Split-Radix Algorithms for Length-p^m DFT's

MARTIN VETTERLI, MEMBER, IEEE, AND PIERRE DUHAMEL, SENIOR MEMBER, IEEE

Abstract—**The split-radix algorithm for the discrete Fourier transform of length-2^m is by now fairly popular. First, we give the reason why the split-radix algorithm is better than any single-radix algorithm on length-2^m DFT's. Then, the split-radix approach is generalized to length-p^m DFT's. It is shown that whenever a radix-p^2 outperforms a radix-p algorithm, then a radix-p/p^2 algorithm will outperform both of them. As an example, a radix-3/9 algorithm is developed for length-3^m DFT's.**

I. Introduction

THE calculation of the discrete Fourier transform (DFT) via a fast algorithm depends on the length of the sequence on which the transform has to be evaluated. When the length N is the product of coprime factors ($N = \Pi N_i$, $(N_i, N_j) = 0$, $i \neq j$), one generally uses Good's mapping [8] to obtain a multidimensional transform that can then be evaluated with the Winograd [20] or the prime factor [10] algorithm (note that the Cooley–Tukey mapping could be used as well, but would be less efficient). When the factors of the length are not coprime (typically when N is a power of a small prime number p), then the Cooley–Tukey mapping [2] must be used and leads to a radix-r algorithm (r being equal to p or one of its powers). Examples are radix-2 or radix-4 algorithms for the length-2^m DFT. In this type of algorithm, a large part of the computational complexity arises from intermediate multiplications, the so-called twiddle factors [12].

In this paper, we will be concerned with the second case (that is, $N = p^m$) and discuss so-called split-radix algorithms. Note that these cannot be obtained via the classical Cooley–Tukey mapping and that this might be the reason why they were discovered only recently. Split-radix algorithms for length-2^m DFT's were introduced simultaneously by several authors in 1984 [4], [11], [18], [16] and lead to the lowest known number of operations (multiplications and additions) for DFT's on these lengths [21]. Recently, a number of papers have explored the split-radix idea, for example, from an implementation point of view [6], [14], for real data [6], [15], [7], or for the computation of convolutions and Hartley transforms [7]. Note, however, that only problems of lengths which are powers of two were considered up to now.

The idea behind the decimation in frequency (DIF) split-radix algorithm for length-2^m DFT's is to start with a radix-2 decomposition on the even indexed outputs of the DFT while using a radix-4 decomposition on the odd indexed part. This approach takes the best of both radix-2 and radix-4 algorithms (in terms of minimizing the number of nontrivial twiddle factors at any one stage while minimizing the number of such stages). In the following, we call the "split-radix" algorithm described in the literature a "radix-2/4" algorithm since it represents only one among many possible splittings of the problem as will be shown.

The more fundamental idea behind the split-radix algorithm is that different decompositions can be used for different parts of an algorithm, an idea that can be more generally applied than just for length-2^m DFT's once it is understood why it works. Actually, the principle is that independent parts of an algorithm should be computed independently and should each use the best possible computational scheme, regardless of what scheme is used for other parts of the algorithm.

The above statement is true, in particular, for computational complexity considerations (which is our point of view in this investigation) but might be relativized when other considerations like regularity come into play as well. In the following, however, we will be mainly concerned with the number of operations only (in particular, the number of multiplications).

The outline of the paper is as follows. Section II investigates the case of length-2^m DFT's. This case is special because small DFT's (like length-2 or 4) are multiplication free and because computational savings are also due to trivial twiddle factors (2nd, 4th, and 8th root of unity). The radix-2/4 algorithm is first motivated as a compromise between a radix-2 and a radix-4 algorithm, and then shown to be best among the various ways to split the computation of a length-2^m DFT. Section III considers the case of DFT's of length-p^m, $p > 2$. It is demonstrated that whenever there exists an algorithm for the length-p^2 DFT that is more efficient than the radix-p algorithm, then the radix-p/p^2 algorithm will outperform both the radix-p and the radix-p^2 algorithm for length-p^m DFT's. Finally, Section IV applies the general method presented earlier to derive a radix-3/9 algorithm for length-3^m DFT's. While the savings obtained over a radix-9 algorithm (a gain of at most 7–11 percent) or other radix-3 schemes [3], [17] are not spectacular, it shows nevertheless that the split-

Manuscript received December 9, 1987; revised May 9, 1988. The work of the first author was supported in part by the American National Science Foundation under Grant CDR-84-21402.

M. Vetterli is with the Department of Electrical Engineering and Center for Telecommunications Research, Columbia University, New York, NY 10027.

P. Duhamel is with CNET/PAB/RPE, Centre National d'Etude des Télécommunications, 38-40 Avenue du Général Leclerc, F-92131 Issy-les-Moulineaux, France.

IEEE Log Number 8824492.

radix idea is more generally applicable than what was thought up to now.

In the following, all DFT's are assumed to have complex inputs. The number of operations is given in terms of real multiplications and additions. Note that we choose to compute the complex multiplication with the algorithm requiring 3 real multiplications and additions [12].

