A simple derivation of Glassman's general $N$ fast Fourier transform, and a corresponding FORTRAN program, is presented. This fast Fourier transform is based upon a representation of the discrete Fourier transform matrix as a product of sparse matrices.

AMS (MOS) Subject Classification: 65T05

Key Words: FFT, Fast Fourier transform factorization, Discrete Fourier transform

Work Unit Number 8 (Computer Science)

Sponsored by the United States Army under Contract No. DAAG29-75-C-0024. This material is based upon work supported by the National Science Foundation under Grant No. MCS78-09525.
SIGNIFICANCE AND EXPLANATION

The discrete Fourier transform is the basis for several accurate techniques for the numerical solution of partial differential equations. The fast Fourier transform, an algorithm which allows one to compute rapidly the discrete Fourier transform, makes these techniques computationally efficient. This paper attempts to present a lucid description of one fast Fourier transform, the fast Fourier transform presented by Glassman.

In the past people have frequently been content to compute rapidly the discrete Fourier transform of vectors whose length is a power of two. Glassman's fast Fourier transform allows rapid computation of the discrete Fourier transform of vectors of arbitrary length.

The responsibility for the wording and views expressed in this descriptive summary lies with MRC, and not with the author of this report.
1. Introduction

Let the \( N \)-vector \( v \) be the discrete Fourier transform (DFT) of the \( N \)-vector \( u \), i.e., the components \( v_k \) of \( v \) are computed from the components \( u_k \) of \( u \) by the rule

\[
v_k = \frac{1}{N} \sum_{j=1}^{N} w_n(k-1)(j-1) \text{ for } k = 1, 2, \ldots, N
\]

where

\[
w_n = \exp\left(-2\pi j / N\right)
\]

is a principle \( N \)-th root of unity. It is easily demonstrated that the components of \( u \) can be recovered from the components of \( v \) by the rule

\[
u_k = \frac{1}{N} \sum_{j=1}^{N} w_n^{-1}(k-1)(j-1) \text{ for } k = 1, 2, \ldots, N
\]

The \( N \)-point DFT matrix \( W_N \) is defined to be the matrix of order \( N \) whose entry in row \( i \), column \( j \) is

\[
W_{ij} = w_n(i-1)(j-1).
\]

Therefore the relations between \( u \) and \( v \) presented above can be written as

\[
v = \frac{1}{N} W_N u \quad \text{and} \quad u = \frac{1}{N} W_N^{-1} v
\]

where \( W_N^{-1} \) denotes the matrix obtained by replacing each entry of \( W_N \) by its complex conjugate.

A fast Fourier transform (FFT) is generally considered to be any algorithm which rapidly computes the DFT of a given vector. One of the most popular FFTs was presented by...

Sponsored by the United States Army under Contract No. DAAG29-75-C-0024. This material is based upon work supported by the National Science Foundation under Grant No. MCS78-09525.

$$N \cdot (R_1 + R_2 + \ldots + R_K)$$

complex operations, where one operation denotes one multiplication followed by one
addition, whenever N admits the representation

$$N = R_1 R_2 \ldots R_K$$
as a product of K positive integers $R_1, R_2, \ldots, R_K$. Since the publication of their
article numerous authors have presented other FFTs, each requiring approximately the same
number of complex operations. One notable exception is the FFT of Winograd [7].

In this paper I will present a description of Glassman's [5] FFT. This description of
Glassman's FFT differs from one presented by Drubin [4] only in the definition of the
tensor product. (However, neither Glassman nor Drubin presented a FORTRAN program which
computes the DFT of a given N-vector.) I define the tensor product $A \otimes B$ of two
matrices $A, B$ to be the matrix which, when partitioned into blocks the size of $A$, has
$A_{i,j}$ as the entry in block row $i$ and block column $j$. In the appendix of this paper I
have presented proofs of three well known properties possessed by this tensor product.

Glassman's FFT computes the DFT of an N-vector using the same number of complex
operations as the Cooley-Tukey FFT. The main advantage of Glassman's FFT is that it is
easily coded, a fact which should be compared with Singleton's [6] FFT. The main
disadvantage of Glassman's FFT is that it requires an N-vector of working storage to
compute the DFT of an N-vector. I will show how one can, to some extent, eliminate this
disadvantage.

