Recursive Pruning of the 2-D DFT with 3-D Signal Processing Applications

Knud Steven Knudsen, *Student Member, IEEE*, and Leonard T. Bruton, *Fellow, IEEE*

Abstract—**A recursively pruned radix-(2 × 2) two-dimensional (2-D) fast Fourier transform (FFT) algorithm is proposed which reduces the number of operations involved in the calculation of the 2-D discrete Fourier transform (DFT). It is able to compute input and output data points having multiple and possibly irregularly shaped (that is, nonsquare) regions of support. The computational performance of the recursively pruned radix-(2 × 2) 2-D FFT algorithm is compared with that of pruning algorithms based on the one-dimensional (1-D) FFT. The proposed recursively pruned 2-D FFT algorithm is shown to yield significant computational savings when employed in the recently proposed combined 2-D DFT/1-D linear difference equation filter method to enhance three-dimensional spatially planar image sequences, and when employed in the MixeD moving object detection and trajectory estimation algorithm.**

I. Introduction

THE one-dimensional (1-D) discrete Fourier transform (DFT) is part of many signal processing algorithms. The 1-D fast Fourier transform (FFT) algorithm [1] provides a computationally efficient method of calculating a length-N 1-D DFT given N input data points. However, when a number of the input data or the output data points can be neglected, increased computational efficiency may be achieved by modifying the FFT algorithm. This is commonly referred to as "pruning" (i.e., removing unnecessary computations from) the FFT algorithm [2]–[4] and has found application in data interpolation, least squares signal approximation, and cepstral smoothing [2].

This contribution extends the idea of FFT pruning to two dimensions. The 2-D FFT pruning methods presented are a result of a logical extension of 1-D FFT pruning methods. The first two algorithms presented have been previously investigated [5], although not as thoroughly as in this contribution. They provide the background knowledge and inspiration for the development of the final, novel 2-D FFT pruning algorithm.

First, a pruned 2-D FFT algorithm is developed based on the row–column decomposition [6] of the 2-D DFT. The row and column DFT's are computed using pruned 1-D FFT algorithms [2]–[4]. A second, pruned 2-D FFT algorithm is developed by following the same approach used to prune 1-D FFT algorithms and by (naively) extending it to a radix-(2 × 2) FFT algorithm [6]. For example, branches of the radix-(2 × 2) 2-D FFT butterflies not involved in the computation of the anticipated output points are pruned (removed). At this point, a problem is recognized. Both of the 2-D FFT pruning algorithms above require the anticipated input or output data points to have a single, contiguous, square region of support with sides that are a power of two in length. It is argued that this constraint severely limited the usefulness of these algorithms whenever there are multiple, unconnected regions of support or when a region of support is irregularly shaped.

An algorithm is proposed that overcomes the above limitation. The 2-D FFT signal flow graph is considered in its entirety. The required input and output data points for each butterfly in each pass are determined via a recursive procedure. Flags are stored in lookup tables for each butterfly indicating which input and output data points are required. The regularity of the 2-D FFT signal flow graph allows the tables to be small. The resultant recursively pruned 2-D FFT algorithm is very computationally efficient, especially when it is applied to a series of images; for example, in image sequence processing applications. The recursively pruned 2-D FFT algorithm is able to handle input and output data points on multiple, separate, and irregularly shaped regions of support. Such regions of support occur in applications such as discrete image analysis and interpolation [7], [8], discrete image sequence filtering [9]–[11], 3-D moving object tracking filters [12], moving object detection and trajectory estimation [13], and high definition television (HDTV) signal processing [5]. As an example, consider a radiologist who selects several, irregularly shaped regions of interest on a nuclear magnetic resonance image and wishes to expand (zoom in on) the regions to show the available detail more clearly. Whereas previous interpolation methods would require several invocations of a pruned 2-D FFT algorithm [7], [8], the recursively pruned 2-D FFT algorithm can accomplish the task in just one invocation. Other applications, such as constrained iterative image reconstruction and multidimensional spectral estimation [6], should also benefit from using the recursively pruned 2-D FFT algorithm.

Manuscript received February 9, 1991; revised May 15, 1992. This work was supported by the Natural Sciences and Engineering Research Council of Canada.

K. S. Knudsen was with the Department of Electrical and Computer Engineering, University of Calgary, Calgary, Alberta, Canada T2N1N4. He is now with the Burchill Communications Research Group, Department of Electrical Engineering, Technical University of Nova Scotia, Halifax, NS Canada, B3J 2X4.

L. T. Bruton is with the Department of Electrical and Computer Engineering, University of Calgary, Calgary, Alberta, Canada, T2N 1N4.

IEEE Log Number 9206023.

Two practical applications are presented which demonstrate the computational efficiency of the recursively pruned 2-D FFT algorithm. In the first problem, the 2-D DFT is combined with 1-D linear difference equation (LDE) filters to realize a discrete transform/spatiotemporal mixed domain, hereafter referred to as MixeD, filter [9]–[11]. The filter is designed to enhance 3-D spatially planar pulse signals [14]. In the second problem, the 2-D DFT and a 1-D high resolution spectral estimation method are combined to form a MixeD moving object detection and trajectory estimation algorithm. This algorithm is able to detect the number of 2-D objects moving on linear trajectories in an image sequence and provide very accurate estimates of their trajectories [13]. In both examples, the recursively pruned 2-D FFT algorithm is used to compute 2-D DFT coefficients with irregularly shaped and, in the latter problem, multiple, unconnected regions of support.

The remainder of the contribution is presented as follows. In Section II, the localized region of support is defined and methods for pruning the 2-D DFT are considered on the basis of previously proposed 1-D FFT pruning techniques [2]–[5], [8]. In Section III, the proposed recursive pruning method is presented and, in Section IV, some experimental results are presented. Finally, in Section V, the recursively pruned radix-(2 × 2) 2-D FFT algorithm is compared with an unpruned radix-(2 × 2) 2-D FFT algorithm in two practical signal processing problems.

II. Pruning the 2-D DFT Using a Localized Region of Support

Consider a discrete finite-extent sequence $x(n_1, n_2)$ with a 2-D region of support R^2, where

$$R^2 \equiv \{(n_1, n_2) |\ 0 \le n_1 < N_1 < \infty,\quad 0 \le n_2 < N_2 < \infty,\quad n_1, n_2, N_1, N_2 \in \mathbb{N}\}. \tag{1}$$

where $\mathbb{N}$ is the set of natural numbers. The region of support R^2 is the finite set of 2-tuples (n_1, n_2) over which $x(n_1, n_2)$ is defined. The sequence $x(n_1, n_2)$ is related to the sequence $X(k_1, k_2)$ via the 2-D DFT pair

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2} \quad \text{for } 0 \le k_1 < N_1,\ 0 \le k_2 < N_2 \tag{2}$$

and

$$x(n_1, n_2) = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X(k_1, k_2) W_{N_1}^{-n_1 k_1} W_{N_2}^{-n_2 k_2} \quad \text{for } 0 \le n_1 < N_1,\ 0 \le n_2 < N_2 \tag{3}$$

where $W_N = \exp[-j(2\pi/N)]$. The DFT pair is written more compactly as

$$x(n_1, n_2) \leftrightarrow X(k_1, k_2). \tag{4}$$

In general, $x(n_1, n_2)$ and $X(k_1, k_2) \in \mathbb{C}$, the set of complex numbers. The region of support for $X(k_1, k_2)$ is assumed to be R^2.

We define a particular *region of support* $L^2 \subseteq R^2$ to correspond to the region in R^2 where $|X(k_1, k_2)|$ is not less than some positive real number ϵ. That is,

$$L^2 = \{(k_1, k_2) |\ |X(k_1, k_2)| \ge \epsilon,\ \epsilon \in \mathbb{R},\ \epsilon > 0\} \tag{5}$$

where $\mathbb{R}$ is the set of real numbers, and we define the corresponding subsequence $\{X_L(k_1, k_2)\} \subseteq \{X(k_1, k_2)\}$ as

$$\{X_L(k_1, k_2)\} = \{X(k_1, k_2) |\ |X(k_1, k_2)| \ge \epsilon,\ \epsilon \in \mathbb{R},\ \epsilon > 0\}. \tag{6}$$

For the applications under consideration, ϵ is chosen to be sufficiently large so that L^2 is a set that defines a localized region of support that is much smaller than R^2, as shown in Fig. 1. The proposed DFT pruning techniques are useful if $|X(k_1, k_2)|$ is sufficiently small outside of L^2 (that is, on $\{R^2 \cap \bar{L}^2\}$) that it can be neglected there. We shall refer to the 2-D sequence $x_L(n_1, n_2)$ as the 2-D inverse DFT of $X_L(k_1, k_2)$.

We now consider DFT pruning techniques that take advantage of *a priori* knowledge of the localized region of support to achieve computational savings. Two methods are considered in this section. First, a technique is considered that is based on the well-known row–column decomposition of the 2-D DFT [6] and uses existing 1-D FFT pruning algorithms [2]–[4]. The second method is an extension of the first method to the 2-D radix-(2 × 2) butterfly FFT algorithms [5] and assumes coverage of the localized region of support L^2 by means of rectangular subregions.

A. The Pruned Row–Column 2-D FFT

An approach has been proposed [5], [7], [8] for the pruning of the 2-D FFT algorithm based on the row–column decomposition of the 2-D DFT [6]. This method involves replacing 1-D FFT's by pruned 1-D FFT's. We will discuss the details of this approach, but it is prudent to first present an example illustrating a 1-D FFT pruning algorithm [4] and the two necessary modifications so that it may be used in the pruned row–column 2-D FFT algorithm. Hereafter, we will emphasize output-pruning algorithms (that is, those yielding $X_L(k_1, k_2)$), with the understanding that the development of input-pruning algorithms is analogous.

An example of 1-D FFT output pruning, where $\{X_L(k_1)\} = \{X(3), X(4), X(5)\}$, is shown in Fig. 2. The thick lines show the computations that must be performed to obtain these points. This algorithm, though based on the 1-D radix-2 decimation-in-time (DIT) output-pruned FFT algorithm [4], differs from it in two ways; the number of output points is not an integer power of two and the 1-D localized region of support L^1 does not include the origin of the discrete frequency space (i.e., $X(0) \in \{X_L(k_1)\}$, $0 \notin L^1$). In cases where the number of output points is not an integer power of two, examination of the 1-D DIT FFT signal flow graph shows that full and partial (half) radix-2 butterflies are computed. Additional steps are therefore required to indicate when and how to compute the partial butterflies.

