Matrix Multiplication and Fast Fourier Transforms

By W. MORVEN GENTLEMAN

(Manuscript received January 29, 1968)

Factoring a matrix and multiplying successively by the factors can sometimes be used to speed up matrix multiplications. This is, in fact, the trick which creates the fantastic gains of the fast Fourier transform.

The same trick which creates the fantastic gains of the fast Fourier transform may be used with other matrices.

As an example, suppose the matrix

$$\begin{bmatrix} 1 & -10 & 4 & 3 & -14 & 12 \\ -5 & 2 & -20 & -7 & 6 & -28 \\ 2 & -20 & 1 & 6 & -28 & 3 \\ -20 & 1 & -10 & -28 & 3 & -14 \\ 4 & -5 & 2 & 12 & -7 & 6 \\ -10 & 4 & -5 & -14 & 12 & -7 \end{bmatrix}$$

is to be multiplied by a large number of different vectors, so that it is worthwhile to try to be as efficient as possible. At first glance, it would appear that (neglecting the possibility that multiplications by one might not actually be performed) multiplying this matrix with a single column vector would require $6^2 = 36$ multiplications and $6(6–1) = 30$ additions. The crafty person, however, might notice that this matrix may be written as the product of two matrices:

$$\begin{bmatrix} 1 & 2 & 4 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & -1 & -2 & -4 \\ 2 & 4 & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & -4 & -1 & -2 \\ 4 & 1 & 2 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & -2 & -4 & -1 \end{bmatrix} \begin{bmatrix} 1 & \cdot & \cdot & 3 & \cdot & \cdot \\ \cdot & -5 & \cdot & \cdot & -7 & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & 3 \\ 5 & \cdot & \cdot & 7 & \cdot & \cdot \\ \cdot & -1 & \cdot & \cdot & -3 & \cdot \\ \cdot & \cdot & 5 & \cdot & \cdot & 7 \end{bmatrix}$$

The zero elements in the decomposed form have been written as periods to emphasize that these elements need not really enter into the computation when either of these matrices multiply a vector. In view of this, multiplying sequentially by the two factors would require only $6(2) + 6(3) = 30$ multiplications and $6(1) + 6(2) = 18$ additions.

If we are really concerned about efficiency, more can be done by taking into account other special elements. For example, observing that 1 or -1 require only an addition or subtraction would save 3 multiplications in the original form, and 9 multiplications in the decomposed form. Other savings could be made if some of the elements of a column were negatives of other elements in the same column.

In the three years since the fast Fourier transform was first published,[1] there have been numerous accounts of what it is and why it works. The more mathematical of these tend to explain it in terms of the fact that the quotient group of a cyclic subgroup of order MN relative to its cyclic subgroup of order M is itself a cyclic group of order N. Those accounts written by computer people usually consider the binary representation of the time and frequency indices, and observe how each bit enters into the summed products. And accounts written by engineers invariably explain the algorithm in terms of merging the spectra of suitable decimations of the original series to form the spectrum of the original series itself.

These approaches are, of course, all quite valid, but they miss the essence of the fast Fourier transform which is, in fact, contained in the example above. If we wish to multiply a matrix M by a column vector x, it may be possible to find a factorization $M = AB$ such that forming first $y = Bx$ then $z = Ay$ requires less multiplications and additions than would forming $z = Mx$ directly. The factors A and B might themselves be able to be factored further profitably.

The fast Fourier transform is a special case of this, where the matrix of interest is the finite discrete Fourier transform matrix whose elements are $\exp 2\pi i(t\hat{t}/N)$ for $\hat{t}$ and t from 0 to $N - 1$. It is really quite irrelevant that the factors turn out to be (except for a permutation and phase shifts) block diagonal matrices where each block is of the same form as the original matrix—this fact is only used in showing that the factoring can be continued.*

Indeed, the example above has exactly the same structure as a

* In fact, for the fastest programs it is not even quite true. See Bergland.[2] The factors there are not equivalent to each other as the "twiddle factors" have been redistributed to increase the number of coefficients having simple forms.

$6 = 3 \times 2$ point fast Fourier transform, except that the nonzero elements in the factors are different. And it achieves exactly the same savings that the fast Fourier transform does in this case. Even the comments about taking advantage of explicit plus or minus ones or negatives of other elements in the column reflect features currently in the better fast Fourier transform programs.

Having seen that the possibility that matrix factoring will speed things up is not unique to the finite Fourier transform, we might ask when we can expect to take advantage of it. It is immediately evident that it does not improve things all the time. We cannot, for example, reduce the number of operations required to multiply by a diagonal matrix. Can we then identify those matrices for which it is useful? Unfortunately not, except by exhibiting a factorization with the required property.

At this point it is useful to observe that, taking advantage only of zeros and ones, there always exist factorizations which do at least as well as the original matrix. This is trivially true if one of the factors is some permutation matrix, but more interestingly so if we consider factors generated by row (or column) elimination as used in the Gaussian elimination method of solving simultaneous linear equations. In matrix terms this process is based on the observation that

$$\begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \end{bmatrix} = \begin{bmatrix} 1 & \cdot \\ r & 1 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} - rm_{11} & m_{22} - rm_{12} & \cdots & m_{2n} - rm_{1n} \end{bmatrix}$$

The parameter r is then chosen to make one of the elements in the second row vanish. Since this means that the right factor takes one less multiplication and one less addition than the original matrix did, and since the left factor clearly only requires one multiplication and one addition, the total number of operations for the two factors is exactly the same as for the original matrix.