I would also like to mention that de Boor [1] has recently presented an FFT that is
also easily described and coded.
2. Factorization of the Discrete Fourier Transform Matrix

Consider the DFT matrix $W_{PQ}$ where $P, Q$ are two positive integers.

Partition the $i$-th row of $W_{PQ}$ into $Q$ groups of $P$ successive entries. The entries in the $q$-th group are

$$[w_{(i-1)(0+(q-1)P)}; w_{(i-1)(1+(q-1)P)}; \ldots; w_{(i-1)(P-1+(q-1)P)}].$$

Each member of this group contains the common factor

$$w_{PQ}^{(i-1)(q-1)P} = w_{PQ}^{(i-1)(q-1)},$$

therefore the $q$-th group admits the representation

$$w_{PQ}^{(i-1)(q-1)} Y_i^{(P,Q)}$$

where

$$Y_i^{(P,Q)} = [w_{PQ}^{(i-1)(0)}; w_{PQ}^{(i-1)(1)}; \ldots; w_{PQ}^{(i-1)(P-1)}]$$

denotes the first $P$ entries in the $i$-th row of $W_{PQ}$.

Next, partition the rows of $W_{PQ}$ into $P$ groups of $Q$ successive rows. The rows in the $p$-th group are

$$[w_{Q}^{((0+p-1)Q)(0)} Y_i^{(P,Q)}; w_{Q}^{((0+p-1)Q)(1)} Y_i^{(P,Q)}; \ldots; w_{Q}^{((0+p-1)Q)(Q-1)} Y_i^{(P,Q)}].$$

Observe that each member of this group contains the term

$$w_{Q}^{((p-1)Q)} = 1,$$

therefore the $p$-th group admits the representation

$$[Y_i^{(P,Q)}; 0; Y_i^{(P,Q)}; \ldots; Y_i^{(P,Q)}].$$

Here the matrix in square brackets is a block diagonal matrix, each block a $1 \times P$ matrix,

where the $i$-th diagonal block is
and \( I_p \) is the identity matrix of order \( P \).

These results allow us to prove the following lemma:

**Lemma:** The DFT matrix \( W_{PQ} \) admits the factorization

\[
W_{PQ} = F(P, Q)[I_p \oplus W_0]
\]

where

\[
Y(P, Q) = \begin{bmatrix}
\omega^{(i-1)(0)}_P & \omega^{(i-1)(1)}_P & \cdots & \omega^{(i-1)(P-1)}_P
\end{bmatrix}
\]

denotes the first \( P \) entries in the \( i \)-th row of \( W_{PQ} \), and

\[
F(P, Q) = \begin{bmatrix}
Y(P, Q) & Y(P, Q) & \cdots & Y(P, Q) \\
\omega^{(i-1)(0)}_P & \omega^{(i-1)(1)}_P & \cdots & \omega^{(i-1)(P-1)}_P \\
\omega^{(i+1)(0)}_P & \omega^{(i+1)(1)}_P & \cdots & \omega^{(i+1)(P-1)}_P \\
\vdots & \vdots & \ddots & \vdots \\
\omega^{(p-1)(0)}_P & \omega^{(p-1)(1)}_P & \cdots & \omega^{(p-1)(P-1)}_P
\end{bmatrix}
\]

is a \( PQ \times Q \) block matrix with \( 1 \times P \) blocks.

**Proof:** From the definition of \( Y(P, Q) \) we find that the \( p \)-th group of \( Q \) successive rows of

\[
F(P, Q)[I_p \oplus W_0]
\]

is

\[
\begin{bmatrix}
Y(P, Q) & Y(P, Q) & \cdots & Y(P, Q) \\
\omega^{(i-1)(0)}_P & \omega^{(i-1)(1)}_P & \cdots & \omega^{(i-1)(P-1)}_P \\
\omega^{(i+1)(0)}_P & \omega^{(i+1)(1)}_P & \cdots & \omega^{(i+1)(P-1)}_P \\
\vdots & \vdots & \ddots & \vdots \\
\omega^{(p-1)(0)}_P & \omega^{(p-1)(1)}_P & \cdots & \omega^{(p-1)(P-1)}_P
\end{bmatrix}
\]

which our previous computations have shown to be the \( p \)-th group of \( Q \) successive rows of \( W_{PQ} \). Since \( p \) was arbitrary we therefore conclude that

\[
W_{PQ} = F(P, Q)[I_p \oplus W_0].
\]