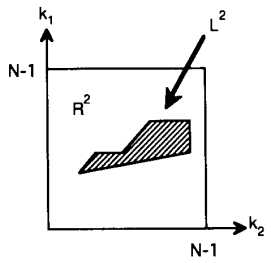
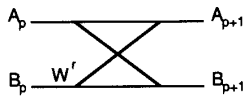
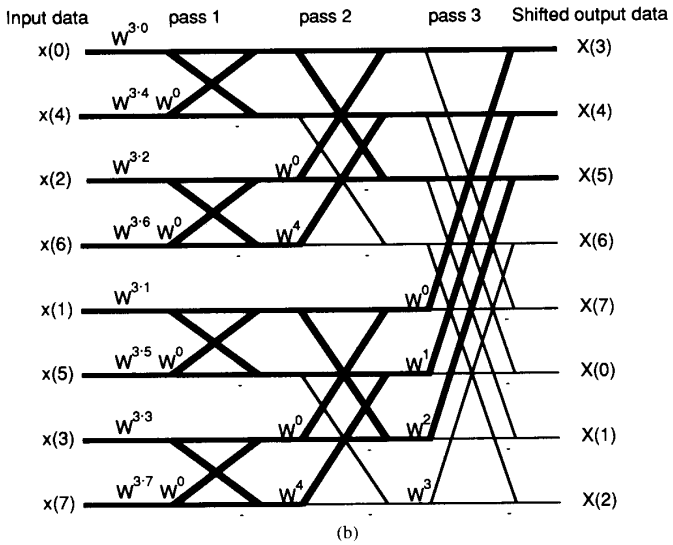


Fig. 1. A localized region of support, L^2.



$$A_{p+1} = A_p + W^r B_p$$
$$B_{p+1} = A_p - W^r B_p$$

(a)



(b)

Fig. 2. (a) A 1-D radix-2 butterfly and the equations it represents. (b) An eight point 1-D FFT with output pruning. The set of output points is $\{X_L(k_1)\} = \{X(3), X(4), X(5)\}$. The input is modulated by $W_N^{in} = W_8^{3n}$ which shifts $\{X_L(k_1)\}$ to the start of the algorithm's output set. These points are later shifted to the correct location in the discrete 1-D frequency domain.

If $0 \notin L^2$ there are two possible modifications. Information may be stored in lookup tables (one table for each input, intermediate, and output pass of the FFT algorithm) indicating which branches of the signal flow graph need to be traversed to compute the set $\{X_L(k_1)\}$. The second modification, which we adopt here, relies on the DFT modulation theorem

$$W_N^{in} x(n) \leftrightarrow X(k + i). \tag{7}$$

Each input data point is multiplied by $W_N^{in} = \exp[-j(2\pi/N)in]$, where i corresponds to the index of the first coefficient in the set X_L. The complex modulation of the input data points may be incorporated into the first pass of the FFT algorithm. The output-pruned FFT algorithm is used to obtain output points which are, as a result of the complex modulation, circularly shifted up i locations. If necessary, the output points are shifted down i locations to place the DFT coefficients in their proper place.

The modifications discussed above are shown in Fig. 2 where partial butterflies are present in the second and third passes and the input data is modulated by W^{3n}. To place the $\{X_L(k_1)\}$ coefficient set at its proper location in the

discrete frequency domain, the set is circularly shifted down three places.

Now consider 2-D FFT output pruning using the row-column 2-D FFT algorithm. The row and column DFT's are computed using the output-pruned 1-D FFT algorithm. A typical example of a localized region of support L^2 is shown in Fig. 1 with the rows indexed by k_1 and the columns by k_2. The localized regions of support for the row- and column-pruned 1-D FFT's are determined as follows.

Algorithm 1: The Output-Pruned Row-Column 2-D FFT

1) Row Output-Pruned FFT's: The input data to the column FFT routines is shown in Fig. 3. For each row index k_1, the DFT coefficients are computed for the localized region of support $L^1_{k_1} \equiv \{k_2 | k_2^{\text{start}} \le k_2 \le k_2^{\text{end}}\}$. The indices k_2^{start} and k_2^{end} are passed to each call of the row output-pruned 1-D FFT routine within the output-pruned row-column 2-D FFT routine.

2) Column Output-Pruned FFT's: For each column index k_2, the DFT coefficients are computed for the localized region of support $L^1_{k_2} = \{k_1 | k_1^{\text{start}}(k_2) \le k_1 \le k_1^{\text{end}}(k_2)\}$. The indices k_1^{start} and k_1^{end} are functions of the columns index, indicating their dependence on L^2. The 2-tuples $(k_1^{\text{start}}(k_2), k_1^{\text{end}}(k_2))$ are passed to the column output-pruned 1-D FFT routine for each column k_2. Elements of one such 2-tuple are shown in Fig. 4. □

This completes the description of the output-pruned row-column 2-D FFT. The algorithm for an input-pruned row-column 2-D FFT routine is developed in a similar fashion.

The computational savings for the output-pruned 1-D FFT routine can easily be calculated. However, due to the fact that the 2-tuples $(k_1^{\text{start}}(k_2), k_1^{\text{end}}(k_2))$ are a function of the column index k_2, the overall computational savings for the pruned row-column 2-D FFT routine are not readily calculated. It is easier to experimentally measure the savings for a typical localized region of support L^2. This is done in Section IV.

The row-column approach implies that L^2 is contiguous in the directions of the columns and rows. That is, the required coefficients on L^2 must be two-connected (have two neighbors) in the row and column directions, except, of course, the first and last coefficients. Thus, localized regions of support resembling those shown in Fig. 5 are not allowed. The pruned row-column 2-D FFT algorithm might be modified to allow such regions, but at the cost of substantial computational overhead.

B. Rectangular Pruning of the Radix-(2 × 2) 2-D FFT

The rectangularly pruned radix-(2 × 2) 2-D FFT algorithm [5] is very similar to the pruned row-column 2-D FFT algorithm of Section II-A. The main difference is that a radix-(2 × 2) 2-D butterfly based 2-D FFT algorithm is used. The 2-D butterfly, shown in Fig. 6, is based on the following decomposition of (2) for $N_1 = N_2$

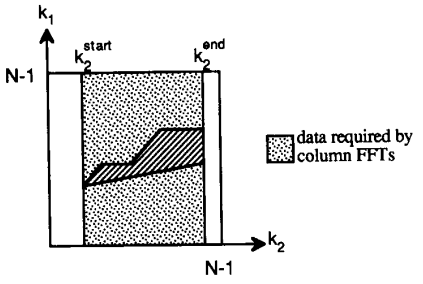


Fig. 3. The region of support for the data output by the row FFT's of the output-pruned row-column 2-D FFT routine.

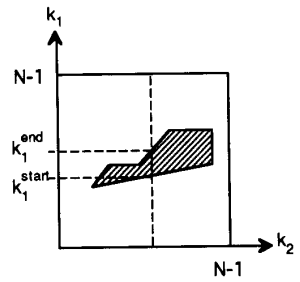


Fig. 4. The start and end points of the localized region of support $L^1_{k_2}$ for one column output-pruned 1-D FFT call within the output-pruned row-column 2-D FFT routine.

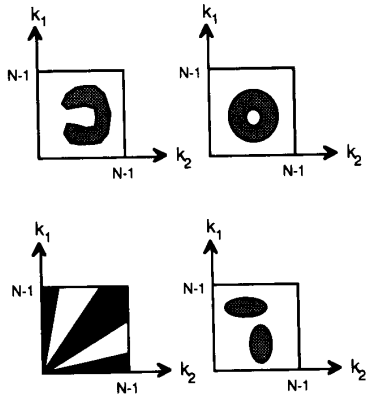


Fig. 5. Four regions of support ill-suited for use by the pruned row-column 2-D FFT algorithm.

$= N$ [6]:

$$X(k_1, k_2) = S_{00}(k_1, k_2) + W_N^{k_2} S_{01}(k_1, k_2) + W_N^{k_1} S_{10}(k_1, k_2) + W_N^{k_1 + k_2} \cdot S_{11}(k_1, k_2) \tag{8a}$$

$$X\left(k_1 + \frac{N}{2}, k_2\right) = S_{00}(k_1, k_2) + W_N^{k_2} S_{01}(k_1, k_2) - W_N^{k_1} S_{10}(k_1, k_2) - W_N^{k_1 + k_2} \cdot S_{11}(k_1, k_2) \tag{8b}$$

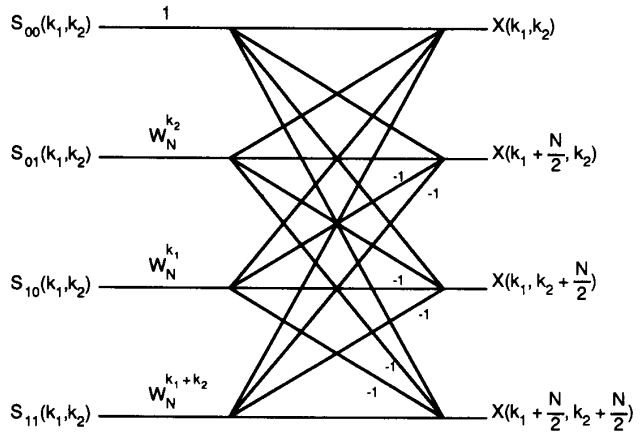


Fig. 6. A radix-(2 × 2) butterfly. Three complex multiplications and eight complex additions are required to calculate the output coefficients.

$$X\left(k_1, k_2 + \frac{N}{2}\right) = S_{00}(k_1, k_2) - W_N^{k_2} S_{01}(k_1, k_2) + W_N^{k_1} S_{10}(k_1, k_2) - W_N^{k_1 + k_2} \cdot S_{11}(k_1, k_2) \quad (8c)$$

$$X\left(k_1 + \frac{N}{2}, k_2 + \frac{N}{2}\right) = S_{00}(k_1, k_2) - W_N^{k_2} S_{01}(k_1, k_2) - W_N^{k_1} S_{10}(k_1, k_2) + W_N^{k_1 + k_2} \cdot S_{11}(k_1, k_2) \quad (8d)$$

where the S terms are calculated by breaking (2) into even/odd combinations of the 2-tuple (n_1, n_2),

$$S_{00}(k_1, k_2) \equiv \sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} x(2n_1, 2n_2) W_N^{2n_1k_1 + 2n_2k_2} \quad (9a)$$

$$S_{01}(k_1, k_2) \equiv \sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} x(2n_1, 2n_2 + 1) \cdot W_N^{2n_1k_1 + 2n_2k_2} \quad (9b)$$

$$S_{10}(k_1, k_2) \equiv \sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} x(2n_1 + 1, 2n_2) \cdot W_N^{2n_1k_1 + 2n_2k_2} \quad (9c)$$

$$S_{11}(k_1, k_2) \equiv \sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} x(2n_1 + 1, 2n_2 + 1) \cdot W_N^{2n_1k_1 + 2n_2k_2}. \quad (9d)$$

The 2-D DFT of the input $x(n_1, n_2)$ is calculated via a "divide and conquer" strategy based on (8) and (9).