II. Split-Radix Algorithms for Length-2^m DFT's

Consider the DFT of length $N = 2^m$ defined by

$$X_k = \sum_{n=0}^{N-1} x_n \cdot W_N^{nk}, \qquad W_N = e^{-j2\pi/N},$$
$$k = 0 \cdots N-1. \tag{1}$$

We will consider decimation in frequency (DIF) algorithms only (decimation in time algorithms can be derived similarly and lead to the same computational complexity). The idea is to divide the set of output values $\{X_k\}$, $k = 0 \cdots N-1$, into subsets whose union forms the complete set of output values. Each of the subsets is then computed with an adequate algorithm. For example, the radix-2 DIF algorithm performs the following division:

$$\{X_k\}_{\text{radix-2}} = \{X_{2k_1}\} \cup \{X_{2k_1+1}\}$$
$$k_1 = 0 \cdots \frac{N}{2} - 1, \tag{2}$$

whereas the radix-4 DIF algorithm (assuming that m is even) uses the division

$$\{X_k\}_{\text{radix-4}} = \{X_{4k_1}\} \cup \{X_{4k_1+1}\} \cup \{X_{4k_1+2}\}$$
$$\cup \{X_{4k_1+3}\}, \qquad k_1 = 0 \cdots \frac{N}{4} - 1. \tag{3}$$

As is well known, the evaluation of each of the subsets, when chosen properly, is done through a DFT of the size of the subset, plus possibly some auxiliary operations in the form of multiplications by twiddle factors.

Now, the division of the output set into r sets of size N/r each is only one of many possible divisions. Obviously, as long as the following equality holds:

$$\{X_{k_1}\} \cup \{X_{k_2}\} \cup \cdots \cup \{X_{k_l}\} = \{X_k\}, \tag{4}$$

that is, that the subdivision is complete, then we have a valid algorithm for computing the DFT of the input sequence $\{x_n\}$, $n = 0 \cdots N-1$.

In the following, we will explore various subdivisions and explain why the radix-2/4 algorithm is actually a good choice. Then, by elimination, we show that it is actually the best choice among a whole class of DFT algorithms of length-2^m. Finally, it is shown that better algorithms can be developed, but only by using more efficient small DFT algorithms. Since the small DFT algorithms for length 2, 4, 8, and 16 are optimal using the radix-2/4 approach, it turns out that the first improved algorithm is obtained by using a better 32-point DFT [9]. This leads to a radix-2/32 algorithm that outperforms other approaches as will be shown on an example.

In order to derive a best possible algorithm for length-2^m DFT's, we look first at the subset $\{X_{2k_1}\}$ in (2), that is, from (1),

$$X_{2k_1} = \sum_{n=0}^{N-1} x_n \cdot W_N^{2nk_1} = \sum_{n=0}^{N/2-1} (x_n + x_{n+N/2}) \cdot W_{N/2}^{nk_1},$$
$$k_1 = 0 \cdots \frac{N}{2} - 1. \tag{5}$$

Thus, the set of outputs $\{X_{2k_1}\}$ is exactly the output of a length $N/2$ DFT, without any multiplicative cost. If we are trying to develop the best possible algorithm for the length-2^m DFT (in terms of number of operations), then we *must* use that algorithm for the half-size DFT as well. Therefore, it seems clear that the division operated in (5), and which can be stated as

$$\{\cup_k \{X_k\}\} = \{X_{2k_1}\} \cup \{\cup_j \{X_j\}\},$$
$$k_1 = 0 \cdots \frac{N}{2} - 1, \qquad j \neq 2k_1, \tag{6}$$

is a good strategy, so far, since no multiplicative cost has occurred yet; and it is even the optimum strategy (computationally speaking), since whenever a shorter DFT can be found at no cost inside the initial one, then the optimum algorithm can be applied to it as well. Actually, this division is exactly what is done in optimal algorithms for length-2^m DFT's [5], [9].

But now, there is no *a priori* reason to consider all odd indexed terms at once, as it is done in a radix-2 approach. One possibility is to use a radix-2^l decomposition for the odd indexed terms, that is,

$$\{X_{2k_1+1}\} = \left\{ \bigcup_{j=0}^{2^{(l-1)}-1} \{X_{2^l k_1 + 2j+1}\} \right\},$$
$$k_1 = 0, 1, \cdots, \frac{2^m}{2^l} - 1. \tag{7}$$

If the subsets are chosen properly, such a decomposition will require

- $N/2^l$ DFT's of size 2^l where only the odd indexed outputs are needed ("input" stage)
- approximately $N/2 - 2^l$ twiddle factors (some of them being trivial)
- 2^{l-1} DFT's of size $N/2^l$ ("output" stage).

These numbers can be easily checked out on an example. Let us consider the computation of the odd indexed outputs in the case of a radix-4 decomposition (as it is done in the radix-2/4 algorithm):

$$\{X_{2k_1+1}\} = \{X_{4k_1+1}\} \cup \{X_{4k_1+3}\},$$
$$k_1 = 0 \cdots \frac{N}{4} - 1. \tag{8}$$

Then

$$X_{4k_1+2j+1} = \sum_{n=0}^{N-1} x_n \cdot W_N^{(4k_1+2j+1)n}$$

$$= \sum_{n_1=0}^{N/4-1} \sum_{n_2=0}^{3} x_{n_1+n_2N/4} \cdot W_N^{(4k_1+2j+1)(n_1+n_2N/4)}$$

$$= \sum_{n_1=0}^{N/4-1} W_{N/4}^{k_1 n_1} \cdot W_N^{n_1(2j+1)} \cdot \sum_{n_2=0}^{3} x_{n_1+n_2N/4} \cdot W_4^{n_2(2j+1)},$$

$$k_1 = 0 \cdots \frac{N}{4} - 1, \quad j = 0, 1. \tag{9}$$

The last sum corresponds to a length-4 DFT where only the odd indexed terms (1 and 3) are evaluated. There are $N/4$ such reduced DFT's. The next term corresponds to twiddle factors, and the left sum is a length-$N/4$ DFT (there are two of them).

