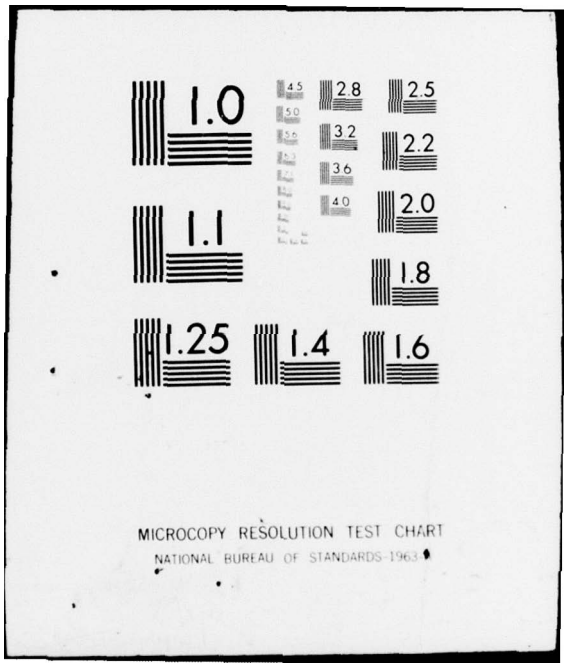
AD-A046 523 MASSACHUSETTS INST OF TECH CAMBRIDGE RESEARCH LAB OF--ETC F/G 12/1
VECTOR RADIX FAST FOURIER TRANSFORM,(U)
1977 D B HARRIS, J H MCCLELLAN, D S CHAN N00014-75-C-0951

UNCLASSIFIED NL

1 OF 1
AD A046523

END
DATE FILMED
12-77
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
VECTOR RADIX FAST FOURIER TRANSFORM

5. TYPE OF REPORT & PERIOD COVERED
Reprint

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
David B. Harris, James H. McClellan, David S. K. Chan, Hans W. Schuessler

8. CONTRACT OR GRANT NUMBER(s)
N00014-75-C-0951
NSF-ENG-71-02319

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
NR049-308

11. CONTROLLING OFFICE NAME AND ADDRESS
Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

12. REPORT DATE
1977

13. NUMBER OF PAGES
6

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Office of Naval Research
Information Systems Program, Code S437
Arlington, Virginia 22217

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES
Proc. 1977 IEEE International Conference on Acoustics, Speech and Signal Processing, May 9-11, 1977, Hartford, Connecticut, pp. 548-551

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Fast Fourier transform
Vector radix FFT
Digital signal processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
A new radix-2 two-dimensional direct FFT developed by Rivard is generalized in this paper to include arbitrary radices and non-square arrays. It is shown that the radix-4 version of this algorithm may require significantly fewer computations than conventional row-column transform methods. Also, the new algorithm eliminates the matrix transpose operation normally required when the array must reside on a bulk storage device. It requires the same number of passes over the array on bulk storage as efficient matrix transpose routines, but produces the transform in bit-reversed order. An additional pass over the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

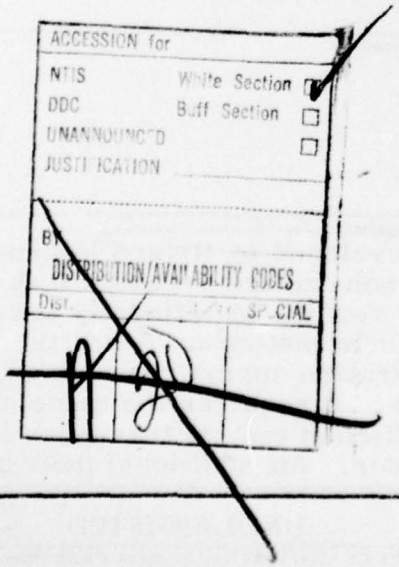
UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

304 050

20. ABSTRACT

data is necessary to sort the array if normal ordering is desired.



1977 IEEE Int. Conf. on Acoustics, Speech and Signal Processing, May 9-11, 1977, Hartford, Conn.

VECTOR RADIX FAST FOURIER TRANSFORM

David B. Harris, James H. McClellan
David S. K. Chan and Hans W. Schuessler

Massachusetts Institute of Technology
Research Laboratory of Electronics
Cambridge, Mass. 02139

ABSTRACT

A new radix-2 two-dimensional direct FFT developed by Rivard is generalized in this paper to include arbitrary radices and non-square arrays. It is shown that the radix-4 version of this algorithm may require significantly fewer computations than conventional row-column transform methods. Also, the new algorithm eliminates the matrix transpose operation normally required when the array must reside on a bulk storage device. It requires the same number of passes over the array on bulk storage as efficient matrix transpose routines, but produces the transform in bit-reversed order. An additional pass over the data is necessary to sort the array if normal ordering is desired.