Output pruning this 2-D FFT algorithm is similar to pruning the 1-D FFT algorithm (refer to Section II-A), except that the localized region of support L^2 is square, 2^m by 2^m in extent, where $2^m < N$ and $m \in \mathbb{N}$, and L^2 starts at the origin ($k_1 = k_2 = 0$). In the 2-D case, a signal flow graph (similar to Fig. 2) becomes too complicated to prune by hand. Instead, a graphical method based on (8) and 2-D matrices (lookup tables), one for each of the $\log_2(N)$ passes of the 2-D FFT algorithm, may be used. The indices of (8) are modified to be

$$X_p(k_1, k_2) = S_{00}(k_1, k_2) + W_N^{k_2} S_{01}(k_1, k_2) + W_N^{k_1} S_{10}(k_1, k_2) + W_N^{k_1 + k_2} \cdot S_{11}(k_1, k_2) \quad (10a)$$

$$X_p(k_1 + \alpha, k_2) = S_{01}(k_1, k_2) + W_N^{k_2} S_{01}(k_1, k_2) - W_N^{k_1} S_{10}(k_1, k_2) - W_N^{k_1 + k_2} S_{11}(k_1, k_2) \quad (10b)$$

$$X_p(k_1, k_2 + \alpha) = S_{00}(k_1, k_2) - W_N^{k_2} S_{01}(k_1, k_2) + W_N^{k_1} S_{10}(k_1, k_2) - W_N^{k_1 + k_2} \cdot S_{11}(k_1, k_2) \quad (10c)$$

$$X_p(k_1 + \alpha, k_2 + \alpha) = S_{00}(k_1, k_2) - W_N^{k_2} S_{01}(k_2, k_2) - W_N^{k_1} S_{10}(k_1, k_2) + W_N^{k_1 + k_2} \cdot S_{11}(k_1, k_2) \quad (10d)$$

where

$$\alpha = \frac{N}{2^{\log_2(N) - p + 1}}$$

$p \equiv$ pass number of the algorithm,

$$p = 1, 2, \cdots, \log_2(N).$$

According to (8) and (10), the butterfly performs an in-place calculation; that is, the output points replace the input points. The input and output points for each of the three passes necessary for an 8 × 8 2-D DFT are shown

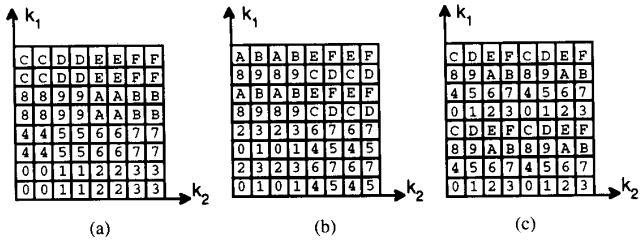


Fig. 7. The distribution of radix-(2 × 2) butterfly input and output points for the three passes of an 8 × 8 2-D DFT. (a) Pass = 1. (b) Pass = 2. (c) Pass = 3.

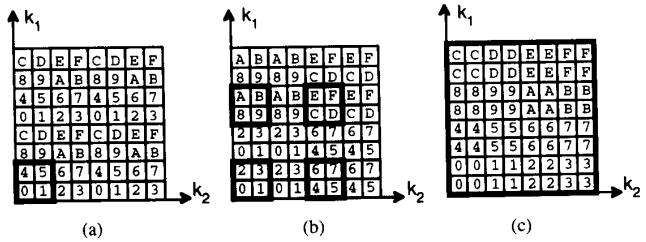


Fig. 8. The distribution of radix-(2 × 2) butterfly input and output points for the three passes of an 8 × 8 output-pruned 2-D DFT. The heavy lines outline the required output points computed in each pass. (a) The output X_L from the final pass of the algorithm (pass = 3). (b) The output X'_L of pass 2 necessary as input to pass 3 (pass = 2). (c) The output X''_L of pass 1 necessary as input to pass 2 (pass = 1).

in Fig. 7. The points used in each butterfly are denoted by hexadecimal numerals, for example, "A" = 1010_2 represents points involved in the tenth 2-D butterfly of the algorithm.

To illustrate rectangular output-pruning of the radix-(2 × 2) 2-D FFT algorithm, consider computing the two-by-two set of coefficients $\{X_L(k_1, k_2)\} = \{X_3(0, 0), X_3(0, 1), X_3(1, 0), X_3(1, 1)\}$. This set is shown in Fig. 8(a). Begin by considering the third (output) pass, $p = 3$. Only the zeroth, first, fourth, and fifth butterflies are required and for these butterflies only (10a) needs to be computed. Next consider pass two, $p = 2$. The output of pass two is used as input to pass three. By examining Fig. 8(a) and (10) we observe the set of intermediate points $\{X'_L(k_1, k_2)\} = \{X_2(0, 0), X_2(0, 1), X_2(1, 0), X_2(1, 1), X_2(4, 0), X_2(4, 1), X_2(5, 0), X_2(5, 1), X_2(0, 4), X_2(0, 5), X_2(1, 4), X_2(1, 5), X_2(4, 4), X_2(4, 5), X_2(5, 4), X_2(5, 5)\}$ is computed in pass two. This set is shown in Fig. 8(b), which indicates that all sixteen butterflies are required to calculate the set of intermediate points, but that only (10a) needs to be computed for each. Finally, Fig. 8(b) and (10) are examined to determine the required output of pass one, $\{X''_L(k_1, k_2)\}$ (and hence the input to pass two). To calculate the indicated points in pass two, all the output points of all the butterflies of pass one are required.

1) Rectangular Tiling: Consider now the pruning example shown in Fig. 1. Two problems are immediately obvious. First, the output localized region of support is not square and, second, it is not at the origin of the 2-D frequency space. The first problem is solved by calling

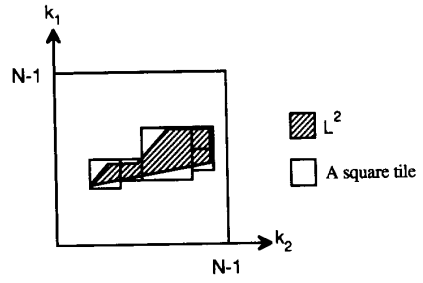


Fig. 9. A possible placement of tiles to facilitate the computation of DFT coefficients on the localized region of support L^2 of Fig. 1 using the rectangularly pruned radix-(2 × 2) 2-D FFT algorithm.

the rectangularly pruned radix-(2 × 2) 2-D FFT algorithm several times. Several different square subregions of support are used to cover the localized region of support L^2 in a tilewise fashion, as illustrated in Fig. 9. The second problem is solved using the 2-D DFT modulation theorem. The rectangularly pruned radix-(2 × 2) 2-D FFT algorithm computes coefficients on the square subregion, or tile, that starts at the origin of the frequency space. By modulating each input data piont by $W_N^{l_1 n_1 + l_2 n_2}$, where (l_1, l_2) is the index of the lower left corner of the tile at its final location in the frequency space, the coefficients generated by the rectangularly pruned radix-(2 × 2) 2-D FFT algorithm will correspond to the coefficients of the tile. It remains only to shift the square subregion up and right by l_1 and l_2 points, respectively. Before proceeding any fur-

ther with the algorithm development, a simple observation will show that this method is, in general, impractical.

Consider the typical tiling scheme shown in Fig. 9—the localized region of support L^2 is often irregularly shaped, requiring that several different square tiles be specified to cover the extent of L^2. The computation of the coefficients in each tile that does not start at the origin of the frequency space will require the complex modulation of each input data point on R^2. These operations may be incorporated in the butterflies of the first pass of the algorithm. There are $N^2/4$ butterflies per pass of a radix-(2 × 2) 2-D FFT algorithm and four input/output points per butterfly. Thus the required complex modulation results in $(N^2/4) \cdot 4 = N^2$ complex multiplications in the first pass (the three complex multiplications of the normal radix-(2 × 2) butterfly being incorporated into the modulation coefficients). The minimum number of complex multiplications required by the remaining passes of the rectangularly pruned radix-(2 × 2) 2-D FFT algorithm are for a tile size of (2 × 2). For this case, pass $p = \log_2(N)$ requires $(4 \cdot 3)$ complex multiplies, pass $p = (\log_2(N) - 1)$ requires $(16 \cdot 3)$ complex multiplies, and pass number i requires $(4^{\log_2(N) - i + 1} \cdot 3)$ complex multiplies. Thus, the minimum number of complex multiplications required to produce a (2 × 2) tile of output points, away from the origin of the 2-D frequency space, is

$$C^*_{\text{tile}} = N^2 + 3 \sum_{i=1}^{\log_2(N)-1} 4^{\log_2(N) - i + 1}$$

or

$$C^*_{\text{tile}} = N^2 + 3 \sum_{i=1}^{\log_2(N)-1} 4^i. \tag{11}$$

If there are T (2 × 2) tiles involved, then

$$C^*_{\text{tile}_{\text{total}}} = T\left(N^2 + 3 \sum_{i=1}^{\log_2(N)-1} 4^i\right) \tag{12}$$

complex multiplications are required. In comparison, the full radix-(2 × 2) 2-D FFT algorithm requires

$$C^*_{2\times 2_{\text{total}}} = \frac{3N^2}{4}(\log_2(N) - 1) \tag{13}$$

complex multiplications [6]. The required number of complex additions is similar for both algorithms. To achieve computational savings, we require

$$C^*_{\text{tile}_{\text{total}}} < C^*_{2\times 2_{\text{total}}}$$

or

$$T\left(N^2 + 3 \sum_{i=1}^{\log_2(N)-1} 4^i\right) < \frac{3N^2}{4}(\log_2(N) - 1). \tag{14}$$

For example, if $N = 256$, computational savings are achieved only if $T < 2.625$; there must be two or fewer (2 × 2) tiles! This result suggests that the rectangularly pruned radix-(2 × 2) 2-D FFT algorithm is not practical because the localized region of support L^2 is typically irregularly shaped (requiring multiple square tiles to cover it). As will be shown, an output-pruned 2-D FFT algorithm exists which does not require complex modulation or multiple invocations to compute DFT coefficients on square tiles, thereby avoiding the primary computational cost of the tiling approach.

