Input and/or Output Pruning of Composite Length FFTs Using a DIF-DIT Transform Decomposition

Modesto Medina-Melendrez, Miguel Arias-Estrada, and Albertina Castro

***Abstract*—Pruned fast Fourier transforms (FFTs) can be efficient alternatives to compute DFTs when the input vector is zero padded and/or several output elements are not required. In this correspondence, a new method to prune composite length FFTs is proposed. The proposed pruning method uses decimation in frequency (DIF) and decimation in time (DIT) to decompose a DFT into stages of smaller DFTs. The pruning process is carried out on the input stage and the output stage of the decomposed transform. The proposed pruning method is flexible since it can perform input and/or output pruning over any composite length FFT, action that no other pruning method reported in the literature can carry out. Additionally, no restriction exists with the number of consecutive inputs and consecutive outputs that can be used. Finally, it is shown that the proposed pruning method generates efficient pruned power-of-three and power-of-two length FFTs.**

***Index Terms*—Fast Fourier transform, FFT pruning, transform decomposition.**

I. Introduction

The discrete Fourier transform (DFT) is one of the most important tools used in digital signal processing. A DFT can be implemented with efficient algorithms generally classified as fast Fourier transforms (FFTs). Typically, given an input vector of length N, the computation of a FFT of length N generates an output vector also of length N. However, there are applications in which an input vector with a length smaller than N has to be zero padded until the length N is reached and/or only a subset of the N output elements is required. In the literature, several methods have been developed to eliminate or reduce the computation with zero values of the required arithmetic operations and the computation of the non required outputs in FFT algorithms. These methods are collectively known as "FFT pruning."

Sorensen *et al.* [1] proposed a transform decomposition method to prune the input and another one to prune the output of any composite length FFT. They demonstrated that their pruned FFTs require less arithmetic operations than the pruned FFTs achieved by Markel in [2] and Skinner in [3] to prune the input or the output of power-of-two length FFTs. Since then, additional methods have been proposed to prune power-of-two length FFTs. Bouguezel *et al.* [4] used a decomposition process to prune the output of a FFT, the pruned FFT consists in the computation of a few stages of butterflies. In [5], Fan *et al.* presented a grouped scheme to prune the output of a FFT.

Few methods in the literature can carry out input and output pruning at the same time. Sorensen *et al.* [1] sketched how to prune the input and output of FFTs, but they concluded that their pruning method was less efficient than other pruning methods when both, the number of input and output elements are limited. They recommended the use of the method proposed by Sreenivas *et al.* in [6] to prune the input and output of power-of-two length FFTs. A more efficient pruning method for power-of-two length FFTs was proposed by Roche in [7].

In this correspondence, a new transform decomposition to perform input and/or output pruning of composite length FFTs is presented. The proposed method is a combination of the transform decompositions introduced by Sorensen *et al.* in [1], but with the extra capability to perform input and output pruning at the same time. We focus on cases where the input elements and the output elements are consecutive and start in the element zero. Any number of input elements and/or output elements can be used. An input stage, an output stage and an intermediate stage result from the decomposition transform and the pruning process. Pseudo codes to implement the input and the output stages are included. Finally, the efficiency of the resulting pruned FFTs is shown for power-of three length FFTs and for power-of-two length FFTs.

II. Pruning a DIF and DIT Transform Decomposition

The definition of the DFT of length N (DFT_N) can be expressed as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \quad k = 0, 1, \ldots, N-1 \tag{1}$$

where $W_N^{nk} = e^{-j2\pi nk/N}$ represents the kernel of the transform.

Lets consider that the number of consecutive input elements that can be different of zero is L_i and that there is an integer decomposition factor D_{ip}, such that $L_i \leq M = N/D_{ip}$ where M is an integer. Thus, it can be carried out the decimation defined by

$$\begin{aligned} k &= k_1 + D_{ip}k_2, & k_1 &= 0, 1, \ldots, D_{ip}-1 \\ & & k_2 &= 0, 1, \ldots, M-1 \\ n &= m_1 + Mm_2, & m_1 &= 0, 1, \ldots, M-1 \\ & & m_2 &= 0, 1, \ldots, D_{ip}-1 \end{aligned} \tag{2}$$

that introduced in (1) gives

$$X(k_1 + D_{ip}k_2) = \sum_{m_1=0}^{M-1} \left\{ W_N^{m_1k_1} \left[\sum_{m_2=0}^{D_{ip}-1} x(m_1 + Mm_2) W_{D_{ip}}^{m_2k_1} \right] \right\} W_M^{m_1k_2}. \tag{3}$$