In other words, row (or column) elimination preserves the number of operations required to form the product of the matrix with an arbitrary vector. This assertion assumes, of course, that in the elimination we do not destroy more special elements (such as zeros or ones) than we create. In fact, if we can create more of these special elements than we had before, we have won: we have achieved a factorization requiring less operations than did the original matrix.

Notice that in the above example we used a nonsquare matrix. In fact, nothing in the whole discussion suggested M be square, and considering nonsquare matrices is no more difficult than considering square ones. An immediate application of this is to the case where a set of only a few Fourier coefficients are required from a large number of very long sequences. Up until now, usually the best that could be done was to compute the complete fast Fourier transforms and discard the unneeded coefficients.

But it is apparent that by carefully factoring the matrix consisting of those rows of the finite Fourier transform matrix which are of interest, a more efficient algorithm can be produced, tailored to the problem. A reasonable factorization to start from might be fast Fourier factorization of the complete matrix. This is illustrated below for the case where three coefficients are wanted from an eight point transform. The four factor matrices are the reordering and the three passes of the Cooley factorization. Only those rows of each matrix which are marked by arrows need actually be computed. ($W = \exp[(2\pi i)/8]$, explicit negatives and ones are represented as such).

$$
\begin{array}{c}
\rightarrow \\ \rightarrow \\ \rightarrow \\ \\ \\ \\ \\ \\
\end{array}
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & W & W^2 & W^3 & -1 & -W & -W^2 & -W^3 \\
1 & W^2 & -1 & -W^2 & 1 & W^2 & -1 & -W^2 \\
1 & W^3 & -W^2 & W & -1 & -W^3 & W^2 & -W \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & -W & W^2 & -W^3 & -1 & W & -W^2 & W^3 \\
1 & -W^2 & -1 & W^2 & 1 & -W^2 & -1 & W^2 \\
1 & -W^3 & -W^2 & -W & -1 & W^3 & W^2 & W
\end{bmatrix}
$$

$$
=
\begin{array}{c}
\rightarrow \\ \rightarrow \\ \rightarrow \\ \\ \\ \\ \\ \\
\end{array}
\begin{bmatrix}
1 & & & & 1 & & & \\
& 1 & & & & W & & \\
& & 1 & & & & W^2 & \\
& & & 1 & & & & W^3 \\
1 & & & & -1 & & & \\
& 1 & & & & -W & & \\
& & 1 & & & & -W^2 & \\
& & & 1 & & & & -W^3
\end{bmatrix}
\begin{array}{c}
\rightarrow \\ \rightarrow \\ \rightarrow \\ \\ \rightarrow \\ \rightarrow \\ \rightarrow \\ \\
\end{array}
\begin{bmatrix}
1 & & 1 & & & & & \\
& 1 & & W^2 & & & & \\
1 & & -1 & & & & & \\
& 1 & & -W^2 & & & & \\
& & & & 1 & & 1 & \\
& & & & & 1 & & W^2 \\
& & & & 1 & & -1 & \\
& & & & & 1 & & -W^2
\end{bmatrix}
$$

$$\begin{array}{c}\to\\\to\\\to\\\to\\\to\\\to\\\to\\\to\end{array}\begin{bmatrix}1 & 1 & & & & & & \\ 1 & -1 & & & & & & \\ & & 1 & 1 & & & & \\ & & 1 & -1 & & & & \\ & & & & 1 & 1 & & \\ & & & & 1 & -1 & & \\ & & & & & & 1 & 1 \\ & & & & & & 1 & -1\end{bmatrix}\begin{array}{c}\to\\\to\\\to\\\to\\\to\\\to\\\to\\\to\end{array}\begin{bmatrix}1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1\end{bmatrix}$$

We could also have regarded $\pm i$ as special elements.

Our suggestion then is that if one has a matrix which he wants to multiply efficiently into a great number of arbitrary vectors, it might be worthwhile to try to find a factorization of the matrix such that multiplying sequentially by the factors is cheaper than multiplying by the original matrix. Indeed, it is worthwhile to try to find an extremely good, perhaps even the best, such factorization.

Since we cannot identify *a priori* matrices for which this can be done, let alone give an algorithm for finding the best or even just a good factorization, the best we can recommend is to generate trial factorizations and compare them. A useful tool for this is row (or column) elimination: because of the invariance property mentioned earlier, such a factorization cannot lose much, and might gain. As an exercise to the reader, we suggest deriving the factorization of the matrix given at the beginning of this paper, or the eight point fast Fourier transform above. Notice that in the case of the fast Fourier transform it is useful to express the matrix in real arithmetic before reducing it, because then it is more obvious how to go further in the reduction, since in the computer it is usually the number of real operations that counts.

REFERENCES

1. Cooley, J. W., and Tukey, J. W., "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, *19,* No. 90 (April 1965), pp. 297–301.
2. Bergland, G. D., "A Fast Fourier Transform Algorithm Using Base Eight Iterations," Math. of Computation, *22,* No. 102 (April 1968), pp. 275–279.