III. Recursive Pruning of the Radix-(2 × 2) 2-D FFT

The discussion of the development of the rectangularly pruned 2-D FFT algorithm in Section II-B serves as the starting point for the development of the recursively pruned 2-D FFT algorithm. We begin by recalling the steps involved in the output pruning of a radix-(2 × 2) FFT algorithm for a square localized region of support L^2 with sides that are a power of two in length and having its lower left corner at the origin of the 2-D frequency space. This provides familiarity with the data flow in the algorithm. Then, by relaxing all constraints on the output localized region of support (except that $L^2 \subset R^2$) and examining pass by pass which data points (input, intermediate and output) are absolutely necessary for the calculation of the output points on L^2, regularities are observed that allow for a *recursive determination* of the necessary data points in each pass of the radix-(2 × 2) 2-D FFT algorithm. The locations of the necessary data points are kept in a 2-D logical matrix (one per pass) that is used as a lookup table within the 2-D FFT routine. The determination of the logical matrices is computationally inexpensive since it involves only data comparisons and the setting of logical flags. The computational savings that are achieved, in most cases, justify the increased algorithmic complexity.

To illustrate the development and usefulness of the recursively pruned radix-(2 × 2) 2-D FFT algorithm, consider an example that cannot be computed by either of the pruning algorithms of Section II. Assume an arbitrary localized region of support L^2 for the set $\{X_L(k_1, k_2)\}$, as shown in Fig. 10(a). Each of the output points is assigned a lowercase letter. The region L^2 is clearly not connected and is not square. We begin by determining which output-pass butterflies compute the output points on L^2. Recall the matrices in Fig. 7 and (10). Observe in Fig. 10(a) that, for example, points 'a' and 'j' in pass three may be calculated using (10a) and (10d) (which represent a half or partial butterfly) and require as input points $X_2(3, 3)$, $X_2(3, 7)$, $X_2(7, 3)$, and $X_2(7, 7)$. The input points required in pass three are the output points of pass two. These are shown in Fig. 10(b). Likewise, the input points of pass two are the output points of pass one and these are shown in Fig. 10(c). Finally, it is clear from Fig. 10(c) that all the signal domain points are required as input to pass one, as expected.

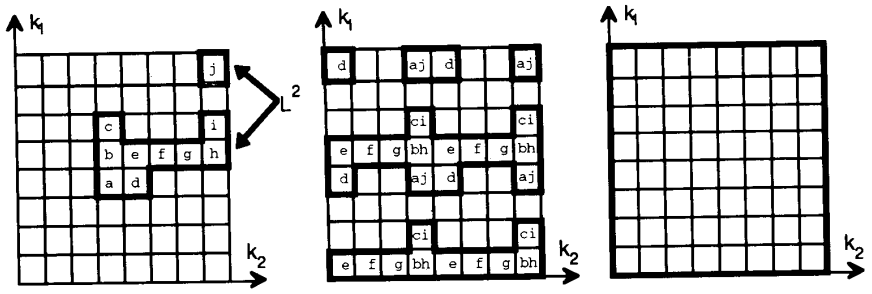


Fig. 10. (a) The required output points, "a" through "j," on the localized region of support L^2 (pass = 3). (b) The output points of pass two needed as input points to pass three (pass = 2). (c) The output points of pass one needed as input points to pass two. All $N^2/4$ full butterflies are required in pass one (pass = 1).

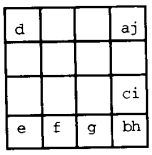


Fig. 11. The pattern of output points in pass 2 replicated in each quadrant of the N by N matrix.

TABLE I
THE OUTPUT POINTS TO BE COMPUTED BY EACH BUTTERFLY IN THE LOWER LEFT QUADRANT OF FIG. 10(b) FOR PASS $p=2$. THE COMBINATIONS OF (10) ARE ASSIGNED A LOGICAL CODE WHICH IS STORED IN THE LOGICAL MATRIX OF PASS 2, AS SHOWN IN FIG. 12.

	Required Output Points				
$k_1\ k_2$	$X_2(k_1, k_2)$	$X_2\left(k_1 + \frac{N}{2^\alpha}, k_2\right)$	$X_2\left(k_1, k_2 + \frac{N}{2^\alpha}\right)$	$X_2\left(k_1 + \frac{N}{2^\alpha}, k_2 + \frac{N}{2^\alpha}\right)$	Eq. (10)
0 0	yes	no	yes	no	a) c)
1 0	no	yes	no	no	b)
0 1	yes	no	yes	no	a) c)
1 1	no	no	yes	yes	c) d)

$\alpha = \log_2(N) - p + 1.$

Fig. 12. The logical matrix for pass two used to compute the output points of Fig. 11.

By working through the example above, and others involving $N > 8$ ($N = 16, 32, 64, \cdots$), it is apparent that the pattern of required output data points for a particular pass, p, repeats itself every $N/2^{\log_2(N)-p}$ rows and columns of the N by N output matrix. For example, in Fig. 10(b), where $p = 2$, the pattern shown in Fig. 11 is repeated every $N/2^{\log_2(N)-p} = 8/2^{3-2} = 4$ rows and columns; that is, in each quarter of the entire matrix. Furthermore, the dimensions of the repeated pattern in pass p indicate that

$$\left(\frac{N}{2^{\log_2(N)-p+1}}\right)^2$$

radix-(2×2) butterflies are required to compute the output data points in the repeated pattern. Again, for $p = 2$ and the pattern in Fig. 11, $(N/2^{\log_2(N)-p+1})^2 = (8/2^{3-2+1})^2 = 4$ butterflies are required. It is only necessary to store one code for each butterfly to indicate which branches of the butterfly are required to compute the desired output data point or points; that is, which of the equations (10) to compute. Since there are four output points per radix-(2×2) butterfly, there are $2^4 = 16$ possible combinations of equations (10). Each possibility is assigned a logical code and the code for each butterfly is stored in the logical matrix at a location corresponding to that butterfly. We conclude that, for each pass of the out-

TABLE II
THE RECURSIVE PROCEDURE THAT PRODUCES THE LOGICAL MATRICES FOR OUTPUT PRUNING OF THE RADIX-(2 × 2) 2-D FFT

A recursive algorithm to calculate the output-pruning logical matrices.

Assign logical codes to the butterfly equations

Code	Equation
0001_2	(10a)
0010_2	(10b)
0100_2	(10d)
1000_2	(10c)

```
quadrantMask ← {0001₂, 1000₂, 0010₂, 0100₂}
doAllButterflies ← 11111111₂

PROCEDURE setOutputPruneMatrixRecursive [
INPUT :     L2Matrix — a 2–D square array of byte containing indicators for the desired
                       and undesired output data points of the pass
            numberOfPoints — the dimension of L2Matrix

OUTPUT :    outputPruneMatrix — a 1–D array of 2–D square logical matrices, one for
                                each pass of the FFT algorithm ]

BEGIN
    passNumber ← LOG₂[numberOfPoints]
    IF passNumber = 1 THEN
        outputPruneMatrix[1][0][0] ← doAllButterflies
    ELSE
        nBy2 ← numberOfPoints/2
        allButterflies ← TRUE
        FOR k₁ = 0 TO k₁ < nBy2 STEP 1
            FOR k₂ = 0 TO k₂ < nBy2 STEP 1

            (*
             * Examine each output point in the radix–(2x2) butterfly.
             * If the point is required, add it to the marker.
             *)
                maskIndex ← 0
                marker ← 0
                FOR i = k₁ TO i < numberOfPoints STEP nBy2
                    FOR j = k₂ TO j < numberOfPoints STEP nBy2
                        IF L2Matrix[i][j] = a desired output point THEN
                            marker ← marker OR quadrantMask[maskIndex]
                        END IF
                        INCREMENT maskIndex BY 1
                    END FOR
                END FOR
            (*
             * The marker now indicates which of the four radix–(2x2) butterfly
             * output data points are required. Assign the marker to the
             * appropriate location in the outputPruneMatrix
             *)
                outputPruneMatrix[passNumber][k₁][k₂] ← marker
                IF allButterflies is TRUE AND marker does not indicate that
                  all output data points are required THEN
                    allButterflies ← FALSE
                END IF
            END FOR (* k₂ *)
        END FOR (* k₁ *)

        IF allButterflies is TRUE THEN
            outputPruneMatrix[passNumber][0][0] ← doAllButterflies
        END IF
    (*
     * Now find the outputPruneMatrix for the previous FFT pass using the
     * outputPruneMatrix of this pass
     *)
        setOutputPruneMatrixRecursive[ outputPruneMatrix[passNumber],
                                       numberOfPoints/2, outputPruneMatrix ]
    END IF

END PROCEDURE
```

The logical matrices for the entire FFT algorithm are computed as follows:

```
outputPruneMatrix[1] = a pointer to a 1x1 array of byte
```

TABLE II (*Continued.*)

outputPruneMatrix[2] = a pointer to a 2x2 array of byte
⋮ ⋮
outputPruneMatrix[p] = a pointer to a $\frac{N}{2^{\log_2(N)-p+1}} \times \frac{N}{2^{\log_2(N)-p+1}}$ array of byte

numberOfPoints ← dimensions of R^2
finalOutputMatrix ← L^2

setOutputPruneMatrixRecursive[finalOutputMatrix, numberOfPoints, outputPruneMatrix]

put pruned 2-D FFT algorithm, it is necessary to store $(N/2^{\log_2(N)-p+1})^2$ logical codes, each of which indicates which of the four radix-(2 × 2) butterfly output data points to compute.