INTRODUCTION

Since the appearance of the original Cooley-Tukey algorithm in 1965, the standard methods of computing the two-dimensional (2-D) discrete Fourier transform (DFT) of an array have capitalized on the separability of the 2-D DFT [1]. Using a 1-D FFT algorithm, row-wise and columnwise, 1-D DFT's can be computed to yield the 2-D transform. This scheme amounts to decimating and transforming the array first in one index and then in the other.

A new algorithm which performs the decimation in both indices simultaneously has been derived by Rivard using a holor algebra formalism [2]. Rivard demonstrated that his radix-2 direct 2-D FFT eliminates 25% of the multiplies required by the conventional row-column approach.

The purpose of this paper is to present an alternate derivation of the new algorithm and to extend it to rectangular arrays and arbitrary radices. Further, we show that even larger savings in multiplies are obtained when the algorithm is generalized to operate with larger radices and on higher dimensional arrays. We refer to the general algorithm as the vector radix algorithm, since to specify the decimation of the array, multiple radices are required, one for each index of the array.

An additional, machine dependent, implication of the new algorithm is explored here as well. When the DFT computation is implemented on a computer with an insufficient amount of core memory to contain the entire array, a matrix transpose operation is a necessary component of the row-column approach. This fact has occasioned a literature on fast transpose techniques, led by Eklundh [3]. However, the vector radix algorithm requires no transpose. The transpose is, in effect, incorporated into the transform. A vector radix transform requires the same number of passes over the array on secondary storage (e.g. disk) as the Eklundh transpose algorithm, when the resulting DFT can be tolerated in bit reversed order. If the DFT must be in correct order, an additional transfer of the array is required to perform bit reversed sorting. On a machine which is I/O intensive, this extra pass over the array may compromise the computational advantages of the new algorithm.

DERIVATION

As with the one-dimensional FFT algorithm, the new direct two-dimensional FFT is derived by decomposing the DFT into sums of smaller DFT's multiplied by "twiddle" factors. We derive here a single stage of the general vector radix algorithm for the decimation in time case. This is all that is necessary, since the complete algorithm is obtained by recursive application of this basic decomposition.

We suppose the 2-D DFT

$$X(k,\ell) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m,n]\, W_M^{km}\, W_N^{\ell n}$$

$$k=0,\ldots,M-1 \qquad \ell=0,\ldots,N-1 \qquad (1)$$

of the MxN array x[m,n] is desired, where

$$W_M = \exp(-j2\pi/M)$$

This work was supported in part by the Advanced Research Projects Agency monitored by ONR under Contract N00014-75-C-0951-NR 049-308 and in part by the National Science Foundation under Grant ENG71-02319.

Provided radix r_1 divides M and radix r_2 divides N:

$$M/r_1 = P \qquad N/r_2 = Q \quad \text{integers,} \tag{2}$$

the DFT may be computed with an $r_1 x r_2$ stage. Using the change of variables:

$$\begin{aligned} m &= r_1 p+u & n &= r_2 q+v \\ p &= 0,\ldots,P-1 & q &= 0,\ldots,Q-1 \\ u &= 0,\ldots,r_1-1 & v &= 0,\ldots,r_2-1 \end{aligned} \tag{3}$$

the DFT (1) is decomposed into $r_1 r_2$ PxQ DFT's. Performing the substitution of variables of (3) in equation (1), we obtain:

$$X(k,\ell) = \sum_{p=0}^{P-1} \sum_{u=0}^{r_1-1} \sum_{q=0}^{Q-1} \sum_{v=0}^{r_2-1} x[r_1 p+u, r_2 q+v] \cdot W_M^{k(r_1 p+u)} \cdot W_N^{\ell(r_2 q+v)} . \tag{4}$$

From equation (2):

$$W_M^{k(r_1 p+u)} = W_P^{kp} W_M^{ku} ; \quad W_N^{\ell(r_2 q+v)} = W_Q^{\ell q} W_N^{\ell v} . \tag{5}$$

Substituting the relations of (5) into equation(4), we obtain the desired result:

$$X(k,\ell) = \sum_{u=0}^{r_1-1} \sum_{v=0}^{r_2-1} W_M^{ku} W_N^{\ell v} X_{PQ}^{uv}(k,\ell) \tag{6}$$

where

$$X_{PQ}^{uv}(k,\ell) = \sum_{p=0}^{P-1} \sum_{q=0}^{Q-1} x[r_1 p+u, r_2 q+v] W_P^{kp} W_Q^{\ell q} \cdot$$

is the 2-D DFT of a PxQ element array.

