：

$$A=\begin{pmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{pmatrix},B=\begin{pmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{pmatrix},C=\begin{pmatrix}c_{11} & c_{12}\\ c_{21} & c_{22}\end{pmatrix}=A.B;$$

用常规法计算C需8次乘、4次加，而用Strassen算法需7次乘、18次加。其算法如下：

* 收文日期 1990-04-11

$$\left.\begin{aligned}
&P=(a_{11}+a_{22})(b_{11}+b_{22});Q=(a_{21}+a_{22})b_{11};\\
&R=a_{11}(b_{12}-b_{22});S=a_{22}(b_{21}-b_{11});\\
&T=(a_{11}+a_{12})b_{22};U=(a_{21}-a_{11})(b_{11}+b_{12});\\
&V=(a_{12}-a_{22})(b_{21}+b_{22});\\
&c_{11}=P+S-T+V;c_{12}=R+T;\\
&c_{21}=Q+S;c_{22}=P+R-Q+U.
\end{aligned}\right\}\qquad(1)$$

该算式可推广到计算两个2n 阶矩阵，只需把 $a_{ij},b_{ij},c_{ij}(i,j=1,2)$ 看作 n 阶块阵即可。计算 $2^n=N$ 阶矩阵相乘的 Strassen 快速算法通过递推计算式(1)而形成。

若令 P_n、A_n、S_n 分别为用 Strassen 算法完成 $2^n=N$ 阶矩阵相乘所需的乘、加法次数及存储单元数，由式(1)可建立递推方程[4]：

$$\begin{cases}P_n=7P_{n-1}\\P_1=7\end{cases}$$

$$\begin{cases}A_n=7A_{n-1}+18(2^{n-1})^2\\A_1=18\end{cases}$$

$$\begin{cases}S_n=7S_{n-1}+2(2^n)^2\\S_1=15\end{cases}$$

求解可得：

$$P_n\doteq N^{2.81};A_n\doteq 6N^{2.81};S_n\doteq\frac{11}{3}N^{2.81}-\frac{8}{3}N^2\qquad(2)$$

从分析结果可知，只有当 N 较大时，Strassen 算法才比常规 算法快 若干倍，但这时 Strassen 算法的空间耗费又难以容忍，因此有必要降低其空间复杂性。

2　一种只需$4N^2$单元的 N 阶矩阵快乘算法

若我们注意到这样一个简单而重要的事实，即：给定问题(N 阵矩阵相乘)的一个子问题(如式(1)中的 P，Q，R，…，V)一经求解，便可立即根据该子问题的计算结果对给定问题的求解作部分贡献(如将 P 累加到 c_{11}、c_{12} 中)，然后将已求解子问题占用的空间转用于求解其它兄弟子问题，并且将这种思想递归使用，则可对 Strassen 算法进行改进。其改进算法如下。