Below, we show by example how to construct a logical matrix, the entries of which indicate which output data points to compute for each butterfly in a particular pass of the FFT algorithm.

Example 1

Let us construct the logical matrix for pass $p = 2$ and $N = 8$ and the output data points shown in Fig. 10(b). The pattern of required coefficient repeats every $N/2^{\log_2(N)-p} = 4$ rows and columns. The indices to consider are

$$k_1 = 0, \cdots, \frac{N}{2^{\log_2(N)-p+1}} - 1 = 1 \quad \text{and } k_2 = 0, 1. \tag{15}$$

The outputs of each butterfly in the lower left quadrant of Fig. 10(b) are now considered; that is, those butterflies having primary indices given by (15). The portion of each butterfly required to compute these output points is determined and a unique character code, indicating that portion, is entered in the logical matrix for pass two at the location given by the (k_1, k_2) tuple. The results are given in Table I. For example, for $k_1 = k_2 = 0$, examination of Fig. 10(b) (or Fig. 11) indicates that $X_2(0, 0)$ and $X_2(0, 2)$ are needed and therefore (10a) and (10c) need to be computed. This is the first table entry. We might assign the following character codes to the different combinations of equations (10) shown in Table I; $\{a), c)\}$ = 'ϵ', $\{b)\}$ = 'β', $\{c), d)\}$ = 'γ'. Then the logical matrix for pass two would look like Fig. 12. □

Notice that the input data points needed to compute the output data points (on L^2) are themselves the output data points of the next to last pass of the FFT algorithm. Likewise, the input data points needed to compute the output data points of the next to last pass are themselves the output data points of the second to last pass of the FFT algorithm, and so on. This suggests a recursive procedure to determine the logical matrix of each pass. The recursive algorithm is outlined in Table II.

The algorithm to compute the output data points on L^2 is summarized as follows.

Algorithm 2: The Recursively Output-Pruned 2-D FFT

1) *Preparation:* given L^2, determine the logical matrices that indicate the required output data points of passes 1, 2, · · · , $\log_2(N)$. Use the algorithm of Table II.
2) *Calculation of the output-pruned 2-D FFT:*
   FOR each pass in the radix-(2 × 2) FFT
     FOR each radix-(2 × 2) butterfly
       find (in the outputPrune Matrix) the logical code corresponding to the current butterfly
       compute the required butterfly output points (using equations (10)) indicated by the logical code
     END FOR
   END FOR □

In step 2) the algorithmic reference to the logical matrix manifests itself as logical comparisons and program branches to the appropriate piece of code.

The computational price of this algorithm is not very high because a *logical-comparison-and-branch is much faster than a complex multiplication or addition.* Furthermore, if multiple 2-D FFT's are to be performed for the same localized regions of support L^2, as is true for the examples in Section V, the cost of the extra programming logic becomes insignificant.

The main advantages of this algorithm are its generality and its avoidance of the use of the 2-D DFT modulation theorem. The algorithm is general in that there are no restrictions on the localized region of support L^2; it may have holes, be discontinuous or irregular in shape. The algorithm generates the minimum number of computations for any given L^2. Unfortunately, because of this generality, it is not possible to analytically determine the number of calculations for a given L^2.

The above procedure may be followed to develop an input-pruned 2-D FFT algorithm. The input-pruned algorithm is different in that it considers, in each pass, which combinations of the "S_{mn}" terms in equations (10) are required for a specified L^2, instead of considering in each pass which combinations of equations (10) are required, as does the output-pruned algorithm.

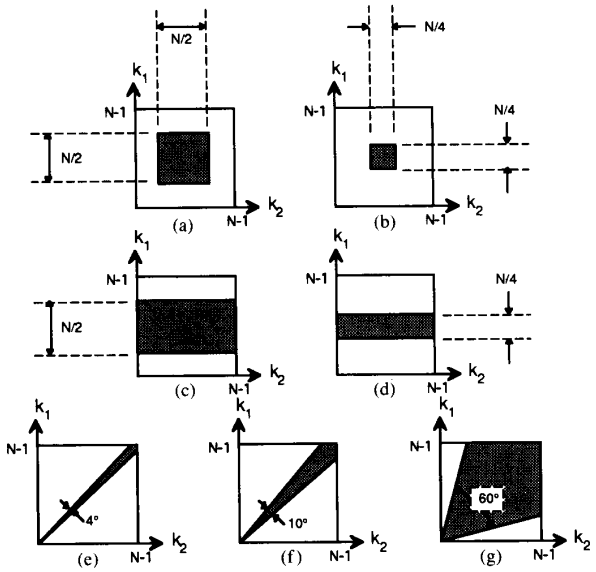


Fig. 13. The set of localized regions of support used to test the recursively pruned radix-(2 × 2) 2-D FFT, and the row-column pruned 2-D FFT algorithms.

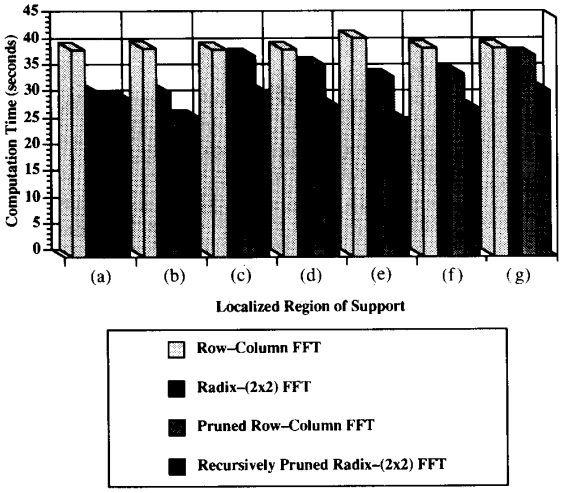


Fig. 14. A comparison of the execution times for the radix-(2 × 2) 2-D FFT, the row-column (R-C) 2-D FFT, the recursively pruned radix-(2 × 2) 2-D FFT, and the row-column pruned 2-D FFT algorithms for each localized region of support shown in Fig. 13.

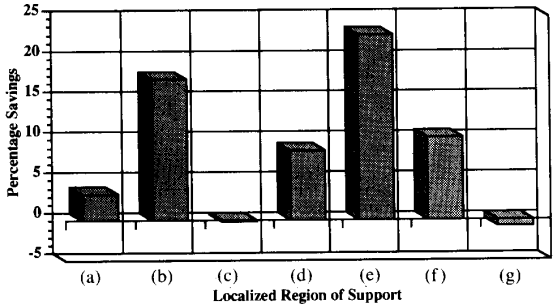


Fig. 15. The computational savings achieved using the recursively pruned radix-(2 × 2) 2-D FFT algorithm compared to using the radix-(2 × 2) 2-D FFT algorithm for each localized region of support shown in Fig. 13.

IV. Evaluations of 2-D FFT Output-Pruning Algorithms for Some Typical L^2

In this section, the recursively pruned radix-(2 × 2) 2-D FFT, the pruned row-column 2-D FFT, the unpruned radix-(2 × 2) 2-D FFT, and the unpruned row-column 2-D FFT algorithms are empirically compared in terms of computation time for a variety of typical localized regions of support L^2. Results are given graphically and discussed.

The tested regions of support are shown in Fig. 13. The 2-D DFT is defined over a region of support R^2 with $N_1 = N_2 = 64$. Execution times are shown in Fig. 14 for each of the algorithms discussed in Sections II and III (row-column 2-D FFT, radix-(2 × 2) 2-D FFT, pruned row-column 2-D FFT, recursively pruned radix-(2 × 2) 2-D FFT). The radix-(2 × 2) 2-D FFT algorithm is about 10% faster than the row-column 2-D FFT algorithm. In many computational environments a complex multiply requires as much time as a complex addition, and therefore we would expect an improvement of about 12% in computational speed [6]. The computational savings achieved using the recursively pruned radix-(2 × 2) 2-D FFT compared to using the (unpruned) radix-(2 × 2) 2-D FFT algorithm are shown in Fig. 15. The percentage savings was calculated using the method of [2].

The pruned row-column 2-D FFT algorithm clearly performs well when the localized region of support is square and is small. This is to be expected because, for these localized regions of support, many columns of coefficients are not computed. For all other localized regions of support, the recursively pruned radix-(2 × 2) 2-D FFT algorithm is the most computationally efficient algorithm. Of course, as the area of the localized region of support approaches that of R^2, the recursively pruned radix-(2 × 2) 2-D FFT algorithm requires as much time as the (unpruned) radix-(2 × 2) 2-D FFT algorithm.

V. Evaluation of the Recursively Pruned 2-D FFT in Practical MixeD [9]–[13] Signal Processing Algorithms

Here, we compare the computational efficiency of the radix-(2 × 2) 2-D FFT to that of the recursively pruned 2-D FFT in two practical 3-D signal processing algorithms. In the first problem, a 3-D spatially planar pulse [14] is enhanced using a combined 2-D DFT/1-D LDE discrete transform/spatiotemporal mixed domain (MixeD) filter [9]–[11]. The 2-D DFT coefficients corresponding to the spatially planar pulse signal are confined to a narrow wedge-shaped region of support. By using the recursively pruned 2-D FFT to compute the DFT coefficients, instead of the (unpruned) radix-(2 × 2) 2-D FFT, a computational saving [2] of about 20% is realized. In the second problem, the 2-D DFT is combined with a 1-D spec-

tral estimation method to detect 2-D objects moving in an image sequence and accurately estimate their trajectories [13]. The localized region of support for the DFT coefficients required by the algorithm is comprised of several, separate points. A computational saving [2] of about 83% is achieved by using the recursively pruned 2-D FFT algorithm.