Since $L_i \leq M$, m_2 is always zero and the DFTs of length D_{ip} $(\mathrm{DFT}_{D_{ip}}\mathrm{s})$ are not required; furthermore, n is always equal to m_1. Thus, (3) can be rewritten as

$$X(k_1 + D_{ip}k_2) = \sum_{n=0}^{M-1} \left[W_N^{nk_1} x(n) \right] W_M^{nk_2}. \tag{4}$$

The substitution of k by $k_1 + D_{ip}k_2$ generates a DIF structure. The elimination of the $\mathrm{DFT}_{D_{ip}}\mathrm{s}$ produces a reduction in the number of arithmetic operations required to compute the DFT_N.

If the number of required consecutive output elements is L_o, approximately or exactly L_o/D_{ip} output elements are computed from each DFT of length M in (4). Lets consider that there is another integer decomposition factor D_{op} such that $L_o/D_{ip} \cong P = M/D_{op}$ where P is an integer. Thus, it can be carried out the decimation defined by

$$\begin{aligned} n &= n_1 + D_{op}n_2, & n_1 &= 0, 1, \ldots, D_{op}-1 \\ & & n_2 &= 0, 1, \ldots, P-1 \\ k_2 &= \quad h_1 + Ph_2, & h_1 &= 0, 1, \ldots, P-1 \\ & & h_2 &= 0, 1, \ldots, D_{op}-1 \end{aligned} \tag{5}$$

Manuscript received October 02, 2008; accepted May 04, 2009. First published June 10, 2009; current version published September 16, 2009. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Markus Peuschel. This work was supported by the National Council of Science and Technology of México (CONACYT) by the Grant 161826.

M. Medina-Melendrez is with the Computer Science Coordination, National Institute of Astrophysics, Optics and Electronics, , Tonantzintla, Puebla, 72840, México, and also with the Culiacán Technological Institute, Culiacán, Sinaloa, México (e-mail: modesto@inaoep.mx).

M. Arias-Estrada is with the Computer Science Coordination, National Institute of Astrophysics, Optics and Electronics, 1 Luis Enrique Erro, Tonantzintla, Puebla 72840, México (e-mail: ariasmo@inaoep.mx).

A. Castro is with the Optics Department, National Institute of Astrophysics, Optics and Electronics, Tonantzintla, Puebla 72840, México (e-mail: betina@inaoep.mx).

Digital Object Identifier 10.1109/TSP.2009.2024855

that introduced in (4) gives

$$X\left(k_1 + D_{ip}(h_1 + Ph_2)\right) = \sum_{n_1=0}^{D_{op}-1}\left\{W_M^{n_1h_1}\left[\sum_{n_2=0}^{P-1}\left(W_N^{(n_1+D_{op}n_2)k_1}x(n_1+D_{op}n_2)\right)\times W_P^{n_2h_1}\right]\right\}W_{D_{op}}^{n_1h_2}. \tag{6}$$

Since $L_o/D_{ip} \cong P$, in (6) just a few outputs of each DFT of length D_{op} (DFT_{Dop}) are required. Thus, a FFT is an inefficient alternative to compute the DFT_{Dop}s since it computes all the possible outputs and not only the required. In (6), multiplications with twiddle factors are needed before computing the DFT_{Dop}s. These multiplications can be avoided if the twiddle factors are merged with the kernel of the DFT_{Dop}s. Thus, (6) can be rewritten as

$$\begin{aligned}X(k_1 + D_{ip}k_2) &= \sum_{n_1=0}^{D_{op}-1}\left\{W_M^{n_1k_2}z(k_2, n_1, k_1)\right\}\\ &= \sum_{n_1=0}^{D_{op}-1}\left\{W_M^{n_1k_2}\sum_{n_2=0}^{P-1}y(n_2, n_1, k_1)W_P^{n_2((k_2))_P}\right\}\end{aligned} \tag{7}$$

where $((k_2))_P = (k_2 \text{ modulo } P) = h_1$ and $y(n_2, n_1, k_1)$ is

$$y(n_2, n_1, k_1) = W_N^{(n_1+D_{op}n_2)k_1}x(n_1 + D_{op}n_2). \tag{8}$$

Summations of complex multiplications are required in (7) to compute the final outputs, instead of the twiddle factors and the DFT_{Dop}s that are required in (6). This substitution can produce a reduction in the number of arithmetic operations required to compute the DFT_N.

