International Journal of Recent Engineering Research and Development (IJRERD)
ISSN: 2455-8761
www.ijrerd.com || Volume 08 – Issue 08 || August 2023 || PP. 01-03

A Note on the Multiplication of Strictly Upper Triangular Matrices

Scott McDermott[1], Ralph P. Tucci[2]
Department of Mathematics and Computer Science, Loyola University New Orleans
6363 St. Charles Avenue, Box 35, New Orleans, LA 70118, USA

Abstract: We present an improved algorithm to multiply strictly upper triangular matrices.
Keywords: Upper triangular matrices.

1. Introduction

Throughout this paper R will denote a ring of strictly upper triangular $n \times n$ matrices. We will develop an algorithm to multiply such matrices using fewer multiplications than other known matrix multiplication algorithms. We first describe three previously known algorithms and then describe and compare an improved algorithm.

Algorithm 1.1 Naive Algorithm: Multiply each row times each column. This method requires exactly n^3 multiplications.

Algorithm 1.2 Strassen's Algorithm: The fastest known algorithm to multiply square $n \times n$ matrices is that of Strassen [1, 2]. This algorithm requires $O(n^{\log_2 7})$ multiplications and is optimal only for large matrices due the overhead required for processing the inputs. Unfortunately, this algorithm does not speed up multiplication of upper triangular matrices [1].

Algorithm 1.3 Naive Upper Triangular Algorithm: This algorithm simply allows us to ignore multiplications involving 0 below the diagonals. Let A and$B$ be upper triangular $n \times n$ matrices and let $C = AB$. The number of multiplications $s(k)$ needed to compute column k in C is exactly

$$s(k) = \sum_{j=1}^{k} j = \frac{k(k+1)}{2} \tag{1}$$

Table 1 illustrates the number of multiplications per column for this algorithm.

Table 1: Number of Multiplications per Column

k	Elements in column k	Number of multiplications s(k)
1	$c_{1,1} = a_{1,1}b_{1,1}$	s(1) = 1
2	$c_{1,2} = a_{1,1}b_{1,2} + a_{1,2}b_{2,2}$ $c_{2,2} = a_{2,2}b_{2,2}$	s(2) = 2 + 1 = 3
3	$c_{1,3} = a_{1,1}b_{1,3} + a_{1,2}b_{2,3} + a_{1,3}b_{3,3}$ $c_{2,3} = a_{2,1}b_{1,3} + a_{2,2}b_{2,3}$ $c_{3,3} = a_{1,3}b_{3,3}$	s(3) = 3 + 2 + 1 = 6

In general, to compute column k in this product we need to perform $s(k)$ multiplications.Hence the total number of multiplications which this algorithm requires is exactly

$$\sum_{k=1}^{n} s(k) = \frac{n(n+1)(n+2)}{6} \tag{2}$$

This formula can be verified by a straightforward induction argument.

Algorithm 1.4 Improved Upper Triangular Algorithm: Below we describe a novel algorithm which requires fewer multiplications than the algorithms described above. The number of multiplications which this new algorithm requires is exactly

$$\frac{n(n-1)(n-2)}{6} \tag{3}$$

2. An Improved Algorithm

Note that in Algorithm 1.3 we avoid multiplying by any element below the main diagonal, since all the elements below the main diagonal are 0. We can speed up this algorithm for strictly upper triangular matrices by not multiplying by the first column and last row, both of which consist entirely of 0. In fact, as we shall see, we can actually eliminate two rows and two columns from each of the matrices being multiplied.

The basic idea is as follows. When we multiply matrices A and B, we reduce A and B to smaller matrices by eliminating those rows and columns whose products result in 0. We then compute the product C of these smaller matrices and use C to construct AB. This algorithm also allows us to calculate BA.

Notation: When we compute AB, we reduce A to $A_L$and B to B_R. When we compute BA, we reduce B to $B_L$and A to A_R. The subscripts L and R stand for “left” and “right”, respectively. We now present our algorithm.

Algorithm 1.4. Improved Upper Triangular Algorithm

Construct A_L.

1. *Eliminate the first column of* A. Every element in the first column of A is 0, so any product involving these elements is 0.
2. *Eliminate the last row of* A. Every element in the last row of A is 0, so any product involving these elements is 0.
3. *Eliminate the rightmost column of* A. The elements in the rightmost column of $A$are of the form $a_{j,n}$ and the elements $b_{n,k}$ in B are all 0.
4. *Eliminate row* $n - 1$ *of* A. Row $n - 1$ of A has only one possible non-zero element, namely $a_{n-1,n}$ and the elements $b_{n,k}$ in B are all 0.