A. The Enhancement of a 3-D Spatially Planar Signal Using a 2-D DFT/1-D LDE MixeD Filter

In this section the (unpruned) radix-(2 × 2) 2-D FFT algorithm and the recursively pruned radix-(2 × 2) 2-D FFT algorithm are compared computationally for an application where a 3-D spatially planar pulse is enhanced using a combined 2-D DFT/1-D LDE transform/spatio-temporal mixed domain (MixeD) filter. The filter design method is taken from the examples presented in [9]–[11], where a more complete discussion of the MixeD filter method can be found. The 2-D DFT/1-D LDE MixeD filter is implemented as follows.

Algorithm 3: The 2-D DFT/1-D LDE MixeD Filter

1) The 2-D DFT is applied along the spatial dimensions, n_1 and n_2, of the input signal $x(n_1, n_2, n_3)$, giving the MixeD signal $X(k_1, k_2, n_3)$.

2) The complex-valued sequences along n_3, one for each 2-tuple (k_1, k_2), are input to 1-D LDE filters. The result is the mixed domain sequence $Y(k_1, k_2, n_3)$. Note that there are N_1N_2 such 2-tuples and, hence, N_1N_2 1-D LDE filters are (in general) necessary.

3) The 2-D inverse DFT is applied along the k_1 and k_2 dimensions of $Y(k_1, k_2, n_3)$ giving the desired signal $y(n_1, n_2, n_3)$. □

Of primary concern here is the 2-D DFT portion of the MixeD filter, where the recursively pruned 2-D FFT algorithm is to be used. Next, enough information is presented to justify the choice of a localized region of support L^2.

The class of spatially planar (SP) signals is important in many multidimensional signal processing problems, including seismic data analysis and image enhancement. One temporal sample (i.e., an image frame) of a 3-D SP pulse signal shown in Fig. 17(a). In general, a 3-D discrete signal, $x(n_1, n_2, n_3)$, is SP if it constant in all surfaces

$$\alpha_1 n_1 + \alpha_2 n_2 + \alpha_3 n_3 = d, \qquad \forall d \in \mathbb{R} \tag{16}$$

with $\alpha_1, \alpha_2, \alpha_3 \in \mathbb{R}$ and $n_1, n_2, n_3 \in \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers.

It may be shown [14] that the 3-D DFT of a SP signal $x(n_1, n_2, n_3)$ yields 3-D discrete frequency domain coefficients, $X_L(k_1, k_2, k_3)$, that are zero everywhere except on the line $L(m_1, m_2, m_3)$, which has the localized region of support

$$L^3 = \left\{ (m_1, m_2, m_3) \middle| \frac{m_1}{\alpha_1} = \frac{m_2}{\alpha_2} = \frac{m_3}{\alpha_3}, \quad m_i \in \mathbb{Z} \right\}.$$

A passband enclosing the line $L(m_1, m_2, m_3)$ will selectively enhance the 3-D SP signal.

Ideal beam and cone shaped 3-D passband shapes have been described and discussed [14], [15]. A cone filter is generally preferred over a beam filter for selectively enhancing 3-D spatially planar signals because the cone shaped passband is equally selective in the 3-D frequency space, irrespective of the distance from the origin of the frequency space [15].

A *thin pyramidal* shaped passband, such as is shown in Fig. 16, can be used to approximate a cone shaped passband [11]. The thin pyramidal shaped passband is realized by the 2-D DFT/1-D LDE MixeD filter as follows. The 2-D DFT is applied along the spatial, n_1 and n_2, dimensions. The required 2-D localized region of support $\mathcal{L}^2$ is the projection of the thin pyramidal passband onto the $k_1 - k_2$ plane. $\mathcal{L}^2$ is an isosceles triangle centered on the line L^2 that is the projection of L^3 on to the $k_1 - k_2$ plane. Clearly, only the 2-D DFT coefficients on $\mathcal{L}^2$ are required. The recursively pruned 2-D FFT can be used to efficiently compute the 2-D DFT coefficients on $\mathcal{L}^2$. The 1-D LDE filters, which are applied to the complex-valued sequences along n_3, are confined to $\mathcal{L}^2$. It remains to characterize the LDE filter passbands.

The 2-D DFT of a spatially planar pulse yields complex-valued sequences in n_3 that are sinusoidal in the steady state [10], [12]. A general spatially planar signal is also a linear trajectory signal [14] and may therefore be written in the form

$$x(n_1, n_2, n_3) = x(n_1 - \delta_1 n_3, n_2 - \delta_2 n_3, 0) \tag{17}$$

where

$$\delta_1 = -\frac{\alpha_3}{\alpha_1} \quad \text{and} \quad \delta_2 = -\frac{\alpha_3}{\alpha_2}$$

and where $\delta_{1,2}$ are the vertical/horizontal (row/column) displacements of the SP image in pixels per frame. The shift property of the 2-D DFT is employed to show that the 2-D DFT of a SP (and therefore linear trajectory) signal $x(n_1, n_2, n_3)$ over the variables n_1 and n_2 is given by

$$\begin{aligned} X(k_1, k_2, n_3) &= \mathrm{DFT}_{(k_1,k_2)}[x(n_1 - \delta_1 n_3, n_2 - \delta_2 n_3, 0)] \\ &= X(k_1, k_2, 0)\, W_{N_1}^{\delta_1 k_1 n_3} W_{N_2}^{\delta_2 k_2 n_3} \end{aligned}$$

or, assuming $N_1 = N_2 = N$

$$X(k_1, k_2, k_3) = X(k_1, k_2, 0)\, W_N^{n_3(\delta_1 k_1 + \delta_2 k_2)} \tag{18}$$

where $\mathrm{DFT}_{(k_1,k_2)}[\cdot]$ is the forward 2-D DFT operator and, hence, $X(k_1, k_2, n_3) = \mathrm{DFT}_{(k_1,k_2)}[x(n_1, n_2, 0)]$. Equation (18) implies that the passband sequences $X(k_1, k_2, n_3)$ at each 2-tuple (k_1, k_2) are complex sampled sinusoids that may be selectively transmitted by employing LDE filters that are characterized by narrow-band bandpass magnitude frequency response functions having normalized center frequencies given by

$$\nu(k_1, k_2) = \frac{2}{N}(\delta_1 k_1 + \delta_2 k_2). \tag{19}$$

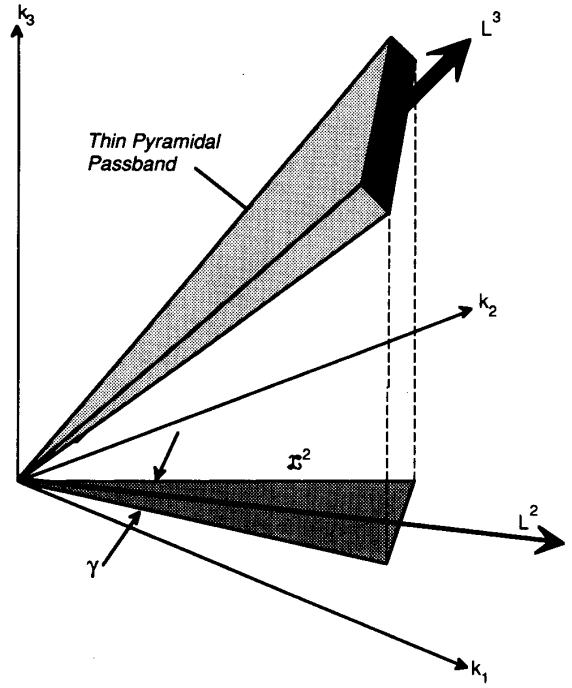


Fig. 16. Thin pyramidal MixeD filter passband. The nonzero impulse response sequence 2-tuples are within the shaded fan-shaped localized region of support $\mathcal{L}^2$ on the $k_1 - k_2$ plane.

We choose the bandwidths $B(k_1, k_2)$ of these narrow-band bandpass LDE's to be proportional to the center frequencies $\nu(k_1, k_2)$ [10], [12], so that

$$B(k_1, k_2) = K\nu(k_1, k_2),$$
$$K > 0,\ K \in \mathbb{R}, \text{ and } K \text{ is constant.} \tag{20}$$

Since the LDE filters are applied along n_3, the 1-D bandwidths given in (20) correspond to the wedge-shaped portion of the desired thin pyramidal passband in the k_3 direction (see Fig. 16).

This completes the description of the MixeD filter.

B. MixeD Filter Outputs and Execution Timing Comparison

In this section, the MixeD filter is used to enhance a 3-D spatially planar pulse in the presence of another, similar pulse and Gaussian noise. The MixeD filter algorithm is implemented once with the radix-(2 × 2) 2-D FFT algorithm, and once with the recursively pruned 2-D FFT algorithm and $\mathcal{L}^2$ as previously defined (see Fig. 16). The total execution times are compared.

Consider a 3-D input sequence given by

$$x(n_1, n_2, n_3) = x_D(n_1, n_2, n_3) + x_U(n_1, n_2, n_3) \tag{21}$$

where $x_D(n_1, n_2, n_3)$ is a 3-D spatially planar pulse having an intensity value 127 within a distance of 10 voxels (i.e., volume elements) of the 3-D plane $0.47n_1 - 1.00n_2 - 0.23n_3 = -64$, and an intensity of zero otherwise. The 3-D pulse is therefore 20 voxels thick about the plane and is shown in Fig. 17(a), spatially bounded in n_1 and n_2. The 3-D signal $x_U(n_1, n_2, n_3)$ is defined as zero-mean Gaussian random noise having a voxel intensity variance of 448, corresponding to a signal to noise ratio over the volume of the 3-D pulse $x_D(n_1, n_2, n_3)$ equal to -10.5 dB, superimposed on an undesired 3-D pulse $x'(n_1, n_2, n_3)$ that is identical to $x_D(n_1, n_2, n_3)$ except that it surrounds the plane $0.47n_1 - 1.00n_2 - 2.00n_3 = -130$. The undesired pulse $x'(n_1, n_2, n_3)$ is shown in Fig. 17(b), spatially bounded in n_1 and n_2. This is a good example of a spectral filtering problem that does not lend itself to the enhancement of $x_D(n_1, n_2, n_3)$ using 1-D or 2-D methods because $x_D(n_1, n_2, n_3)$ and $x'(n_1, n_2, n_3)$ have identical magnitude spectra in their 1-D and 2-D DFT's. A sample frame of $x(n_1, n_2, n_3)$, for $n_3 = 60$, is shown in Fig. 17(c) where the 3-D pulses are not visible in the noise. The signal is spatially and temporally bound with $N_1 = N_2 = 128$ and $N_3 = 100$.