The substitution of n by $n_1+D_{op}n_2$ generates a DIT structure. Since a DIF structure is obtained with the former decimation and a DIT structure is obtained with the latter decimation, the FFT pruned with the proposed transform decomposition (TD) is referred as $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$.

In Fig. 1, a diagram of the $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ is shown. Three stages are identified: an input stage (implementation of (8)), an intermediate stage (the computation of the $D_{ip}D_{op}$ DFT_Ps) and an output stage (implementation of (7)).

In the input stage of Fig. 1, it is considered that $L_i = M$; but if $L_i < M$, the inputs of the DFT_Ps that correspond to $n_1 + D_{op}n_2 > L_i - 1$ are directly set to zero. In the output stage, the computation of all the possible final outputs is sketched; but in practice, only the L_o consecutive required final outputs will be computed. The array inside the argument of complex exponentials and inside the mapping of the final outputs means that many final outputs can depend on the same set of outputs of the intermediate stage. Then, depending on the required final output is the element of the array that will be used.

III. Implementation of the Input and the Output Stages

A. Implementation of the Input Stage

Given an element of $x(n)$, all the elements of $y(n_2, n_1, k_1)$ that depends on it are generated. The index n should be decomposed in the indexes $n_2 = \mathrm{floor}(n/D_{op})$ and $n_1 = ((n))_{Dop} = n - D_{op}n_2$ of the required array $y(n_2, n_1, k_1)$, such that $n = n_1 + D_{op}n_2$. Since k_1 can take D_{ip} different values, each element of $x(n)$ is used in D_{ip} DFT_Ps. The DFT_Ps with the same value of k_1 are referred as "set of D_{op} DFT_Ps." In this work, the values of W_N^{nk1} are pre-computed and stored into the array W of length N, which variation index is equal to $((nk_1))_N$.

The element $x(0)$ does not need to be multiplied and it is just mapped to each set of D_{op} DFT_Ps in the coordinate $(n_1 = 0, n_2 = 0)$. The inputs of each set of D_{op} DFT_Ps that corresponds to $1 \le n_1+D_{op}n_2 \le L_i - 1$ are generated with a direct method. In the proposed direct method, $x(n_1 + D_{op}n_2)$ is mapped to the coordinate (n_1, n_2) of the first set of D_{op} DFT_Ps ($k_1 = 0$); for the other sets of D_{op} DFT_Ps ($k_1 \neq 0$), $x(n_1 + D_{op}n_2)$ is first multiplied per W_N^{nk1} and then mapped. The inputs of the DFT_Ps that correspond to a value in the interval $L_i \le n_1 + D_{op}n_2 \le N - 1$ are directly set to zero. In Fig. 2, a pseudo code of a function to implement the input stage is shown. Since multiplications per one are not performed, the previously described implementation of the input stage requires $(L_i - 1)(D_{ip} - 1)$ complex multiplications.

B. Implementation of the Output Stage

Given a required final output $X(k)$, its index k is decomposed in the indexes $k_2 = \mathrm{floor}(k/D_{ip})$ and $k_1 = ((k))_{Dip} = k - D_{ip}k_2$. k_1 indicates which set of D_{op} DFT_Ps has to be used to compute the final output, and $((k_2))_P$ indicates which output of each DFT_P needs to be used. n_1 is the variation index of the summation of complex multiplications required to compute the final outputs. The values of W_M^{n1k2} can be obtained from the pre-computed array W using $((n_1k_2D_{ip}))_N$ as variation index.

According to (7), the computation of the first D_{ip} final outputs does not require complex multiplications since $k_2 = 0$. Thus, if $L_o \le D_{ip}$, only a summation of $D_{op} - 1$ complex values is required in the computation of each final output. If $L_o > D_{ip}$, the first D_{ip} required final outputs are computed as summations and the remaining final outputs have to be computed as summations of complex multiplications. The element in every summation that corresponds to $n_1 = 0$ does not need to be multiplied. Therefore, a direct implementation (direct method) of the output stage requires $L_o(D_{op} - 1)$ complex additions and zero complex multiplications if $L_o \le D_{ip}$; otherwise, it requires the same number of complex additions but $(L_o - D_{ip})(D_{op} - 1)$ complex multiplications.