Now, the crucial compromise for a good choice of l in a radix-$2/2^l$ algorithm is as follows. As the radix ($r = 2^l$) increases, the size of the final DFT's decreases, thus reducing the complexity of the output stage. However, the number of operations for the input DFT's is raised, thus increasing the complexity of the input stage. Note that the number of twiddle factors remains approximately constant. Therefore, an optimal tradeoff has to be found, and this can be done by trial and error.

In a first step, let us limit our search to the case where the length-2^l odd DFT is computed through any radix-2^k DFT, including radix-2/4. The number of multiplications required for computing a DFT of any length can be obtained easily by iterating the above arithmetic complexities. In that case, it turns out by inspection that the radix-2/4 algorithm is the best among the radix-$2/2^l$ algorithms. The most likely reason for that fact is that the length-4 DFT is the largest multiplication free DFT, and thus, the input stage requires no multiplications at all. Fig. 1 shows schematically a comparison between the radix-2, radix-4, and radix-2/4 algorithms for a length-16 DFT. This shows how the radix-2/4 is actually a compromise between the two other algorithms.

In a second step, let us consider a radix-2/32 algorithm, and compare the use of the classical algorithm for the length-32 DFT with the use of an optimum algorithm for the length-32 DFT as basic building blocks for larger DFT's. As an example, we derive a radix-2/32 algorithm for the length-1024 DFT. Obviously, the even indexed outputs are obtained with 512 input additions (corresponding to $x_n + x_{n+N/2}$) and a length-512 DFT. The odd indexed outputs are now derived using a radix-32 approach. This implies 32 DFT's of length-32, but where only odd indexed outputs are required. Such a length-32 DFT with odd outputs only takes 48 multiplies ([12] gives this result using a Rader–Brenner algorithm [13], but the same number can be obtained with a split-radix algorithm

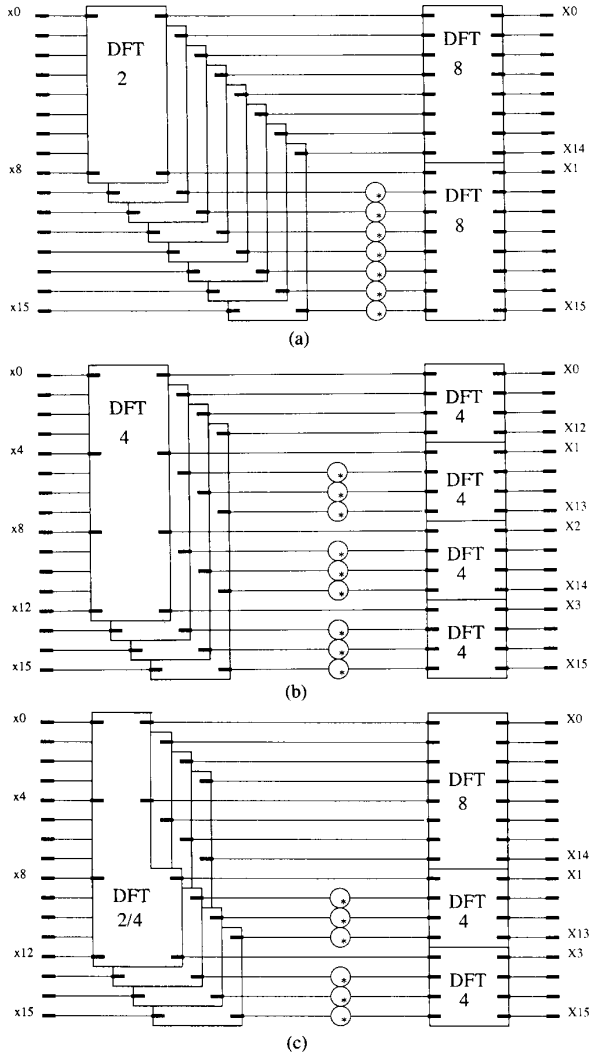


Fig. 1. Schematic comparison of various FFT algorithms for the length-16 DFT. (a) Radix-2 algorithm. (b) Radix-4 algorithm. (c) Radix-2/4 algorithm.

[6] with fewer additions). Then, there are 16 times 31 twiddle factors. Finally, there are 16 output DFT's of length-32. Using the lowest known operation counts (for practical algorithms) for length-32 and 512 DFT's as given, for example, in [18], we arrive at a total of 7188 multiplications. This is actually an increase by 16 over the 7172 multiplications that are used by the radix-2/4 algorithm.

The only way to improve this algorithm, at least as far as multiplications are concerned, is to use a better length-32 DFT algorithm like the one proposed by Heideman and Burrus [9] which uses 64 instead of 68 multiplications (at the cost of 116 additions). In that case, the length-1024 DFT requires 6988 multiplications (using a radix-2/32 approach in all parts of the algorithm). This is better than a radix-2/4 algorithm (using also the improved 32-point DFT) which takes 7088 multiplications. Of course, these

algorithms are too costly in terms of additions in order to be practical, but this is not our point here. Note that the optimal minimum multiplication algorithm would only require 3872 multiplications [9], but at the cost of an unknown yet very large number of additions. The results of this section can be summarized in the following two remarks [19].

Remark 1: The algorithm requiring the lowest number of multiplications for computing the length-2^m DFT will be a radix-$2/2^k$ algorithm, k remaining to be found.