The matrix \( F(P, Q) \) defined in the above lemma has several interesting limiting cases, in particular

\[
F(P, 1) = W_P \quad \text{and} \quad F(1, Q) = I_Q.
\]
These observations aid us in the proof of the following theorem.

Let $N$ admit the representation

$$N = R_1 R_2 \ldots R_K$$

as a product of $K$ positive integers $R_1, R_2, \ldots, R_K$. Then $W_N$ admits the representation

$$W_N = F_1 F_2 \ldots F_K$$

as a product of $K$ sparse matrices $F_1, F_2, \ldots, F_K$ where

$$F_L = I_{R_1 \ldots R_{L-1}} \otimes F_{R_L \ldots R_K}$$

(The products $R_1 \ldots R_{L-1}$ for $L = 1$ and $R_{L+1} \ldots R_K$ for $L = K$ are defined to be 1.)

Proof: The previous lemma, with $P = R_L$ and $Q = R_{L+1} \ldots R_K$, states that

$$W_N = F_L[I_R \otimes W_{R_{L+1} \ldots R_K}]$$

Therefore the identity

$$W_N = F_1 \ldots F_{L-1}[I_{R_1 \ldots R_{L-1}} \otimes W_{R_L \ldots R_K}]$$

holds for $L = 2$. Let us suppose the identity holds for some $L < K$. The previous lemma, with $P = R_L$ and $Q = R_{L+1} \ldots R_K$, states that

$$W_{R_L \ldots R_K} = F_L[I_R \otimes W_{R_{L+1} \ldots R_K}]$$

and so

$$I_{R_1 \ldots R_{L-1}} \otimes W_{R_L \ldots R_K} = F_L[I_{R_1 \ldots R_{L-1}} \otimes W_{R_{L+1} \ldots R_K}]$$

Consequently, if the identity holds for some $L < K$ then it holds for $L + 1$ too. Therefore the identity must hold for $L = K$, i.e.

$$W_K = F_1 F_2 \ldots F_K$$

where we have noted that

$$I_{R_1 \ldots R_{K-1}} \otimes W_{R_K} = I_{R_1 \ldots R_{K-1}} \otimes F_K = F_K$$
3. A FORTRAN Implementation of Glassman's Fast Fourier Transform

The previous theorem, due to Glassman, allows us to easily code a FFT. For to compute

\[ W_u \]

with the result stored over \( u \), we only need apply the factors \( F_1, F_2, \ldots, F_K \) of \( W_u \) to \( u \) in the reverse order.

Suppose that we have just applied the factor

\[ F_{L+1} = I_B \otimes F^{(C,A)} \]

to \( u \), where (A = after, B = before, and C = current)

\[ A = R_{L+2} \ldots R_K, \]  
\[ B = R_1 \ldots R_L, \]  
\[ C = R_{L+1}. \]

Then we should next apply the factor

\[ F_L = I_B/R_L \otimes F^{(R_L, AR_{L+1})} \]

to \( u \). This computation can be described as

1. \( A \times A \times C \)
2. Let \( C \) be the divisor \( R_L \) of \( B \)
3. \( R = R/C \)
4. \( u = I_B \otimes F^{(C,A)} u \).

Since the order of the divisors \( R_1, R_2, \ldots, R_K \) of \( N \) is unimportant we find that the entire algorithm may be described as

1. \( A + 1 \)
2. \( B + N \)
3. \( C + 1 \)
4. While \( B > 1 \) do
5. \( A + A \times C \)
6. Let \( C > 1 \) be a divisor of \( B \\
7. \quad B = B/C \\
8. \quad u = \text{I}_R \otimes \mathbf{f}^{(C,A)} u \\
9. \quad \endwhile.