Sorensen *et al.* in [1] proposed a method called filtering 2BF to compute a subset of final outputs from their transform decomposition. The structure of their transform decomposition is equal to the structure of the transform decomposition in Fig. 1 for each set of D_{op} DFT_Ps (given a value of k_1). The computation of each final output using the filtering 2BF method requires $6D_{op}$ real arithmetic operations. Using the direct method, this computation requires $(8D_{op}-8)$ real arithmetic operations if a complex multiplication is carried out with two real additions and four real multiplications. Then, only when $D_{op} > 4$ the filtering 2BF method requires fewer real arithmetic operations than using the direct method. Therefore, the direct method is used to implement the output stage if $1 < D_{op} \le 4$, otherwise, the filtering 2BF method is used. In Fig. 2, a pseudo code of a function to implement the output stage is shown.

IV. Values Selection for the Decomposition Parameters

The saving in number of required arithmetic operations from the input stage is due to the elimination of the DFT_{Dip}s. Then, this saving increase for large values of D_{ip}, but it is required that $D_{ip} \le N/L_i$. As discussed in Section II, a saving can be obtained from the output stage only if $L_o/D_{ip} \cong M/D_{op}$. This means that $D_{op} \cong D_{ip}M/L_o = N/L_o$. Thus, as a rough selection the pair of multiplicative factors of N nearest to $(N/L_i, N/L_o)$ is proposed to be used for (D_{ip}, D_{op}). A