For example, the optimal algorithm [5], [9] (in terms of multiplications) for the length-2^m DFT is a radix-$2/2^{m-1}$ algorithm. The computation of the odd part of size 2^{m-1} leads, however, to an unpractical number of additions, and thus, smaller radixes are chosen.

Remark 2: As soon as there is a length-2^l algorithm that has a smaller number of multiplications than the radix-$2/4$ algorithm for the length-2^l DFT, then the radix-$2/2^l$ will have less multiplications than the radix-$2/4$ algorithm.

It seems, therefore, that the radix-$2/4$ algorithm is the best, as far as the total number of operations is concerned, among a rather general class of practical algorithms that map length-2^m DFT's into smaller DFT's and twiddle factors, due to its low number of additions and its regular structure. Additional improvements seem only possible by using more efficient small DFT's (like the one in our radix-$2/32$ example). Nevertheless, as a side result, it was shown that many efficient mappings are possible besides the classical Cooley–Tukey mapping. Now, we will use our understanding of the radix-$2/4$ algorithm in order to develop split-radix algorithms for length-p^m DFT's, $p > 2$.

III. Split-Radix Algorithms for Length-p^m DFT's, $p > 2$

The situation in the case $p > 2$ is somewhat simpler because there are no trivial twiddle factors (except the zeroth power of the roots of unity, of course). We will concentrate our attention on radix-p/p^2 algorithms since, on the one hand, we know from the case $p = 2$ that they are interesting, and, on the other hand, there are not many improved practical algorithms available for p^3, $p > 2$. Thus, we consider the following division of the outputs of a length-p^m DFT:

$$\{\cup_k\{X_k\}\} = \{X_{pk_1}\} \cup \{\cup_j\{X_{p^2k_2+j}\}\}$$

$$k_1 = 0 \cdots \frac{N}{p} - 1,$$

$$k_2 = 0 \cdots \frac{N}{p^2} - 1,$$

$$j = 0 \cdots p^2 - 1$$

$$\text{and} \quad (j) \bmod p \neq 0. \qquad (10)$$

That is, the output terms with indexes multiple of p are considered in a radix-p fashion, while the others are considered in a radix-p^2 fashion.

In the following, we are going to show precisely that whenever the radix-p^2 algorithm is more efficient than the radix-p algorithm (that means there is a better algorithm for the length-p^2 DFT than the radix-p solution), then the radix-p/p^2 algorithm is better than both of them.

Consider first the multiplicative complexity of a radix-p algorithm for computing length-p^m DFT's. The first stage uses p^{m-1} DFT's of length-p. The output stage uses p DFT's of length-p^{m-1}. In between are p^m twiddle factors, but $p^{m-1} + p - 1$ of them are trivial. Denoting by N_p and N_t the number of multiplications of a length-p DFT and of a twiddle factor, respectively, and by $O_p(m)$ the number of multiplications of the length-p^m DFT (the subscript p denoting the radix-p), we get the following recursion:

$$O_p(m) = p \cdot O_p(m-1) + p^{m-1} \cdot N_p + \left(p^{m-1}(p-1) - p + 1\right) \cdot N_t. \qquad (11)$$

The initial conditions of this recursion are

$$O_p(0) = 0$$

$$O_p(1) = N_p$$

$$O_p(2) = 2pN_p + N_t(p-1)^2. \qquad (12)$$

A general solution of (11) is of the form

$$O_p(m) = a_1 \cdot m \cdot p^m + a_2 \cdot p^m + a_3. \qquad (13)$$

Solving (13) to meet the initial conditions given by (12) leads to

$$a_1 = \frac{N_p + N_t(p-1)}{p}$$

$$a_2 = -N_t$$

$$a_3 = N_t. \qquad (14)$$

Obviously, we will be most interested in the leading term a_1 in (14). Note that the above result holds for $p = 2$ if one disregards the simplifications due to 2nd, 4th, and 8th roots of unity.

Assume now that m is even, and that we compute the DFT in a radix-p^2 fashion. The number of multiplications for the length-p^2 DFT, denoted by N_{pp}, is equal to

$$N_{pp} = 2pN_p + N_t(p-1)^2 - \alpha, \qquad (15)$$

that is, it is equivalent to the radix-p version of the length-p^2 DFT, minus a gain α. We can now use N_{pp} and p^2 in place of N_p and p in (14) in order to get a complexity of the radix-p^2 algorithm. The new leading term, a_1', equals

$$a_1' = \frac{2pN_p + N_t(p-1)^2 + N_t(p^2-1) - \alpha}{p^2}. \qquad (16)$$

The complexity of the radix-p^2 algorithm, denoted by $O_{p^2}(m)$, has to be compared to the complexity of the radix-p algorithm at $2m$, that is, $O_p(2m)$. The leading terms

are

$$O_p(2m) = a_1 \cdot 2m \cdot p^{2m} = 2a_1 \cdot m \cdot (p^2)^m$$

$$O_{p^2}(m) = a_1' \cdot m \cdot (p^2)^m \tag{17}$$

from where it is clear that, for m large enough,

$$O_{p^2}(m) < O_p(2m) \Leftrightarrow a_1' < 2a_1. \tag{18}$$

Now, the ratio a_1'/a_1 can be rewritten as

$$\frac{a_1'}{a_1} = 2 - \frac{\alpha}{p(N_t(p-1) + N_p)}. \tag{19}$$

Clearly, the radix-p^2 algorithm outperforms the radix-p algorithm as soon as α is greater than zero [see (15)], which is an expected result, of course.