The MixeD filter design parameters are chosen as follows. All the 1-D LDE filters are bandpass with center frequencies determined by (19), have second-order analog prototype Butterworth transfer functions, and are designed by bilinear transformation. They have bandwidth proportionality constants $K = 0.05$. The fan-shaped region of support on the $k_1 - k_2$ plane is chosen to have a half angle γ of five degrees, as shown in Fig. 16. The LDE filters were applied only to the MixeD sequences on the region of support $\mathcal{L}^2$.

The typical output sequence $y(n_1, n_2, n_3)$ from the MixeD filter is shown in Fig. 17(d) for the 60th frame, $y(n_1, n_2, 60)$. The 3-D pulse $x_D(n_1, n_2, n_3)$ is clearly recovered in this image, with the signal-to-noise ratio improving from -10.5 to 18 dB, while the second 3-D pulse $x'(n_1, n_2, n_3)$ is attenuated by 11 dB.

The execution times of the MixeD filters using unpruned and pruned 2-D FFT algorithms are shown in Table III. Two methods were used [2], [3] to compute the percentage time savings.

C. MixeD Moving Object Detection and Trajectory Estimation [13]

A method that detects 2-D objects moving on straight lines and at constant velocities (that is, having linear trajectories [14]), and that is able to estimate their trajectories is useful in applications such as night-sky satellite tracking, land and air vehicle monitoring, and storm cloud tracking. The MixeD solution to the problem [13] combines the 2-D DFT with a 1-D spectral estimation method, such as the modified forward-backward linear prediction (FBLP) method [17]. The details of the MixeD moving object detection and (linear) trajectory estimation method

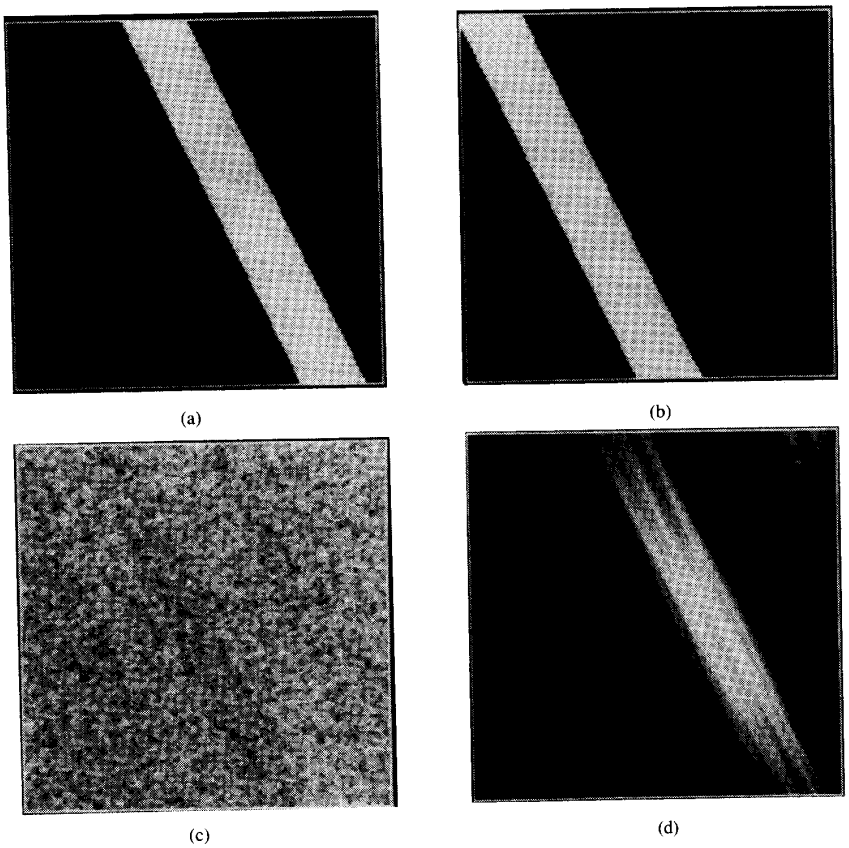


Fig. 17. Frame 60 of (a) the desired spatially planar (SP) input signal $x_D(n_1, n_2, n_3)$, (b) the undesired SP input signal $x'(n_1, n_2, n_3)$, (c) the input signal $x(n_1, n_2, n_3)$, and (d) the output signal $y(n_1, n_2, n_3)$.

TABLE III
COMPARISON OF EXECUTION TIMES FOR MIXED FILTERS USING UNPRUNED AND RECURSIVELY PRUNED 2-D FFT ALGORITHMS. THE FILTERS WERE USED TO ENHANCE THE SIGNAL $x_D(n_1, n_2, n_3)$ OF THE INPUT SEQUENCE EQUATION (21)

MixeD Filter Design	Execution Time (Seconds)	$t_r \equiv \frac{t}{t_{\max}}$	Time Savings $t_S[2] \equiv 1/t_r - 1$	$t_S[3] \equiv 1 - t_r$
Radix-(2 × 2) 2-D FFT LDE filters on $\mathcal{L}^2$	2906.9	$\frac{2906.9}{2906.9}$	0.0%	0.0%
Recursively Pruned 2-D FFT, LDE filters on $\mathcal{L}^2$	2417.4	$\frac{2417.4}{2906.9}$	20.2%	16.8%

are presented elsewhere [13]. Here, we present enough of the method to motivate the use of the recursively pruned 2-D FFT algorithm.

Consider the 3-D input sequence $x(n_1, n_2, n_3)$ as a sequence of discretely sampled images in n_3. A spatiotemporal domain model for 2-D linear trajectory objects is

$$x(n_1, n_2, n_3) = \sum_{s=1}^{O} o_s(n_1 - \delta_{s1}n_3, n_2 - \delta_{s2}n_3, 0) \quad (22)$$

where $n_1, n_2, n_3 \in \mathbb{N}$, $o_s(n_1, n_2, 0)$ is the sth of O objects at $n_3 = 0$, and δ_{s1} and δ_{s2} are the sth object's vertical and horizontal displacement coefficients, respectively, in pixels per frame (ppf). It is easily shown [14] that the energy of each moving object is confined to planes in the 3-D discrete frequency domain that satisfy

$$\delta_{s1}k_1 + \delta_{s2}k_2 + k_3 = mN, \qquad m, k_1, k_2, k_3 \in \mathbb{N} \quad (23)$$

when $N_1 = N_2 = N_3 = N$. The planes for which $m \neq 0$ are aliased versions of the plane for $m = 0$. However, they are of little consequence because the majority of a typical object's energy is concentrated within a circle of radius $\pi/3$ centered at the origin of the plane for which $m = 0$. We hereafter consider only the plane for which $m = 0$.

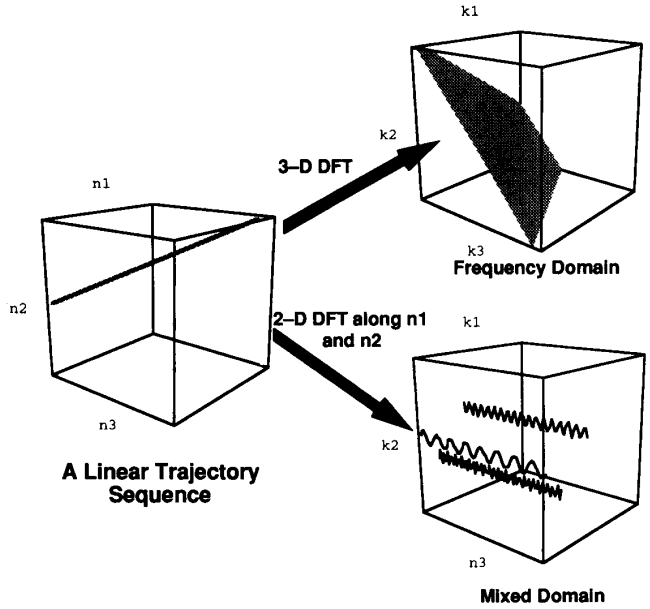


Fig. 18. A linear trajectory object in the spatiotemporal domain and its representation in the frequency domain and in the mixed domain.

When the 2-D DFT is applied to $x(n_1, n_2, n_3)$ along n_1 and n_2, that is,

$$X(k_1, k_2, n_3) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2, n_3) \cdot \exp\left[-j\frac{2\pi}{N}(n_1k_1 + n_2k_2)\right] = \sum_{s=1}^{O} O_s(k_1, k_2, 0) \exp\left[-j\frac{2\pi}{N} \cdot (\delta_{s1}k_1 + \delta_{s2}k_2)n_3\right] \quad (24)$$

the result is 1-D complex-valued sinusoidal sequences along n_3 at every (k_1, k_2) 2-tuple on the $k_1 - k_2$ plane. This is illustrated in Fig. 18.

If three points lying on a plane can be found, the equation for that plane can be determined. Then, using (22) and (23), the coefficients, δ_{s1} and δ_{s2}, that describe the trajectory of the sth object can be found. Three points can be found using MixeD sequences at selected (k_1, k_2) 2-tuples as input to the modified FBLP spectral estimation method. For a particular (k_1, k_2) 2-tuple, the modified FBLP method returns frequency estimates for the O sinusoids in the MixeD sequence along n_3. From (24), the estimate of the sth frequency is

$$\hat{\nu}_s(k_1, k_2) \approx \frac{2}{N}(\delta_{s1}k_1 + \delta_{s2}k_2). \quad (25)$$

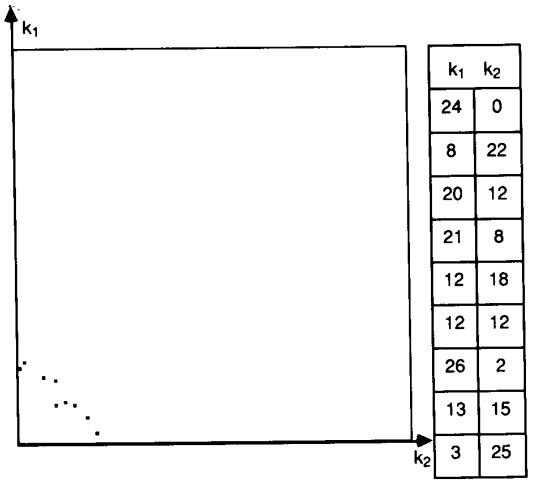
Then, as mentioned, the set of three (or more) points $\{(k_1, k_2, \hat{\nu}_s(k_1, k_2))\}$ can be used to determine the equation of the plane that corresponds to the sth object and, hence, that object's trajectory coefficients.