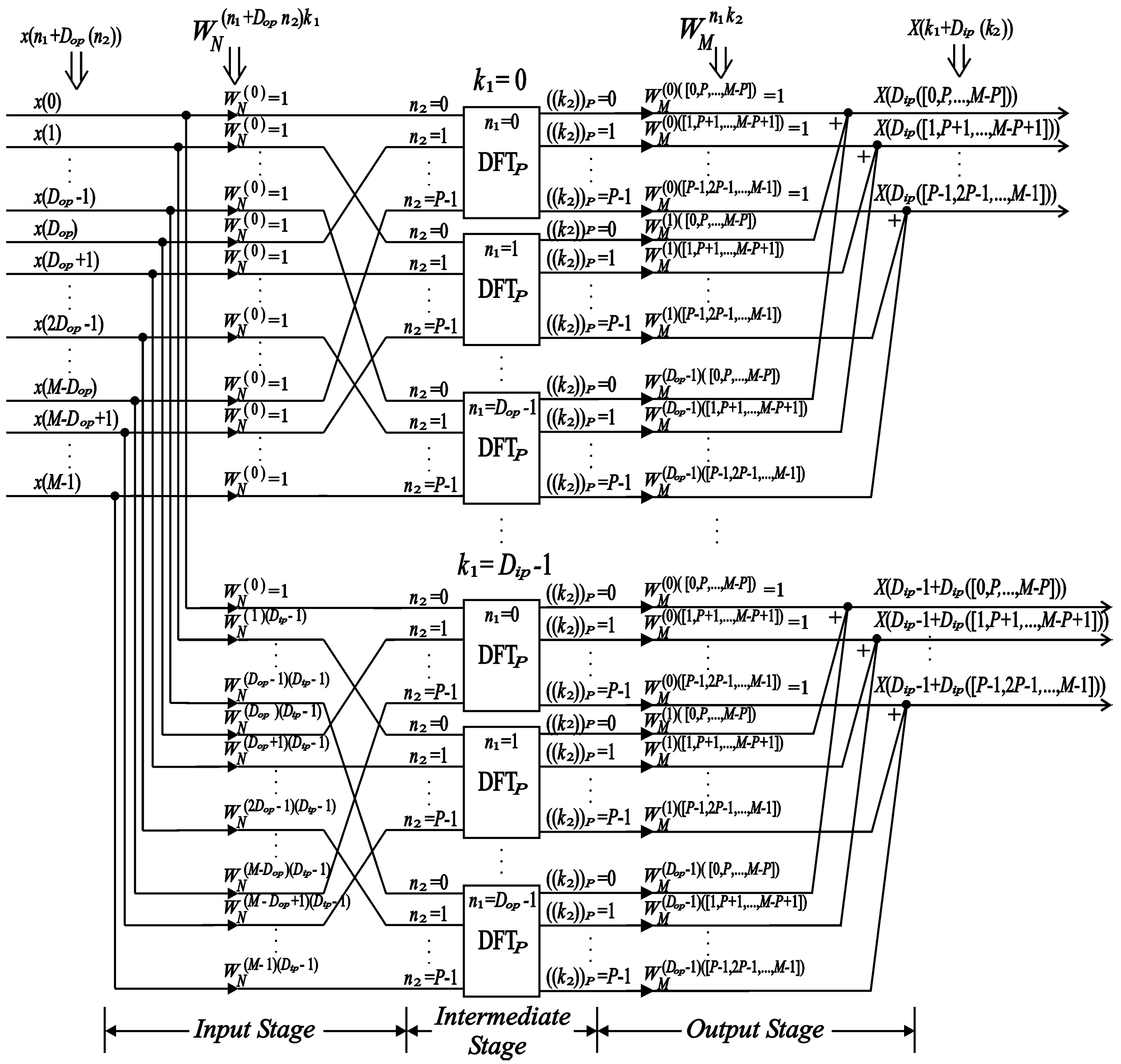

Fig. 1. General diagram of the $\mathbf{FFT}_{\mathrm{DIF-DIT-TD}}$.

deeper analysis or an exhaustive search is needed to obtain the exact pair of values for (D_{ip}, D_{op}) that achieve the minimum number of required arithmetic operations.

Since in current processors a multiplication and an addition require almost the same execution time, in this work, the number of required arithmetic operations refers to the total number of real arithmetic operations. The total number of required arithmetic operations to implement the $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ is the sum of the operations required to implement the input stage, the intermediate stage and the output stage:

$$\begin{aligned}\mathrm{OPER}_{\mathrm{tot}} &= \mathrm{OPER}_{\mathrm{input}} + D_{ip}D_{op}\mathrm{OPER}_{\mathrm{DFT}_P} + \mathrm{OPER}_{\mathrm{output}}\\ &= AD_{ip} + BD_{op} + D_{ip}D_{op}\left(C + \mathrm{OPER}_{\mathrm{DFT}_P}\right) + E \quad (9)\end{aligned}$$

where A, B, C, and E depends on certain conditions. These coefficients are shown in Table I.

The pair of values for D_{ip} and D_{op} that results in the lowest number of required arithmetic operations, computed using (9), should be adopted. This pair can be found by an exhaustive search when the number of arithmetic operations required by the DFT_Ps is known. Nevertheless, when the number of arithmetic operations required by the DFT_Ps is unknown, the use of the rough selection is suggested.

In the literature, algorithms that implement the DFTs of length P with $P\log P$ complexities have been reported. One of the most efficient FFT algorithms to compute power-of-two length DFTs is the split-radix FFT [8], this requires $4P\log_2(P) - 6P + 8$ arithmetic operations. In [9], a radix-3 FFT is proposed to compute power-of-three length DFTs, this requires $5.04P\log_2(P) - 4P + 4$ arithmetic operations.