We now turn our attention to the radix-p/p^2 algorithm for length-p^m DFT's. The splitting is done according to (10) and leads to the following expression:

$$X_{pk_1} = \sum_{n_1=0}^{(N/p)-1} W_{N/p}^{n_1 k_1} \cdot \sum_{n_2=0}^{p-1} x_{n_1 + n_2 \cdot N/p},$$

$$k_1 = 0 \cdots (N/p) - 1 \tag{20}$$

$$X_{p^2k_3+k_2} = \sum_{n_1=0}^{(N/p^2)-1} W_{N/p^2}^{n_1 k_3} \cdot W_N^{n_1 k_2} \cdot \sum_{n_2=0}^{p^2-1} W_{p^2}^{n_2 k_2} \cdot x_{n_1 + n_2 N/p^2}$$

$$k_2 = 0 \cdots p^2 - 1$$

$$\text{and} \quad (k_2) \bmod p \neq 0,$$

$$k_3 = 0 \cdots (N/p^2) - 1. \tag{21}$$

The above equations can be verified by deriving a radix-p algorithm for X_{pk_1} in (20) and a radix-p^2 algorithm for $X_{p^2k_3+k_2}$ in (21).

Considering the computational complexity of the above equations, we note that (20) corresponds to a DFT of length-p^{m-1}. The right sum in (21) corresponds to N/p^2 DFT's of size-p^2, but where outputs with indexes multiple of p are not required. By extension, we call this an "odd-DFT" of size-p^2. The left sum in (21) amounts to $p^2 - p$ DFT's of size-p^{m-2} (the minus p comes from the excluded values of k_2). Finally, there are $p^2 - p$ groups of twiddle factors, each with $N/p^2 - 1$ nontrivial ones.

Note that the "odd-DFT" of size-p^2 requires the multiplicative complexity of a size-p^2 DFT minus the one of a size-p DFT, since the outputs with indexes multiple of p are not required. The complexities can be subtracted because the two computations are independent. Also, the gain α of the size-p^2 DFT [see (15)] carries over completely to the "odd-DFT" of size-p^2 as well.

The above numbers lead to the following recursion for the multiplicative complexity $O_{p/p^2}(m)$ of the radix-p/p^2 algorithm:

$$O_{p/p^2}(m) = O_{p/p^2}(m-1) + p(p-1) \cdot O_{p/p^2}(m-2) + p^{m-2}(N_{pp} - N_p + p(p-1)N_t) - p(p-1)N_t. \tag{22}$$

The general solution of (22) is of the form (see the Appendix)

$$O_{p/p^2}(m) = a_1'' \cdot m \cdot p^m + a_2'' \cdot p^m + a_3'' \cdot (1-p)^m + a_4''. \tag{23}$$

The terms in p^m and $(1-p)^m$ correspond to the homogeneous solution of the second-order recursive difference equation (p and $1-p$ being the eigenvalues of the transition matrix) while $a_1'' \cdot m \cdot p^m$ and a_4'' are related to the inhomogeneous part. The initial conditions are

$$O_{p/p^2}(0) = 0$$

$$O_{p/p^2}(1) = N_p$$

$$O_{p/p^2}(2) = N_{pp} \tag{24}$$

and $O_{p/p^2}(3)$ is obtained from (22) using (24). This leads to four equations for the unknowns a_1'', a_2'', a_3'', and a_4''. The solution can be found in the Appendix, from where we take the leading term which equals

$$a_1'' = \frac{N_t \cdot p^2 - N_t \cdot p + N_{pp} - N_p}{p(2p-1)}. \tag{25}$$

Similarly to (17) and (18), we can state that, for large enough m,

$$O_{p/p^2}(2m) < O_{p^2}(m) \Leftrightarrow 2a_1'' < a_1'. \tag{26}$$

Using N_{pp} from (15), a_1' from (16), and a_1'' from (25), one can verify that

$$\frac{2a_1''}{a_1'} = \frac{f(p, N_t, N_p, \alpha)}{f(p, N_t, N_p, \alpha) + \alpha} \tag{27}$$

where $f(p, N_t, N_p, \alpha)$ is given in the Appendix. Therefore, whenever α is greater than zero, then the ratio in (27) is smaller than one, which means that the radix-p/p^2 approach performs better than the radix-p^2 algorithm.

We recall that α is the gain or improvement of a length-p^2 DFT algorithm over an equivalent radix-p version. Therefore, with (19) and (27), we have proven that for large m and $\alpha > 0$, the following relation holds:

$$O_{p/p^2}(2m) < O_{p^2}(m) < O_p(2m) \tag{28}$$

which is the central result of this section. An interpretation of this result seem appropriate at this point. The radix-p/p^2 algorithm takes the best of both the radix-p and the radix-p^2 algorithm. First, from the radix-p algorithm, it takes the multiplication free mapping into a size-N/p problem for the outputs with indexes multiple of p. Then, from the radix-p^2 algorithm, it takes the more efficient computation of the "odd" part of the size-p^2 DFT. Further improvements seem only possible by using more ef-

ficient DFT's of size-p^l, $l > 2$. Obviously, these results are parallel to the ones derived in the previous section for length-2^m DFT's.

IV. An Example

In this section, we derive a radix-3/9 algorithm that is more efficient than the straight radix-3 or radix-9 algorithms for length-3^m DFT's.