We introduce a second change of variables to split the computation of $X(k,\ell)$ into $r_1 x r_2$ element butterfly groupings:

$$\begin{aligned} k &= a+bP & \ell &= c+dQ \\ a &= 0,\ldots,P-1 & c &= 0,\ldots,Q-1 \\ b &= 0,\ldots,r_1-1 & d &= 0,\ldots,r_2-1 \end{aligned} \tag{7}$$

Performing the change of variables (7) on equation (6) yields:

$$X(a+bP, c+dQ) = \sum_{u=0}^{r_1-1} \sum_{v=0}^{r_2-1} W_M^{(a+bP)u} W_N^{(c+dQ)v} X_{PQ}^{uv}(a+bP, c+dQ) . \tag{8}$$

Again:

$$W_M^{(a+bP)u} = W_M^{au} W_{r_1}^{bu} ; \quad W_N^{(c+dQ)v} = W_N^{cv} W_{r_2}^{dv} \tag{9}$$

The PxQ point DFT's are periodic:

$$X_{PQ}^{uv}(a+bP, c+dQ) = X_{PQ}^{uv}(a,c) . \tag{10}$$

Using substitutions (9) and 10) in equation (8) produces:

$$X(a+bP, c+dQ) = \sum_{u=0}^{r_1-1} \sum_{v=0}^{r_2-1} W_{r_1}^{bu} W_{r_2}^{dv} [(W_M^{au} W_N^{cv}) X_{PQ}^{uv}(a,c)]. \tag{11}$$

Equation (11) defines the basic structure of the $r_1 x r_2$ butterfly. A few observations will clarify this. First, there is one butterfly for each combination of values of a and c, giving a total of PQ butterflies. Thus, a and c are the butterfly indices. Each butterfly computes $r_1 r_2$ values of $X(\cdot,\cdot)$ (indexed by b and d) from $r_1 r_2$ values of the smaller PxQ element DFT's (indexed by u and v). As input to the butterfly, one element is taken from each PxQ point transform. These elements are not used by any other butterflies in the stage, so that the $r_1 r_2$ output values of the butterfly may be stored in the memory locations occupied by the input elements. Therefore, as with the 1-D FFT algorithm, the vector radix transform may be computed in place.

The inputs to the butterfly are premultiplied by the twiddle factors (in parentheses), then an $r_1 x r_2$ 2-D DFT of the resulting array (in square brackets) is computed. The fact that the butterfly can be interpreted as a 2-D DFT calculation suggests several structures for performing the computation. The conventional row-column FFT structure can be employed or the vector radix approach can be used. The latter is generally more efficient.

One very important observation is that the product of the twiddle factors W_M^{au} and W_N^{cv} may be precomputed and stored in a table. The computational advantage enjoyed by the vector radix algorithm arises from the fact that such products are not computed, whereas they are computed implicitly in the row-column approach.

As an example we consider the radix 2x2 case, where $M=N=2^{\nu}$, ν an integer, and $r_1=r_2=2$. Equation (11) reduces to:

$$X(a+b\tfrac{N}{2}, c+d\tfrac{N}{2}) = \sum_{u=0}^{1} \sum_{v=0}^{1} (-1)^{bu+dv} [(W_N^{au+cv}) X_{\frac{N}{2}\frac{N}{2}}^{u\,v}(a,c)] \tag{12}$$

$$a,c=0,\ldots,N/2-1 \qquad b,d=0,1$$

One possible butterfly structure for the computation of equation (12) is shown in Figure 1. This structure has a minimum number of adds and multiplies, and is the same as an efficient radix

four one-dimensional butterfly structure with the internal multiplies by j removed.