Fig. 3 shows various curves representing the number of arithmetic operations required by the $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ when $D_{ip} = D_{op} = 1$ (without pruning) and when D_{ip} and D_{op} are those obtained by the rough selection and the exhaustive search. For the examples in Fig. 3(a), P is a power-of-two and the split-radix in [8] is used to compute the DFT_Ps. For the examples in Fig. 3(b), P is a power-of-three and the

```
/////////////////////////////////////////////////////////////////
//Function to generate the input stage//
[$y$]=InputStage($N,L_i,D_{ip},D_{op},x,W$)
{   //Generation of the elements of $y(n_2,n_1,k_1)$ that depends
    //on $x(0)$
    for ($k_1=0$; $k_1<=D_{ip}-1$; $k_1=k_1+1$)
        $y(0,0,k_1)=x(0)$;
    //Generation of the elements of $y(n_2,n_1,k_1)$ that depends
    //on $x(n)$ for $1<=n<=L_i-1$
    for ($n=1$; $n<=L_i-1$; $n=n+1$)
    {   $n_2$=floor($n/D_{op}$);            $n_1=n-D_{op}\times n_2$;
        //Mapping $x(n)$ to one input of the first set of $D_{op}$ DFT$_P$s
        $t_1=x(n)$;
        $y(n_2,n_1,0)=t_1$;
        //Multiplication of $x(n)$ per $W_N^{n\times k_1}$ and mapping to the
        //other sets of $D_{op}$ DFT$_P$s
        for ($k_1=1$; $k_1<=D_{ip}-1$; $k_1=k_1+1$)
            $y(n_2,n_1,k_1)=W(((n\times k_1))_N)\times t_1$; }
    //Zero padding of the elements of $y(n_2,n_1,k_1)$ for
    //$L_i<=n<=N-1$
    for ($n=L_i$; $n<=N-1$; $n=n+1$)
    {   $n_2$=floor($n/D_{op}$);            $n_1=n-D_{op}\times n_2$;
        for ($k_1=0$; $k_1<=D_{ip}-1$; $k_1=k_1+1$)
            $y(n_2,n_1,k_1)=0$;} }
```

```
/////////////////////////////////////////////////////////////////
//Function to generate the output stage//
[$X$]=OutputStage($N,L_o,D_{ip},D_{op},P,z,W$)
{   if ($L_o<=D_{ip}$)
        //Computation of $X(k)$ for $0<=k<=L_o-1$
        for ($k_1=0$; $k_1<=L_o-1$; $k_1=k_1+1$) {
            $X(k_1)=z(0,0,k_1)$;
            for ($n_1=1$; $n_1<=D_{op}-1$ ; $n_1=n_1+1$)
                $X(k_1)=z(0,n_1,k_1)+X(k_1)$;}
    else
    {   //Computation of $X(k)$ for $0<=k<=D_{ip}-1$
        for ($k_1=0$; $k_1<=D_{ip}-1$; $k_1=k_1+1$) {
            $X(k_1)=z(0,0,k_1)$;
            for ($n_1=1$; $n_1<=D_{op}-1$ ; $n_1=n_1+1$)
                $X(k_1)=z(0,n_1,k_1)+X(k_1)$;}
        //Computation of $X(k)$ for $D_{ip}<=k<=L_o-1$
        If ($D_{op}<=4$)
            //Using the direct method
            for ($k=1$; $k<=L_o-1$; $k=k+1$) {
                $k_2$=floor($k/D_{ip}$);            $k_1=k-D_{ip}\times k_2$;
                $X(k)=z(((k_2))_P,0,k_1)$;
                for ($n_1=1$; $n_1<=D_{op}-1$ ; $n_1=n_1+1$)
                    $X(k)=X(k)+z(((k_2))_P,n_1,k_1)\times W(((n_1\times k_2\times D_{ip}))_N)$;
        else
            //Using the filtering 2BF method
            for ($k=1$; $k<=L_o-1$; $k=k+1$) {
                $k_2$=floor($k/D_{ip}$);            $k_1=k-D_{ip}\times k_2$;
                $t_1=z(((k_2))_P,D_{op}-1,k_1)$;
                $t_4=t_1\times(2\times$real$(W(((k_2\times D_{ip}))_N)))$;
                for ($n_1=D_{op}-2$; $n_1>=1$; $n_1=n_1-1$) {
                    $t_2=t_1$;
                    $t_1=z(((k_2))_P,n_1,k_1)+t_4$;
                    $t_3=t_1\times(2\times$real$(W(((k_2\times D_{ip}))_N)))$;
                    $t_4=t_3-t_2$; }
                $t_5=z(((k_2))_P,0,k_1)+t_4$;
                $X(k)=t_5-t_1\times$conj$(W(((k_2\times D_{ip}))_N))$ ;}}}
```

Fig. 2. Pseudo codes to implement the input and the output stages.

TABLE I
COEFFICIENTS OF EQUATION (9)

A	B	C	E	*Use Conditions*
$6(L_i-1)$	$2L_o$	0	$6(1-L_i)-2L_o$	$D_{ip}\geq L_o$
$6L_i$	$8L_o$	-6	$6(1-L_i)-8L_o$	$D_{ip}<L_o;D_{op}\leq 4$
$6L_i-8$	$6L_o$	-4	$6(1-L_i)$	$D_{ip}<L_o;D_{op}>4$

radix-3 in [9] is used. In most of the cases, the numbers of required arithmetic operations achieved by the use of the decomposition parameters chosen by the rough selection and by the exhaustive search are the same. Although, the decomposition parameters chosen by the rough selection not always result with the minimum number of required arithmetic operations, their use can achieve considerable savings in comparison with the implementation of the FFT without pruning in most of the cases.