With the exception of steps 6 and 8, each step of this algorithm can be directly implemented in FORTRAN. Observe that step 6 admits the expansion

6.1 \( C = 2 \)  \\
6.2 While \( B \mod C \neq 0 \) do  \\
6.3 \( C = C + 1 \)  \\
6.4 \quad \endwhile

into steps that can be directly implemented in FORTRAN. We next consider the expansion of step 8.

Let the product \( RS \) of the integers \( R, S \) be a divisor of \( N \). For any \( N \)-vector \( w \) we define \( w^{(R)} \) to be the FORTRAN array of dimension \( (R,N/R) \) which is equivalent to \( w \), and \( w^{(R,S)} \) to be the FORTRAN array of dimension \( (R,S,N/RS) \) which is equivalent to \( w \). This definition merely implies that

\[
\begin{align*}
  w^{(R)}_{i,j} &= w_{i+(j-1)R} \\
  w^{(R,S)}_{i,j,k} &= w_{i+(j-1)R+(k-1)RS}
\end{align*}
\]

Let

\[
  v = \text{I}_R \otimes \mathbf{f}^{(C,A)} u
\]

denote the result of the computation described in step 8. As shown in the appendix we find that

\[
  v^{(B)} = u^{(B)} F^{(C,A)} T,
\]

or equivalently that

\[
  v^{(B)}_{i,j} = \left( \sum_{k=1}^{AC} u^{(B)}_{i,k} f^{(C,A)}_{j,k} \right)
\]

for \( i = 1, 2, \ldots, B \) and \( j = 1, 2, \ldots, AC \). If we express \( j \) in the form

-7-
with $1 \leq j_A < A$ and $1 \leq j_C < C$, then the nonzero entries in the $t-i$ row of $A$
are the numbers
\[ (j_A - 1)(i - 1) \]
\[ u_{AC} \]
in columns $k = i + (j_A - 1)A$ for $k = 1, 2, \ldots, C$. We therefore find that
\[ v_{j_A, j_C} = \frac{C}{A} \sum_{i=1}^{C} u_{i, i, i, j_A, j_C} \]
or equivalently that
\[ v_{j_A, j_C} = \frac{C}{A} \sum_{i=1}^{C} u_{i, i, i, j_A, j_C} \]
for $i = 1, 2, \ldots, B$, $j_A = 1, 2, \ldots, A$ and $j_C = 1, 2, \ldots, C$. Consequently, step 0 admits the expansion

8.1 For $j_C = 1, 2, \ldots, C$
8.2 For $j_A = 1, 2, \ldots, A$
8.3 For $i = 1, 2, \ldots, B$
8.4, $v_{j_A, j_C} = \frac{C}{A} \sum_{i=1}^{C} u_{i, i, i, j_A, j_C}$
8.5 Next $i$
8.6 Next $j_A$
8.7 Next $j_C$

into steps that can be directly implemented in FORTRAN.

Figure 1 presents a FORTRAN version of Glassman's FFT. For comparison we present
de Boor's [1] FFT in Figure 2. I have found that Glassman's FFT runs several percent
faster than de Boor's FFT on the University of Wisconsin's UNIVAC 1100. This increase in speed is probably due to the fact that the loop structure used in Glassman's FFT can more efficiently be implemented in FORTRAN than the loop structure used in de Boor's FFT. The increase in speed would therefore vanish if one were to hand code both FFT's using such a language.

-8-
FIGURE 1
SUBROUTINE CL3SC(A, B, C, HIM, SUM, TAU, R, C)
INTEGER A, B, C
COMPLEX HIM, SUM, TAU(A, A, C)
LOGICAL R

THIS SUBROUTINE IS CALLED FROM SUBROUTINE EFTE.