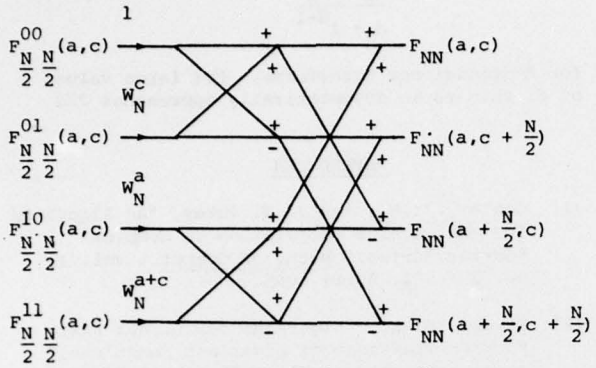


Figure 1: Radix 2x2 Butterfly

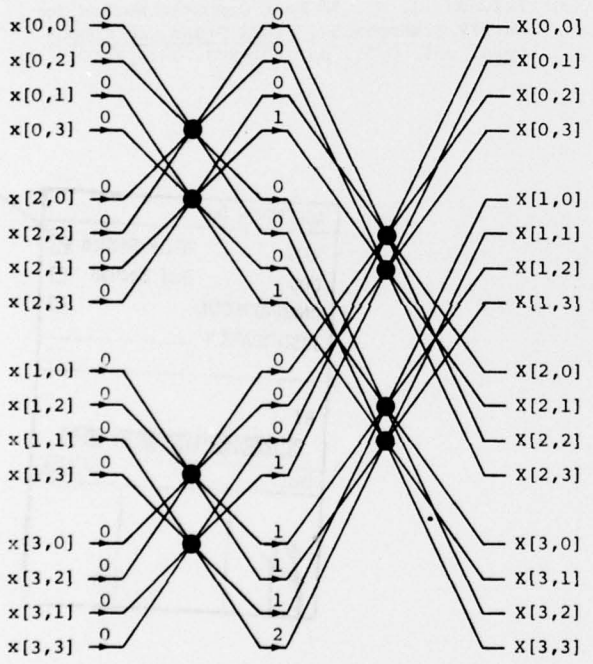


Figure 2: 4x4 DFT with radix 2x2 algorithm. Solid circles indicate 2x2 butterflies and twiddle factor multiplies by W_4^k are indicated by the exponents k.

In Figure 2 we display the radix 2x2 structure for computation of a 4x4 (ν=2) element DFT. Rows of the bit-reversed array are concatenated and arranged vertically on the left hand side of the figure. As is generally true for the 2x2 algorithm, only 2 rows of the array are accessed simultaneously, so that 2 rows worth of core storage are sufficient to implement the algorithm.

COMPARISON AND DISCUSSION

Introduction of the vector radix FFT provides some flexibility in trading CPU speed against I/O transfer rate. In an attempt to quantify this tradeoff, statistics for the two algorithms are presented in a table at the end of this section. The total number of complex multiplies and butterflies required for NxN transforms are tabulated for both algorithms and various radices. The number of butterflies is included as a partial measure of the amount of "overhead" that the algorithms might be expected to have.

In counting the number of multiplies we adopt the philosophy that all butterflies are treated identically. This implies that all twiddle factor multiplies are counted for the second and subsequent stages. Twiddle factors are unity in the first stage, so these multiplies are not counted. Multiplies by ± 1 and $\pm j$ internal to the butterflies are not counted.

As a figure of relative computational speed, the ratio of the total number of multiplies required by each algorithm to the total number of multiplies required by the radix 2 row-column algorithm (for N=4096) is also tabulated.

The left-hand side of the table contains information on the number of times that the entire array must be transfered from bulk memory to core memory and back. This information is presented as a function of the number of rows of the array (2, 4,8,16) that can be stored in core memory. It is assumed that the Eklundh transpose algorithm is used in the row-column approach. This requires $\log_2 N/\log_2 R$ transfers, where R is the available number of rows of core storage and we assume that N is a power of R. No extra transfers are necessary to compute the row and column FFT's, since these may be computed during the first and last stages of the Eklundh algorithm. The vector radix algorithm requires $\log_2 N/\log_2 R + 1$ passes over the array, since (in the radix 2x2 case) $\log_2 R$ stages of the FFT may be computed per array transfer. The extra pass over the array in the vector radix algorithm is necessitated by the 2-D bit-reversed sorting of the data.

It is readily apparent that the vector radix rxr algorithm has fewer multiplies and butterflies than the row-column radix r algorithm. In the case where r=2, there is a 25% reduction in number of multiplies and fewer, albeit more complicated, butterflies with the 2x2 algorithm. Experiments with FORTRAN codings of the two algorithms have shown that a 25% reduction in computation time is achieved in practice for N=64.