V. Comparison With Other Pruned FFTs

There are many methods to prune FFTs, but most of them are based on power-of-two length FFTs. In [1], the input pruning case and the output pruning case were compared, but since then, some new output pruning methods have been reported in [4] and [5]. Thus, comparisons among the input and output pruning methods and among the more recent output pruning methods are performed. The pruned power-of-two length FFTs that are compared are: Bouguezel-DIT [4], Grouped Scheme [5], Input&Output R2 [6], Input&Output SR [7] and the proposed $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$. The $\mathrm{DFT}_P\mathrm{s}$ required by the $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ are implemented with split-radix FFTs. Since the number of arithmetic operations required to compute the $\mathrm{DFT}_P\mathrm{s}$ is known, the values for D_{ip} and D_{op} are chosen by an exhaustive search.

Fig. 4 shows several curves of the number of the arithmetic operations required by the different pruned FFTs. In Fig. 4(a), it is shown that the $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ with output pruning requires fewer arithmetic operations than the pruned FFTs reported in [4] and [5]. In Fig. 4(b), it is shown that the $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ requires fewer arithmetic operations than the Input&Output R2 method in [6] that was suggested by Sorensen *et al.* in [1] to be used when an input and output pruning would be required. The savings obtained with the Input&Output SR method are similar to the obtained with the $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ since both are based in split-radix FFTs. If a more efficient algorithm than the split-radix in [8] were used to implement the $\mathrm{DFT}_P\mathrm{s}$, the $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ could require fewer arithmetic operations than the Input&Output SR.

Fig. 5 shows that the input and output pruning reduces the number of required arithmetic operations even more than the independent use of the input pruning and the output pruning when the $\mathrm{FFTD}_{\mathrm{IF-DIT-TD}}$ is used. Additionally, the Goertzel algorithm in [10] is compared. In most of the cases, the $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ requires fewer arithmetic operations than the Goertzel algorithm. Nevertheless, when L_i and L_o are very small, the Goertzel algorithm requires fewer arithmetic operations than the $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$. One advantage of the $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$, over the Goertzel algorithm, is that it achieves savings in all the range

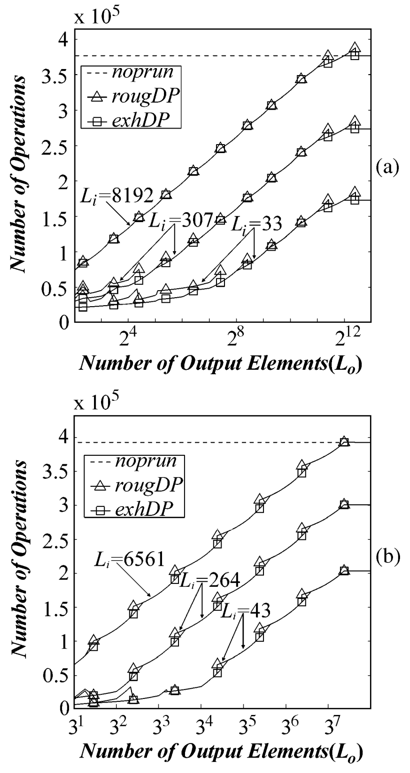

Fig. 3. Number of operations required by the $\mathbf{FFT}_{\mathrm{DIF-DIT-TD}}$ when $\boldsymbol{D}_{ip}$ and $\boldsymbol{D}_{op}$ are: $\boldsymbol{D}_{ip} = \boldsymbol{D}_{op} = \mathbf{1}$ $(\boldsymbol{noprun})$, chosen by the rough selection $(\boldsymbol{rougDP})$ and by the exhaustive search $(\boldsymbol{exhDP})$; (a) $\boldsymbol{N} = \mathbf{8192}$; (b) $\boldsymbol{N} = \mathbf{6561}$.

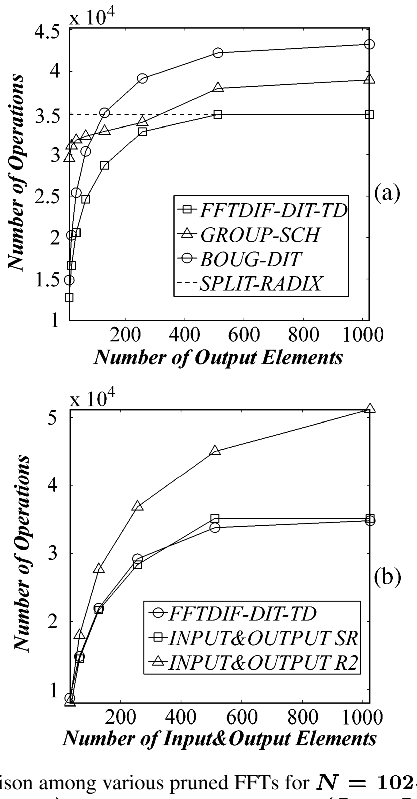

Fig. 4. Comparison among various pruned FFTs for $\boldsymbol{N} = \mathbf{1024}$. (a) Output pruning $(\boldsymbol{L}_i = \mathbf{1024})$; (b) input and output pruning $(\boldsymbol{L}_i = \boldsymbol{L}_o)$.