A length-3 DFT takes 4 real multiplies while an improved length-9 DFT uses 20 real multiplies [1, Appendix B]. Since a radix-3 version of the length-9 DFT would use 6 DFT's of length-3 and 4 twiddle factors (3 multiplications each), that is, a total of 36 multiplications, there is an improvement α equal to 16. Now, the "odd-DFT" of length-9, that is, a DFT of length-9 where x_0, x_3, and x_6 are not used, requires 16 multiplies as can be seen by stripping the length-9 DFT algorithm in [1, Appendix B]. Using these various complexities, we obtain [from (14) and (a5)]

$$O_3(m) = \frac{10}{3} \cdot m \cdot 3^m - 3 \cdot 3^m + 3 \tag{29a}$$

$$O_9(m) = \frac{44}{9} \cdot m \cdot 9^m - 3 \cdot 9^m + 3 \tag{29b}$$

$$O_{3/9}(m) = \frac{34}{15} \cdot m \cdot 3^m - \frac{59}{25} \cdot 3^m - \frac{16}{25} \cdot (-2)^m + 3. \tag{29c}$$

Table I lists the number of multiplications for length-3^m DFT's. For the radix-9 approach when m is odd, a mixed-radix technique (1 step in radix-3, the rest in radix-9) was used. The asymptotic gain (27) of the radix-3/9 over the radix-9 algorithm is

$$\frac{2a_1''}{a_1'} = \frac{68/15}{44/9} = 0.92727 \cdots . \tag{30}$$

Both the radix-3 and the radix-9 algorithms can be improved by using so-called small DFT's with scaled output [12]. In that case, the outputs of the small DFT's have an attached scale factor that can be included into the multiplications by the "twiddle factors," or otherwise, the resulting total DFT will have a scale factor as well, obtained by multiplying the scale factors of the cascaded stages.

The length-3 DFT with scaled output takes 2 multiplications only, leading to 24 multiplications for a radix-3 version of the length-9 DFT. The improved length-9 DFT with scaled output uses 16 multiplies, and thus the gain α is equal to 8. Finally, the "odd-DFT" of length-9 uses 14 multiplies. Using these numbers leads to the following multiplicative complexities [from (14) and (a5)]:

$$O_3(m) = \frac{8}{3} \cdot m \cdot 3^m - 3 \cdot 3^m + 3 \tag{31a}$$

$$O_9(m) = \frac{40}{9} \cdot m \cdot 9^m - 3 \cdot 9^m + 3 \tag{31b}$$

TABLE I
Number of Multiplications for the Length-3^m DFT

m	3^m	$O_3(m)$	$O_9(m)$	$O_{3/9}(m)$
1	3	4	4	4
2	9	36	20	20
3	27	192	144	128
4	81	840	552	536
5	243	3324	2460	2204
6	729	12396	8508	8156
7	2187	44472	32808	29624

$$O_{3/9}(m) = \frac{32}{15} \cdot m \cdot 3^m - \frac{67}{25} \cdot 3^m - \frac{8}{25} \cdot (-2)^m + 3. \tag{31c}$$

Table II lists the multiplicative complexity for the length-3^m DFT's with scaled output. The asymptotic gain of the radix-3/9 over the radix-9 algorithm is

$$\frac{2a_1''}{a_1'} = \frac{64/15}{40/9} = 0.96 \tag{32}$$

which is smaller than in (30). This was to be expected since the gain α was smaller than in the nonscaled version. The last column in Table II lists the theoretical minimum number of multiplications achieved by an optimal algorithm following [9]. Obviously, both the radix-3/9 and the radix-9 algorithms are suboptimal, even for such a small length as $3^3 = 27$. Note that the radix-2/4 algorithm is optimal up to $2^4 = 16$.

Table III gives the number of additions for the various algorithms for length-3^m DFT's. Note that the improvements are of the same order as for multiplications, and that the radix-3/9 algorithm seems to achieve the minimum number of operations (additions plus multiplications) for length-3^m DFT's, just as does the radix-2/4 for length-2^m DFT's.

As far as length-3^m DFT's are concerned, and since they intend to be practical algorithms, a number of remarks are of interest. Several algorithms achieving low arithmetic complexity have been proposed. A first one, by Dubois and Venetsanopoulos [3], makes use of the $(1, \mu)$ plane [$\mu^3 = 1$] in order to reduce the number of multiplications inside the algorithm, at the cost of one real multiplication per point at each "translation" between the $(1, j)$ and the $(1, \mu)$ plane. This is the same algorithm that was used later on by Martens [11], together with the 3 multiplication/addition complex multiplication scheme. Another approach was proposed by Suzuki *et al.* [17], who have shown that when a 4 multiplication complex product was used, the complex product inside the length-3 DFT could be merged with the computation of the twiddle factors and was thus free, and this without using the $(1, \mu)$ plane.

TABLE II
NUMBER OF MULTIPLICATIONS FOR THE LENGTH-3^m DFT WITH SCALED OUTPUT

m	3^m	$O_3(m)$	$O_9(m)$	$O_{3/9}(m)$	optim. algo.
1	3	2	2	2	2
2	9	24	16	16	16
3	27	138	114	106	74
4	81	624	480	472	272
5	243	2514	2082	1954	898
6	729	9480	7536	7360	2816
7	2187	34266	28434	26842	8618

TABLE III
NUMBER OF ADDITIONS FOR THE LENGTH-3^m DFT WITH SCALED OUTPUT

m	3^m	$O_3(m)$	$O_9(m)$	$O_{3/9}(m)$
1	3	16	16	16
2	9	108	90	90
3	27	516	462	408
4	81	2136	1812	1650
5	243	8184	7212	6240
6	729	29892	25518	22602
7	2187	105708	95586	79464

When compared to the algorithms making use of the $(1, \mu)$ plane [3], [11], the radix-3/9 algorithm has both a lower number of multiplications and additions (if the conversion to the ordinary $(1, j)$ is counted), while it achieves less multiplications than the scheme in [17].