On the basis of relative number of multiplies and butterflies, we conclude that the radix 4 row-column algorithm has about the same computational complexity as the vector radix 2x2 algorithm. The fact that the radix 4 1-D algorithm requires one less array transfer argues strongly in its favor. Its disadvantages are less flexibility in transform size (N must be a power of 4 as opposed to a power of 2) and the fact that it produces the transpose of the DFT instead of the normally-ordered DFT. This last, rather minor,

problem is characteristic of all radices in the row-column approach.

The larger radix algorithms are even more attractive as far as the relative numbers of multiplies and butterflies are concerned. The radix 4x4 and 8x8 algorithms appear to be computationally advantageous even when compared with the 1-D radix 4, 8 and 16 algorithms. However, the coding complexity will increase dramatically with increases in the radix size for the vector radix approach. The rather marginal decrease in multiplies in the radix 8x8 case as compared to the radix 4x4 case probably does not justify the extra coding effort. With these facts in mind, we speculate that the radix 4x4 algorithm will be the algorithm of choice for most applications involving machines limited by computational speed or machines with sufficient core storage to hold the entire array.

In situations where the DFT is implemented on an I/O intensive machine and must not be in scrambled order, the row-column approach may be preferable. The row-column algorithm of any radix requires one fewer transfer than the vector radix 2x2 algorithm. When the amount of core storage is sufficient for a number of rows which is not a power of 4, the row-column algorithms require substantially fewer transfers than the vector radix 4x4 algorithm. In such cases, the row-column approach makes better use of increased core memory size.

Transforms of non-square arrays can be accomodated in the vector radix approach by means of mixed radix algorithms. For example, Nx2N transforms may be computed with one stage of radix 2x4 butterflies, followed by $\log_2 N-1$ stages of radix 2x2 butterflies.

In conclusion, we note that the vector radix algorithm can be generalized to higher dimensional transforms. In three dimensions, decimation in all three indices of the array simultaneously leads to a radix 2x2x2 algorithm. This algorithm requires 7/12 the number of multiplies that a conventional algorithm using 1-D FFT's requires. This ratio improves with higher dimensionality and is equal to

$$\frac{2^d - 1}{d \cdot 2^{d-1}}$$

for d-dimensional transforms. For large values of d, this ratio asymptotically approaches 2/d.

REFERENCES

[1] Cooley, J. W., and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Math. of Comput., vol. 19, pp. 297-301, April 1965.

[2] Rivard, G. E., "Algorithm for Direct Fast Fourier Transform of Bivariant Functions," paper presented at the 1975 Annual Meeting of the Optical Society of America, Boston, Massachusetts, October 1975.

[3] Eklundh, J. O., "A Fast Computer Method for Matrix Transposing," IEEE Trans. on Computers, vol. C-21, pp. 801-803, July 1972.

TABLE OF OPERATION COUNTS ($N=2^\nu$)

FFT TYPE	Radix	Disk Accesses 2	4	8	16	Multiplies per Butterfly*	Number of Butterflies	Total Number of Multiplies	Ratio
VR	2x2	$\nu+1$	$\nu/2+1$	$\nu/3+1$	$\nu/4+1$	3+0	$1/4\ N^2\nu$	$3/4\ N^2(\nu-1)$	.75
VR	4x4	-	$\nu/2+1$	$\nu/2+1$	$\nu/4+1$	15+0	$1/32\ N^2\nu$	$15/32\ N^2(\nu-2)$	.426
VR	8x8	-	-	$\nu/3+1$	$\nu/3+1$	63+24	$1/192\ N^2\nu$	$29/64\ N^2(\nu-63/29)$	.405
RC	2	ν	$\nu/2$	$\nu/3$	$\nu/4$	1+0	$N^2\nu$	$N^2(\nu-1)$	1.00
RC	4	ν	$\nu/2$	$\nu/3$	$\nu/4$	3+0	$1/4\ N^2\nu$	$3/4\ N^2(\nu-2)$	.682
RC	8	ν	$\nu/2$	$\nu/3$	$\nu/4$	7+2	$1/12\ N^2\nu$	$3/4\ N^2(\nu-7/3)$	.659
RC	16	ν	$\nu/2$	$\nu/3$	$\nu/4$	15+9	$1/32\ N^2\nu$	$3/4\ N^2(\nu-5/2)$	.648

*Twiddle factor multiplies (first) and internal butterfly multiplies (second) are tabulated separately.