of values of L_i and L_o. In some cases the savings are significant; for instance, if $N = 4096$ and $L_i = L_o = 164$, the savings in the number of

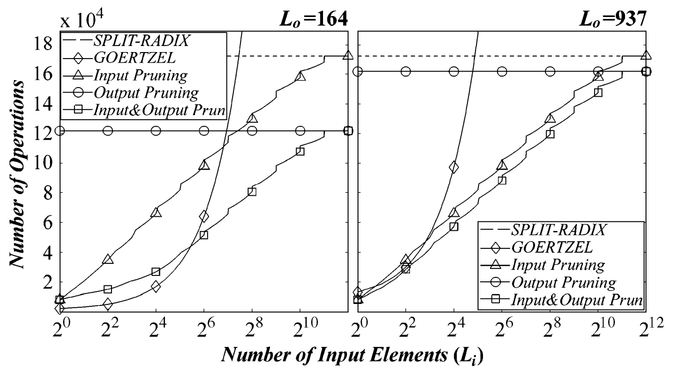

Fig. 5. Comparison of the $\mathbf{FFT}_{\mathrm{DIF-DIT-TD}}$ in different pruning cases and the Goertzel algorithm; $\boldsymbol{N} = \mathbf{4096}$.

arithmetic operations required by the $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ in comparison with a complete split-radix FFT are: 29.50% with the input pruning, 29.25% with the output pruning and 57.94% with the input and output pruning; while with the Goertzel algorithm the saving is only 5.44%.

VI. Conclusion

We have proposed a flexible pruning method that can be used on any composite length FFT and it is capable of carrying out input pruning, output pruning and input and output pruning. No previously reported pruning method can perform these three pruning cases over general composite length FFTs. Any number of consecutive input and/or output elements can be used. It was shown that the proposed pruning method can generate efficient pruned power-of-two and power-of-three length FFTs.

References

[1] H. V. Sorensen and C. S. Burrus, "Efficient computation of the DFT with only a subset of input or output points," *IEEE Trans. Signal Process.*, vol. 41, no. 3, pp. 1184–1199, Mar. 1993.

[2] J. D. Markel, "FFT Pruning," *IEEE Trans. Audio Electroacoust.*, vol. AU-19, no. 4, pp. 305–311, Dec. 1971.

[3] D. P. Skinner, "Pruning the decimation in-time FFT algorithm," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-24, no. 2, pp. 193–194, Apr. 1976.

[4] S. Bouguezel, M. O. Ahmad, and M. N. S. Swamy, "Efficient pruning algorithms for the DFT computation for a subset of output samples," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2003, vol. 4, pp. IV-97–IV-100.

[5] C.-P. Fan and G.-A. Su, "Pruning fast Fourier transform algorithm design using group-based method," *Signal Process.*, vol. 87, pp. 2781–2798, Nov. 2007.

[6] T. V. Sreenivas and P. V. S. Rao, "FFT algorithm for both input and output pruning," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-27, no. 3, pp. 291–292, Jun. 1979.

[7] C. Roche, "A split-radix partial input/output fast fourier transform algorithm," *IEEE Trans. Signal Process.*, vol. 40, no. 5, pp. 1273–1276, May 1992.

[8] H. V. Sorensen, M. T. Heideman, and C. S. Burrus, "On computing the split-radix FFT," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-34, no. 1, pp. 152–156, Feb. 1986.

[9] Y. Susuki, T. Sone, and K. Kido, "A new FFT algorithm of radix 3, 6 and 12," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-34, no. 2, pp. 380–383, Apr. 1986.

[10] G. Goertzel, "An algorithm for the evaluation of finite trigonometric series," *Amer. Math. Monthly*, vol. 65, no. 1, pp. 34–35, Jan. 1958.