COMPLEX DELTA, OMEGA, SUM
DATA HIMPT/6.331 63307 17956/
C
ANGLE = HIMPT/360.0
DELTA = COMPLEX(COS(ANGLE), -SIN(ANGLE))
IF (INVRS) DELTA = CONJG(DELTA)
C
OMEGA = COMPLEX(1.0, 0.0)
DO 30 I = 1, C
  IF (I MOD 30) THEN
    DO 20 J = 1, A
      SUM = HIM(J, C, IA)
  20 CONTINUE
  JC = C + J - 1
  SUM = HIM(JC, JC, IA) + OMEGA*SUM
  CONTINUE
  OMEGA = DELTA*OMEGA
  CONTINUE
30 CONTINUE
C
RETURN
C
END

Figure 1 - Cont'd.
```fortran
*** FFT ***

10. INTEGER N, M, K
20. COMPLEX (A,B), WORK(N)
30. LOGICAL INVR

*** INPUT ***

40. INTEGER N
50. ** A COMPLEX N VECTOR TO BE TRANSFORMED 
60. NVR  ** A LOGICAL VARIABLE

*** OUTPUT ***

70. U  ** THE NTH OF U IF INVVR IS 'FALSE', OR
80. N TIMES THE INVERSE NTH OF U IF INVVR IS 'TRUE'.

*** WORKING STORAGE ***

90. WORK  ** A COMPLEX N VECTOR

100. INTEGER A, B, C
110. LOGICAL INV

120. A = 1
130. A = N
140. C = 1
150. INV = 'TRUE'

160. IF (A GT 1) GO TO 20
170. IF (INV) RETURN
180. DO 20 TN = 1, N
190. T(N) = WORK(I)
200. CONTINUE
210. RETURN

220. C = 30 A = A + 1
230. DO 40 TN = 1, N
240. IF (MOD(TN, 2) EQ 0) GO TO 50
250. CONTINUE
260. C = 40 A = A + 1
270. DO 60 TN = 1, N
280. IF (T(N) EQ 0) CALL PERNOR(A, B, C, N, WORK, INV VR)
290. IF (NOT TN) CALL PERNOR(A, B, C, N, WORK, INV VR)
300. TN = 'NOT TN'
310. CONTINUE
320. CONTINUE
330. CONTINUE
340. CONTINUE
350. CONTINUE
360. CONTINUE
370. CONTINUE
380. CONTINUE
390. CONTINUE
400. CONTINUE
410. CONTINUE
420. CONTINUE
430. CONTINUE
440. CONTINUE
450. CONTINUE
460. CONTINUE
470. CONTINUE
480. CONTINUE
490. CONTINUE
500. CONTINUE
510. CONTINUE
520. CONTINUE
530. CONTINUE
540. CONTINUE
550. CONTINUE
560. CONTINUE
570. CONTINUE
580. CONTINUE
590. CONTINUE
600. CONTINUE
610. CONTINUE
620. CONTINUE
630. CONTINUE
640. CONTINUE
650. CONTINUE
660. CONTINUE
670. CONTINUE
680. CONTINUE
690. CONTINUE
700. CONTINUE
710. CONTINUE
720. CONTINUE
730. CONTINUE
740. CONTINUE
750. CONTINUE
760. CONTINUE
770. CONTINUE
780. CONTINUE
790. CONTINUE
800. CONTINUE
810. CONTINUE
820. CONTINUE
830. CONTINUE
840. CONTINUE
850. CONTINUE
860. CONTINUE
870. CONTINUE
880. CONTINUE
890. CONTINUE
900. CONTINUE
910. CONTINUE
920. CONTINUE
930. CONTINUE
940. CONTINUE
950. CONTINUE
960. CONTINUE
970. CONTINUE
980. CONTINUE
990. CONTINUE
1000. CONTINUE

Figure 2

-11-
```
SUBROUTINE KENDR( A,R,EP,UNIT,TH120)
  INTEGER A,R,EP
  COMPLEX UT(2,2,UNIT),A,R
  LOGICAL TMAP

  THIS SUBROUTINE IS CALLED FROM SUBROUTINE TPEET.