The improvements are not substantial, but they open new possibilities for efficient algorithms of length-$2^k \cdot 3^l$. Note that the density of DFT lengths covered by length $2^k \cdot 3^l$ FFT's is much higher than that of single radix algorithms, thus making such FFT's quite useful.

Finally, the question remains open if there is a radix-3/9 algorithm in the $(1, \mu)$ plane that would improve current algorithms. At this point, there is no known radix-9 algorithm that has a gain factor α greater than zero, and thus, there would be no advantage in going to a radix-3/9 scheme in the $(1, \mu)$ plane.

V. CONCLUSION

First, the split-radix algorithm (or radix-2/4 as it is called here for clarity) for length-2^m DFT's has been reviewed. It was shown that the radix-2/4 is the best algorithm among a general class of possible split-radix algorithms, and that improvements can only be achieved by going to more efficient small DFT algorithms (like an improved length-32 DFT which leads to a more efficient radix-2/32 algorithm).

Then, the split-radix idea was generalized to length-p^m DFT's, $p > 2$. It was proven that whenever there exists an improved length-p^2 DFT algorithm, then the radix-p/p^2 will outperform both the radix-p and the radix-p^2 algorithm.

As an example, a radix-3/9 algorithm was developed which achieves better performance than the radix-3 or the radix-9 algorithm. While not achieving the minimum number of multiplications, this algorithm leads to a good compromise in terms of the total number of operations.

In short, it was shown that the split-radix idea gives a rather general method to devise efficient algorithms for length-p^m DFT's.

APPENDIX
SOLUTION OF THE RECURSIVE DIFFERENCE EQUATION DEFINING THE MULTIPLICATIVE COMPLEXITY OF THE RADIX-p/p^2 ALGORITHM

The recursive difference equation is given in (22) and the initial conditions are given by (24), and

$$\begin{aligned} O_{p/p^2}(3) &= N_{pp} + p(p-1)N_p \\ &\quad + p(N_{pp} - N_p + p(p-1)N_t) \\ &\quad - p(p-1)N_t \end{aligned} \tag{a1}$$

which is found with (22) and (24). The homogeneous part of (22) corresponds to the following transition matrix:

$$\begin{pmatrix} O(m) \\ O(m-1) \end{pmatrix} = \begin{pmatrix} 1 & p(p-1) \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} O(m-1) \\ O(m-2) \end{pmatrix}. \tag{a2}$$

This transition matrix has the eigenvalues $\lambda_1 = p$ and $\lambda_2 = 1 - p$. The inhomogeneous part of (22) will give rise to a leading term in $m \cdot p^m$ and to a constant. This leads to a general solution of the form

$$\begin{aligned} O_{p/p^2}(m) &= a_1'' \cdot m \cdot p^m + a_2'' \\ &\quad \cdot p^m + a_3'' \cdot (1-p)^m + a_4''. \end{aligned} \tag{a3}$$

In order to meet the initial conditions of (24) and (a1), the factors a_i'' of (a3) have to solve the following set of equations:

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ p & p & (1-p) & 1 \\ 2p^2 & p^2 & (1-p)^2 & 1 \\ 3p^2 & p^3 & (1-p)^3 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_1'' \\ a_2'' \\ a_3'' \\ a_4'' \end{pmatrix} = \begin{pmatrix} 0 \\ N_p \\ N_{pp} \\ O_{p/p^2}(3) \end{pmatrix}. \tag{a4}$$

The solution of (a4) leads to

$$a_1'' = \frac{N_t p^2 - N_t p + N_{pp} - N_p}{p(2p-1)}$$

$$a_2'' = -\frac{3N_t p^2 - 2N_t p - 2N_p p + N_{pp}}{(2p-1)^2}$$

$$a_3'' = -\frac{N_t p^2 - 2N_t p + 2N_p p + N_t - N_{pp}}{(2p-1)^2}$$

$$a_4'' = N_t. \tag{a5}$$

Now, we use the number of multiplications for the length-p^2 DFT, N_{pp}, as in (15), and we calculate the ratio of the leading terms of the radix-p/p^2 (a5) and the radix-p^2 algorithm [from (16)]. This ratio can be rewritten as

$$\frac{[a_1]_{\text{radix}-p/p^2}}{[a_1]_{\text{radix}-p^2}} = \frac{1}{2} \cdot \frac{f(p, N_t, N_p, \alpha)}{f(p, N_t, N_p, \alpha) + \alpha} \tag{a6}$$

with

$$f(p, N_t, N_p, \alpha) = 4N_t p^3 + (4N_p - 6N_t)p^2 + 2p(N_t - N_p - \alpha). \tag{a7}$$

Thus, as soon as α is greater than zero, then (a6) is smaller than $1/2$.

Acknowledgment

The authors would like to thank Dr. M. T. Heideman for providing information on the length-32 minimum multiply DFT algorithm that he developed; Prof. M. Kunt for suggesting a general look at the DFT problem which led to the results presented in this paper; as well as the anonymous reviewers for their helpful comments.