Algorithm 1　一种只需$4N^2$单元的 N 阶矩阵快乘算法(递归算法)

```
Procedure Multiply(A,B,C,N)
//计算 C=A·B,A、B、C 均为 N=2^n 阶矩阵,调用时 C 赋全0,结束时为结果//
begin
  定义三个 N/2×N/2 的局部数组 W1(1:N/2;1:N/2),W2(1:N/2;1:N/2);W3(1:N/2;1:N/2);
  if N=2 then //求解2阶矩阵相乘,方法同式(1)//
    begin
      p=(A11+A22)·(B11+B22);  q=(A21+A22)·B11;  r=A11(B12-B22);
      s=A22·(B21-B11);  t=(A11+A12)·B22;  u=(A21-A11)(B11+B12);
      v=(A12-A22)(B21+B22);
      C11=p+s-t+v;  C12=r+t;  C21=q+s;  C22=p+r-q+u;
    end
```

```
else   //递归求解7个子问题，每求解一个子问题便对原给定问题的求解作出部分贡
       献，并将其占用的空间转用于其它兄弟子问题的求解//
begin
  W1=(A11+A22);W2=(B11+B22); //将A中分块阵A11、A22相加后赋给W1，将B中分
                              块阵B11、B22相加后赋给W2//
  W3=0;                      //W3赋全0//
  Call Multiply (W1,W2,W3,N/2);//递归求解P子问题(参见式(1))//
  C11=C11+W3;C22=C22+W3;     //将P子问题的解W3对原问题的求解作部分贡献//
  W1=A21+A22;W2=B11;W3=0;    //递归求解Q子问题，并将其解W3对原问题的求
                              解作部分贡献//
  Call Multiply (W1,W2,W3,N/2);
  C21=C21+W3;C22=C22-W3;
  W1=A11;W2=(B12-B22);W3=0;//递归求解R子问题，并对原问题作部分贡献//
  Call Multiply (W1,W2,W3,N/2);
  C12=C12+W3;C22=C22+W3;

  W1=A22;W2=(B21-B11);W3=0;//递归求解S子问题，并对原问题作部分贡献//
  Call Multiply (W1,W2,W3,N/2);
  C11=C11+W3;C21=C21+W3;

  W1=(A11+A12);W2=B22;       W3=0;//递归求解T子问题，并对原问题作部
                                  分贡献//
  Call Multiply (W1,W2,W3,N/2);
  C11=C11-W3;C12=C12+W3;

  W1=(A21-A11);W2=(B11+B12);W3=0;//递归求解U子问题，并对原问题作部分
                                  贡献//
  Call Multiply (W1,W2,W3,N/2);
  C22=C22+W3;

  W1=(A12-A22);W2=(B21+B22);W3=0;//递归求解V子问题，并对原问题作部分
                                  贡献//
  Call Multiply (W1,W2,W3,N/2);
  C11=C11+W3;
end

end of procedure
```

现对 Algorithm 1的时、空复杂性进行分析。

令 P_n、A_n、S_n 分别为 Algorithm 1完成两个 $N=2^n$ 阶矩阵相乘所需的乘、加法次数及存储单元数，则可得下面三组递推方程：

$$\begin{cases} P'_n = 7P_{n-1} \\ P'_1 = 7 \end{cases}$$

$$\begin{cases} A'_n = 7A_{n-1} + 22(2^{n-1})^2 \\ A'_1 = 18 \end{cases}$$

$$\begin{cases} S'_n = 3 \times (2^n)^2 + S_{n-1} + C \\ S_1 = 19 \end{cases}$$

其中第三组递推方程中的c每级为保存返回地址用的栈单元数(它很小)。

其解分别为：

$$\left.\begin{aligned} P'_n &= 7^n \doteq N^{2.81} \\ A'_n &= \frac{22}{3}(7^n - 4^n) - 4 \cdot 7^{n-1} \doteq 7.1N^{2.81} \\ S'_n &= 7 + 4(2^{2n} - 1) + n \cdot c \doteq 4N^2 \end{aligned}\right\} \quad (3)$$

将式(3)的结果与Strassen算法相应指标进行比较可看出：对于两个N阶($N=2^n; n\in Z^+$)矩阵相乘，Algorithm 1与Strassen算法的时耗几乎相等(因乘法较加法耗时多)，但前者仅需$4N^2$单元。仅这一小的改进将大幅度削减快速矩阵乘法的空间需求量。如$N=10^4$左右时，可计算出Strassen算法所需空间为该算法的1600倍左右。

3　Algorithm 1的实现技术

Algorithm 1与Strassen算法在实现方面都这存在一些困难，这就是如何消除递归产生非递归算法。

图1形象地说明了Algorithm 1在执行过程中如何自顶向下分解及自底向上求解(合并)。Algorithm 1能节省空间的原因在于：一个子问题一经求解便立即将其解对其父问题求解作部分贡献，随后将它占用的空间用于求解其它兄弟子问题。应注意，这种思想在这棵AND树[5]中每一层的问题求解中都将用到。因此我们所需的全部存储空间仅是从根到叶的一条通路上的问题(及子问题)结点所需存储空间之和。为便于实现，用图2所示的结构来保存一条通路n个($n=\log_2 N$)结点所需的数组。