  C
  C
  C
  COMPLEX OMEGA,DELTA,SUM
  DATA TH0PI/1.2831 05367 17957/
  C
  C
  ANGLE = TH0PI/FLOAT(A+1)
  DELTA = CMPLX(COS(ANGLE),=SIN(ANGLE))
  IF (INVRS) DELTA = CONJG(DELTA)
  C
  OMEGA = CMPLX(1.,0.)
  DO 40 ICR1,C
       DO 30 ICR1,A
       DO 20 ICR1,R
           SUM = UIN(IA,IB,C)
       DO 10 JCRP2,C
           JC = C+1=JCR
           SUM = UIN(IA,IB,JC) + OMEGA*SUM
       10 CONTINUE
       OMEGA = DELTA*OMEGA
  20 CONTINUE
  30 CONTINUE
  40 CONTINUE
  C
  RETURN
  C
  END

Figure 2 - Cont'd.
4. Conclusion

Observe that Glassman's FFT requires an N-vector of working storage to compute the FFT of an N-vector, for during the computation

\[ u \ast I_B \ast F^{(CA)}u \]

we need an N-vector to store the result

\[ v = I_B \ast F^{(CA)}u . \]

As explained in the following paragraph, this N-vector of working storage can be replaced by a C-vector of working storage at the expense of additional computational effort.

Let \( p^{(CA)} \) denote the permutation matrix of order \( AC \) which sends row \( \frac{1}{A} + (\frac{j}{A} - 1)C \) of the vector \( w \) into row \( \frac{1}{A} + (\frac{j}{C} - 1)A \) of the vector \( p^{(CA)}w \).

Consequently

\[ I_B \ast p^{(CA)}u = I_B \ast p^{(CA)}F^{(CA)}u \]

where \( p^{(CA)}F^{(CA)} \) is a block diagonal matrix with \( C \times C \) blocks. Therefore the computation

\[ u \ast I_B \ast F^{(CA)}u \]

can be replaced by the equivalent computation

\[ u \ast I_B \ast F^{(CA)}F^{(CA)}u , \]

\[ u \ast I_B \ast p^{(CA)}T u . \]

Careful consideration reveals that this latter sequence of calculations requires only a C-vector of working storage.

It is also possible to incorporate any FFT which computes the DFT of an N-vector for special values of N into Glassman's FFT. Recall that

\[ W_N = F_1F_2\ldots F_k \]

where

\[ F_k = I_{R_1R_2\ldots R_{k-1}} \ast U_{R_k} . \]

Therefore any FFT which computes the DFT of an \( R_k \)-vector can be used when the factor \( U_{R_k} \) is to be applied to the vector being transformed.
Appendix: Some Properties of the Tensor Product

We have defined the tensor product \( A \otimes B \) of two matrices \( A, B \) as the matrix which, when partitioned into blocks the size of \( A \), has \( A_{i,j}B \) as the entry in block row \( i \) and block column \( j \).

Consider now any \( N \)-vector \( w \). If \( R \) is a divisor of \( N \) we define \( w^{(P)} \) to be the FORTRAN array of dimension \((R,N/P)\) equivalent to \( w \), i.e.

\[
 w_{i,j}^{(P)} = w_{i+(j-1)R}^{(P)}
\]

With these definitions in mind let us now prove the following

**Property 1:** Let \( A, B \) be rectangular matrices where \( A \) is a \( R \times C \) matrix. Then

\[
 v = (A \otimes B)u
\]

if and only if

\[
 v^{(R)} = Au^{(C)}B^T.
\]

**Proof:** Let

\[
 v = (A \otimes B)u.
\]

From the definition of the tensor product \( A \otimes B \) we observe, for each \( i \), that

\[
 v_{i,j}^{(R)} = \sum_{j} A_{i,j}^{(C)}u_{i,j}^{(C)} = A^{(C)}Bv_{i,j}^{(R)}.
\]

The sum within the curly brackets is easily identified as the \( i \)-th column of \( u^{(C)}B^T \), consequently we infer that

\[
 v^{(R)} = Au^{(C)}B^T.
\]

The proof of the converse is obtained by reversing the argument presented above.