References

[1] R. E. Blahut, *Fast Algorithms for Digital Signal Processing*. Reading, MA: Addison-Wesley, 1984.

[2] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297-301, Apr. 1965.

[3] E. Dubois and A. N. Venetsanopoulos, "A new algorithm for the radix-3 FFT," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-26, pp. 222-225, June 1978.

[4] P. Duhamel and H. Hollmann, "Split-radix FFT algorithm," *Electron. Lett.*, vol. 20, no. 1, pp. 14-16, Jan. 5, 1984.

[5] P. Duhamel and H. Hollmann, "Existence of a 2^n FFT algorithm with a number of multiplications lower than $2^{(n+1)}$," *Electron. Lett.*, vol. 20, no. 17, pp. 690-692, Aug. 1984.

[6] P. Duhamel, "Implementation of "split-radix" FFT algorithms for complex, real and real-symmetric data," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 285-295, Apr. 1986.

[7] P. Duhamel and M. Vetterli, "Improved Fourier and Hartley transform algorithms—Application to cyclic convolution of real data," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, pp. 818-824, June 1987.

[8] I. J. Good, "The interaction algorithm and practical Fourier analysis," *J. Roy Stat. Soc.*, vol. B-20, pp. 361-372, 1958; vol. B-22, pp. 372-375, 1960.

[9] M. T. Heideman and C. S. Burrus, "On the number of multiplications necessary to compute a length-2^n DFT," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 91-95, Feb. 1986.

[10] D. P. Kolba and T. W. Parks, "A prime factor algorithm using high-speed convolution," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-25, pp. 281-294, Aug. 1977.

[11] J. B. Martens, "Recursive cyclotomic factorization—A new algorithm for calculating the discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 750-761, Aug. 1984.

[12] H. J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*. Berlin: Springer, 1982.

[13] C. M. Rader and N. M. Brenner, "A new principle for fast Fourier transformation," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-24, pp. 264-265, June 1976.

[14] H. V. Sorensen, M. T. Heideman, and C. S. Burrus, "On computing the split-radix FFT," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 152-156, Feb. 1986.

[15] H. V. Sorensen, D. L. Jones, M. T. Heideman, and C. S. Burrus, "Real-valued fast Fourier transform algorithms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, pp. 849-863, June 1987.

[16] R. Stasinski, "Easy generation of small-N discrete Fourier transform algorithms," *IEE Proc.*, vol. 133, pt. G, no. 3, pp. 133-139, June 1986.

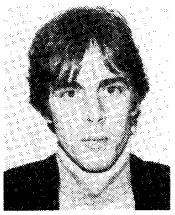
[17] Y. Suzuki, T. Sone, and K. Kido, "A new FFT algorithm of radix 3, 6, and 12," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 380-383, Apr. 1986.

[18] M. Vetterli and H. J. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations," *Signal Processing*, vol. 6, no. 4, pp. 267-278, Aug. 1984.

[19] M. Vetterli and P. Duhamel, "Split-radix algorithms for length-p^m DFT's," in *Proc. IEEE Int. Conf. ASSP*, Apr. 1988, pp. 1415-1418.

[20] S. Winograd, "On computing the discrete Fourier transform," *Proc. Nat. Acad. Sci. U.S.A.*, vol. 73, pp. 1005-1006, Apr. 1976.

[21] R. Yavne, "An economical method for calculating the discrete Fourier transform," in *AFIPS Proc.*, vol. 33, Fall Joint Comput. Conf., Washington, 1968, pp. 115-125.



Martin Vetterli (S'86-M'86) was born in Switzerland in 1957. He received the Dipl. El.-Ing. degree from the Eidgenössische Technische Hochschule Zürich, Switzerland, in 1981, the Master of Science degree from Stanford University, Stanford, CA, in 1982, and the Doctorat ès Science degree from the Ecole Polytechnique Fédérale de Lausanne, Switzerland, in 1986.

In 1982 he was a Research Assistant with the Computer Science Department of Stanford University, and from 1983 to 1986 he was a Researcher at the Ecole Polytechnique. He has worked for Siemens, Switzerland, and AT&T Bell Laboratories in Holmdel, NJ. Since 1986 he has been at Columbia University in New York, first with the Center for Telecommunications Research and now with the Department of Electrical Engineering where he is currently an Assistant Professor. His research interests include multirate signal processing, computational complexity, algorithm design for VLSI, and signal processing for telecommunications.

Dr. Vetterli is a member of the Editorial Board of *Signal Processing* and served as European Liaison for ICASSP-88 in New York. He was recipient of the Best Paper Award of EURASIP in 1984 and of the Research Prize of the Brown Bovery Corporation (Switzerland) in 1986.

Pierre Duhamel (M'87-SM'88) was born in France in 1953. He received the Ingenieur degree in electrical engineering from the National Institute for Applied Sciences, Rennes, France, in 1975, the Dr.Ing. degree in 1978, and the Doctorat es Sciences in 1986 from Orsay University, France.

From 1975 to 1980 he was with Thomspon-CSF, Paris, France, where his research interests were in circuit theory and signal processing, including digital filtering and automatic analog fault diagnosis. In 1980 he joined the National Research Center in Telecommunications (CNET), Issy-les-Moulineaux, France, where his activities were first concerned with the design of recursive CCD filters. He is now working on fast convolution algorithms, including number theoretic transforms and fast Fourier transforms.