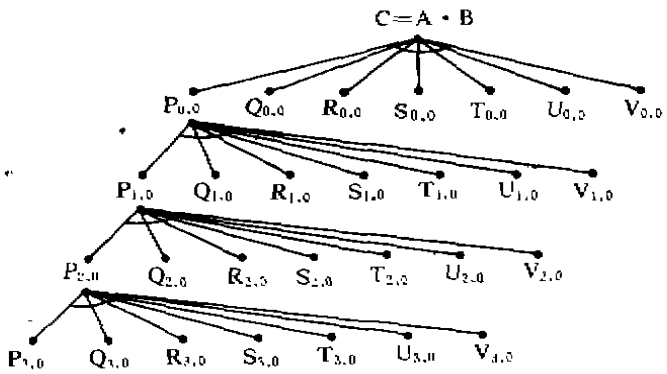

图1　Algorithm 1求解过程的AND("与")树表示(N=32)

在图2所示的$2N\times 2N$($N=32$)的大数组中，用最大三块(每块为$N\times N$)分别保存根结点问题所需的数组A、B、C(它们依次在左下、右上和左上角)，用次大的三块(每块为$\frac{N}{2}\times\frac{N}{2}$)保存第二层的问题结点所需的三个数组(保存方法与根结点情形类似)，

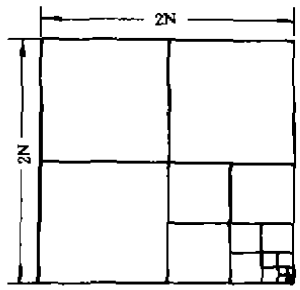

图2　图1中从根到叶通路上各问题所需数组存放图(N=32)(阴影小块不用)

……，用最小三块保存叶结点层问题所需的三个数组。

递归算法 Algorithm 1的执行过程等价于在图1所示的 AND 树中按先序遍历[5]的次序求解各子问题和给定的问题。因此，需设置一个栈 Stack(1:7n,1:3)用于保存那些待求解的子问题。该栈中每行的三个单元分别表示一个子问题的大小、名字(说明它是其父问题的 P、Q、R、S、T、U、V 子问题中哪一个)以及它是否已求解的标志(solved 标志)。

非递归算法一开始将根结点的7个子问题压入栈内(压入顺序是 $V_{0,0}$、$U_{0,0}$、$T_{0,0}$、$S_{0,0}$、$R_{0,0}$、$Q_{0,0}$、$P_{0,0}$)，然后循环检查栈顶子问题的情况，并作相应处理(三种情况之一)。

情况一：若栈顶子问题是原子问题，则先从其父问题所使用的数据区中复制相应的数据到该子问题使用的数据区中，然后求解，将其求解结果累加到其父问题的解答中，再从中栈消去该子问题。若消去的子问题是其父的最后一个子问题，还需将新栈顶子问题的 solved 标志置1。

情况2：若栈顶子问题已得解，则将该子问题的解累加到其父问题的解中，随后在栈中消去它。若它是其父的最后一个子问题，还需将新栈顶子问题的 solve 标志置1。

情况3：若栈顶子问题既未求解又非原子，则先从其父问题使用的数据区是复制相应数据到该子问题的数据区中，然后将该子问题的7个子问题压入栈内(按 V'、U'、T'、S'、R'、Q'、P' 次序压入)。

上述循环过程反复执行，直到栈为空时为止。这时根(原)问题得解。

由于篇幅所限，实现的细节不便叙述，但有了上述的思想，数据结构和算法骨架，则能编出相应的算法及程序。目前，这种只需$4N^2$单元的矩阵快乘算法已在 Dual 83/20机上实现，当 N＝1024时，新算法比常规算法快2倍左右，但空间耗费多了1/4左右。

参 考 文 献

1 卢开澄. 组合数学—算法与分析(下). 北京：清华大学出版社，1983，144～149

2 Aho Hopcroft et al. The Design and Analysis of Computer Algorithms. California: Addision—Wesley, 1976

3 游兆永. 线性代数与多项式的快速算法. 上海：上海科技出版社，1985

4 周振黎，康泰. 组合数学. 重庆：重庆大学出版社，1986，60～65

5 邹海明，余祥宣. 计算机算法基础. 武汉：华中工学院出版社，1985. 154～158