Carl de Boor [21] has noted that this property allows one to easily compute

\[
 v = (A \otimes B)u
\]

given \( u \). For if \( A \) is an \( R \times C \) matrix then

\[
 v = Au^{(C)}B^T = \{B(Au^{(C)})\}^T,
\]

consequently programs which apply \( A \) and \( B \) to vectors can easily be used to apply \( A \otimes B \) to vectors. This property also allows us to easily prove the following
Property 2: Let the products $A_1A_2$ and $B_1B_2$ be defined. Then

$$(A_1A_2) \otimes (B_1B_2) = (A_1 \otimes B_1)(A_2 \otimes B_2).$$

Proof: Let $A_k$ be an $R_k \times C_k$ matrix for $k = 1, 2$. Observe that $C_1 = R_2$ because the product $A_1A_2$ is defined. Let $u$ be an arbitrary vector and define

$$w = [(A_1A_2) \otimes (B_1B_2)]u.$$ 

Property 1 implies that

$$w^T = (A_1A_2)u^T (B_1B_2)^T = A_1(A_2u^T B_2^T)B_1^T.$$ 

If we define

$$v = (A_2 \otimes B_2)u$$ 

then property 1 implies that

$$v^T = A_2u^T B_2^T$$ 

and

$$w^T = A_1v^T B_1^T$$ 

since $C_1 = R_2$. Using property 1 once more we find that

$$w = (A_1 \otimes B_1)v^T$$ 

and so

$$w = (A_1 \otimes B_1)(A_2 \otimes B_2)u.$$ 

Consequently, for an arbitrary vector $u$ we have

$$[(A_1A_2) \otimes (B_1B_2)]u = (A_1 \otimes B_1)(A_2 \otimes B_2)u,$$

therefore

$$(A_1A_2) \otimes (B_1B_2) = (A_1 \otimes B_1)(A_2 \otimes B_2).$$

The last tensor product property that we will need is described as follows.

Property 3: For arbitrary matrices $A_1$, $A_2$, and $A_3$

$$A_1 \otimes (A_2 \otimes A_3) = (A_1 \otimes A_2) \otimes A_3.$$ 

Proof: Let $A_k$ be an $R_k \times C_k$ matrix for $k = 1, 2, 3$. Let $e_i^B$ be the $B$-vector obtained by replacing the $i$-th component of the zero $B$-vector by 1. Let $a_{i,j}^{(k)}$ denote the entry of $A_k$ in row $i$ and column $j$. Observe that
\[
A_k = \left( \begin{array}{ccc}
1, & \ldots, & 1
\end{array} \right)_{i_1, \ldots, i_k}^{e_1, \ldots, e_3} \quad \text{for} \quad k = 1, 2, 3.
\]

Consequently

\[
A_1 \otimes (A_2 \otimes A_3) = \sum a_{i_1, j_1}^{(1)} a_{k, i_2}^{(2)} a_{l, i_3}^{(3)} [e_{i_1} e_{i_2} e_{i_3}] \otimes [e_{k} e_{l} e_{m}] \otimes [e_{n} e_{n}]
\]

and

\[
(A_1 \otimes A_2) \otimes A_3 = \sum a_{i_1, j_1}^{(1)} a_{k, i_2}^{(2)} a_{l, i_3}^{(3)} [e_{i_1} e_{i_2} e_{i_3}] \otimes [e_{k} e_{l} e_{m}] \otimes [e_{n} e_{n}].
\]

From the easily verified identity

\[
\begin{align*}
(R_1) (C_1) & \quad (R_2) (C_2) \quad (R_3) (C_3) \\
[e_{i_1} e_{i_2} e_{i_3}] & \otimes [e_{k} e_{l} e_{m}] \otimes [e_{n} e_{n}]
\end{align*}
\]

we deduce that

\[
A_1 \otimes (A_2 \otimes A_3) = (A_1 \otimes A_2) \otimes A_3.
\]
REFERENCES


# A Simple Derivation of Glassman's General N Fast Fourier Transform

## Authors
Warren E. Ferguson, Jr.

## Performing Organization
Mathematics Research Center, University of Wisconsin
610 Walnut Street
Madison, Wisconsin 53706

## Distribution Statement
Approved for public release; distribution unlimited.

## Key Words
- FFT
- Fast Fourier transform factorization
- Discrete Fourier transform

## Abstract
A simple derivation of Glassman's general N fast Fourier transform, and corresponding FORTRAN program, is presented. This fast Fourier transform is based upon a representation of the discrete Fourier transform matrix as a product of sparse matrices.