Construct A_R.

1. *Eliminate the first column of* A. Every element in the first column of A is 0, so any product involving these elements is 0.
2. *Eliminate the last row of* A. Every element in the last row of A is 0, so any product involving these elements is 0.
3. *Eliminate the first row of* A. The elements in the first row of A are of the form $a_{1,j}$and the elements $b_{k,1}$ in B are all 0.
4. *Eliminate the second column of* A. The second column of A has only one possible non-zero element, namely, $a_{1,2}$, and the elements $b_{k,1}$ in B are all 0.

Construct AB.
AB consists of the $n \times n$ matrix with $A_L B_R$ in the upper right corner and 0 elsewhere.

Example 2.1

$$\text{Let } A = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 0 & 4 & 5 \\ 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{ and } B = \begin{bmatrix} 0 & 10 & 22 & 1 \\ 0 & 0 & 9 & 8 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\text{Then } AB = \begin{bmatrix} 0 & 0 & 9 & 14 \\ 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{ and } BA = \begin{bmatrix} 0 & 0 & 40 & 182 \\ 0 & 0 & 0 & 54 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

To compute AB, Algorithm 1.4 performs the following steps.

1. Construct $A_L = \begin{bmatrix} 1 & 2 \\ 0 & 4 \end{bmatrix}$ and $B_R = \begin{bmatrix} 9 & 8 \\ 0 & 3 \end{bmatrix}$.
2. Compute $A_L B_R = \begin{bmatrix} 9 & 14 \\ 0 & 12 \end{bmatrix}$.
3. The product AB consists of the 4×4 matrix with $A_L B_R$ in the upper right corner and 0 elsewhere. This yields AB.

To compute BA, Algorithm 1.4 performs the following steps.

1. Construct $B_L = \begin{bmatrix} 10 & 22 \\ 0 & 9 \end{bmatrix}$ and $A_R = \begin{bmatrix} 4 & 5 \\ 0 & 6 \end{bmatrix}$.
2. Compute $B_L A_R = \begin{bmatrix} 40 & 182 \\ 0 & 54 \end{bmatrix}$.
3. The product BA consists of the 4×4 matrix with $B_L A_R$ in the upper right corner and 0 elsewhere. This yields BA.

We summarize our result in the following proposition.

Proposition 2.2

1. The number of multiplications which Algorithm 1.4 requires isexactly

$$\sum_{n=1}^{n-2} s(k) = \frac{n(n-1)(n-2)}{6} \tag{4}$$

2. Algorithm 1.4 requires n^2 fewer multiplications than Algorithm 1.3.

Proof 2.2

1. The formula for the number of multiplications which Algorithm 1.4 requires is the same as Algorithm 1.3, except that we reduce the dimension of the matrices which we multiply from n to $n-2$.
2. The number of multiplications which Algorithm 1.4 saves isexactly

$$s(n) + s(n-1) = \frac{n(n+1)}{6} + \frac{n(n-1)}{6} = n^2 \tag{5}$$

Example 2.3

In Example 2.1 we see that Algorithm 1.3 requires 20 multiplications, while Algorithm 1.4 requires 4 multiplications. The number of savings is $16 = n^2$.

3. Conclusion

We summarize our results in the following table, where we list the number of multiplications used in each algorithm.

Table 2:Comparison of Multiplications per Algorithm

Algorithm 1.1 Naïve	**Algorithm 1.2 Strassen**	**Algorithm 1.3 Naive Upper Triangular**	**Algorithm 1.4 Improved Upper Triangular**
n^3	$O(n^{\log_2 7})$	$\frac{n(n+1)(n+2)}{6}$	$\frac{n(n-1)(n-2)}{6}$

In summary, even though Algorithm 1.4 and Algorithm 1.3 are both $O(n^3)$, Algorithm 1.4 saves n^2 multiplications.

References

[1]. Ruder, Jack, Alternatives to the Naive Algorithm for Matrix Multiplication, Semester Project for Advanced Linear Algebra (Math 390), University of Puget Sound, Tacoma, WA., April 25, 2021.

[2]. Strassen, Victor, Gaussian elimination is not optimal, Numerische Mathematik, 13, pp. 354–356.