The locations of the selected MixeD sequences on the $k_1 - k_2$ plane define the localized region of support L^2. In practice, ten or so MixeD sequences are used. L^2 typically resembles Fig. 19. The 2-D DFT coefficients for L^2 can be efficiently computed by the recursively pruned 2-D FFT algorithm.

D. MixeD Moving Object Detection and Trajectory Estimation Results and Execution Timing Comparison

Here, we use the MixeD moving object detection and trajectory estimation algorithm to track three LT objects in an image sequence. The algorithm is implemented once with the radix-(2 × 2) 2-D FFT algorithm, and once with the recursively pruned 2-D FFT algorithm. The execution times are compared.

The experiment is identical to the example presented in [13]. The input sequence was $N_1 \times N_2 = 128 \times 128$ pixels spatially and $N_3 = 100$ frames in length. It consisted of three LT objects. The trajectory velocity and direction are defined as $v \equiv \sqrt{\delta_{s1}^2 + \delta_{s2}^2}$ and $\phi \equiv \tan^{-1}(\delta_{s2}/\delta_{s1})$, respectively. The trajectories were: object 1, $\phi = 40°$, $\nu = \sqrt{2}$ ppf; object 2, $\phi = 53°$, $\nu = 0.5$ ppf; object 3, $\phi = -47°$, $\nu = 3$ ppf. Objects 1 and 2 remained in the sequence for all 100 frames while object 3 left at frame 61. The trajectory estimates are shown in Fig. 20. The execution times are given in Table IV. Again, two methods were used [2], [3] to compute the percentage time savings. As expected, the algorithm execution was much faster using the recursively pruned 2-D FFT.



k_1	k_2
24	0
8	22
20	12
21	8
12	18
12	12
26	2
13	15
3	25

Fig. 19. A localized subregion L^2 for the MixeD moving object detection and trajectory estimation algorithm. The locations of the points that constitute L^2 are given in the table.

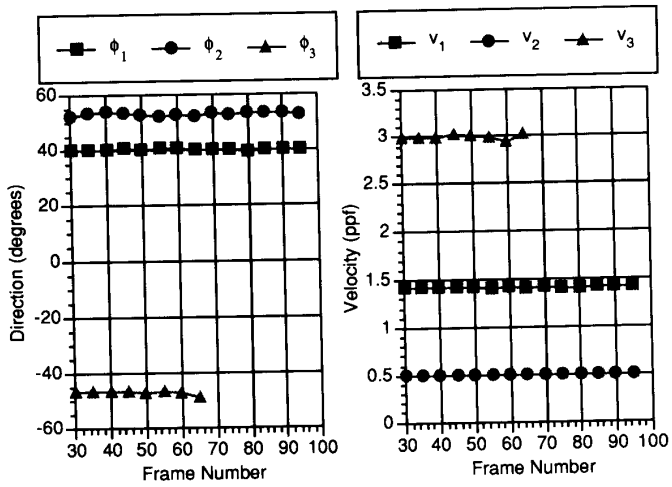


Fig. 20. The estimated trajectory directions and velocities for the three 2-D linear trajectory moving objects.

TABLE IV
COMPARISON OF EXECUTION TIMES FOR THE MIXED MOVING OBJECT DETECTION AND TRAJECTORY ESTIMATION ALGORITHM USING UNPRUNED AND RECURSIVELY PRUNED 2-D FFT ALGORITHMS

MixeD Moving Object Detection and Trajectory Estimation	Execution Time (Seconds)	$t_r = \frac{t}{t_{max}}$	Time Savings $t_S[2] \equiv 1/t_r - 1$	Time Savings $t_S[3] \equiv 1 - t_r$
Radix-(2 × 2) 2-D FFT	1694.6	$\frac{1694.6}{1694.6}$	0.0%	0.0%
Recursively Pruned 2-D FFT	924.8	$\frac{924.8}{1694.6}$	83.3%	45.4%

VI. CONCLUSIONS

For any localized region of support that is not square, the recursively pruned radix-(2 × 2) 2-D FFT algorithm is computationally more efficient than other 2-D FFT pruning algorithms [5], [7], [8], especially those based on existing 1-D FFT pruning algorithms [2]–[4]. Two practical, MixeD signal processing [9]–[13] examples were presented which illustrate the computational savings that can be achieved using the recursively pruned 2-D FFT algorithm. Other applications, involving many 2-D DFT operations and multiple or irregularly shaped localized re-

gions of support (L^2), should also be able to benefit from using the recursively pruned 2-D FFT algorithm.

The recursively pruned algorithm is based on a radix-(2×2) 2-D FFT algorithm. This algorithm is faster than the row-column decomposition based 2-D FFT algorithm since it requires 25% fewer complex multiplications. This accounts for a large percentage of the computational gain of the pruning algorithm. By basing the recursively pruned algorithm on a 2-D FFT algorithm of some other radix, say radix-(4×4), or on a combined factor (split-radix) algorithm [16], even larger computational gains might be realized. For example, a combined-factor algorithm requires only 36% of the complex multiplications required by a radix-2 row-column decomposition algorithm.

Because many sinusoidal transforms have fast algorithms similar to those used to compute the DFT, we believe that the recursive pruning method may be adapted to include other transforms [18]. Recursive pruning may also be extended to higher dimensional (>2) transforms.

Acknowledgment

The authors wish to thank S. T. Nichols and N. R. Bartley for their helpful suggestions concerning this contribution.

References

[1] E. O. Brigham, *The Fast Fourier Transform and Its Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[2] J. D. Markel, "FFT pruning," *IEEE Trans. Audio Electroacoust.*, vol. AU-19, pp. 305-311, Dec. 1971.

[3] D. P. Skinner, "Pruning the decimation-in-time FFT algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-24, pp. 193-194, Apr. 1976.

[4] T. V. Screenivas and P. V. S. Rao, "FFT algorithm for both input and output pruning," *IEEE Trans. Acoust. Speech, Signal Processing*, vol. ASSP-27, pp. 291-293, June 1979.

[5] L. Capodiferro, "Two-dimensional FFT and FFT-pruned algorithms in the context of HDTV images," in *Signal Processing of HDTV*, L. Chiariglione, Ed. New York: Elsevier, 1988.

[6] D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[7] M. R. Smith and S. T. Nichols, "Efficient algorithms for generating interpolated (zoomed) MR images," *Mag. Res. Med.*, vol. 7, pp. 156-171, June 1988.

[8] T. Smit, M. R. Smit, and S. T. Nichols, "Efficient sinc function interpolation technique for center padded data," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 1512-1517, Sept. 1990.

[9] A. A. Choudhury and L. T. Bruton, "Multidimensional filtering using combined discrete Fourier transform and linear difference equation methods," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 523-531, Feb. 1990.

[10] R. W. Issler, "Tracking and enhancement of objects in image sequences using 3-D frequency planar combined DFT/LDE filters," M.Sc. thesis, Dep. Elec. Eng., Univ. of Calgary, June 1990.

[11] K. S. Knudsen and L. T. Bruton, "Mixed domain filtering of multidimensional signals," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 1, pp. 260-268, Sept. 1991.

[12] R. W. Issler and L. T. Bruton, "Tracking and enhancement of objects in image sequences using 3-D frequency planar combined DFT/LDE filters," in *Proc. IEEE Int. Symp. Circuits Syst.*, New Orleans, LA, May 1990, pp. 999-1002.

[13] K. S. Knudsen and L. T. Bruton, "Moving object detection and trajectory estimation in the transform/spatiotemporal mixed domain," in *Proc. 1992 IEEE Int. Conf. Acoust. Speech, Signal Processing*, San Francisco, CA, Mar. 23-26, 1992, vol. 3, pp. 505-508.

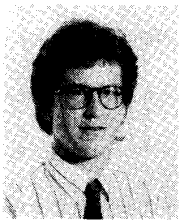
[14] L. T. Bruton and N. R. Bartley, "Three-dimensional image processing using the concept of network resonance," *IEEE Trans. Circuits Syst.*, vol. CAS-32, p. 664-672, July 1985.

[15] L. T. Bruton and N. R. Bartley, "The design of highly selective adaptive three-dimensional recursive cone filters," *IEEE Trans. Circuits Syst.*, vol. CAS-34, pp. 775-781, July 1987.

[16] H. R. Wu and F. J. Paoloni, "The structure of vector radix fast Fourier transforms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1415-1424, Sept. 1989.

[17] S. S. Haykin, *Adaptive Filter Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[18] N. Ahmed and K. R. Rao, *Orthogonal Transforms for Digital Signal Processing*. Heidelberg: Springer-Verlag, 1975.



Knud Steven Knudsen (S'82) was born in 1962 in Kelowna, British Columbia, Canada. He received the B.Sc. degree in computer engineering in 1984 and M.Sc. degree in applied sciences in medicine in 1987, both from the University of Alberta. Presently he is working towards the Ph.D. degree at the Department of Electrical and Computer Engineering, University of Calgary.

In 1986 he worked as a Research Assistant at the Universität Ulm, Germany. From 1987 to 1989 he was employed by MacDonald Dettwiler and Associates of Richmond, British Columbia. His research interests include multidimensional signal processing, nonlinear mathematics, neural networks, and genetic algorithms.

Leonard T. Bruton (M'71-SM'80-F'81) is a Professor of Electrical and Computer Engineering at the University of Calgary, Calgary, Alberta, Canada. His research interests are in the areas of analog and digital signal processing. He is particularly interested in the design and implementation of microelectronic digital filters and the applications of multidimensional circuit and systems theory to digital image processing.