D2.5

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

Martin Vetterli* and Pierre Duhamel**

*Dept. of EE and CTR
Columbia University, New York, NY 10027, USA
** CNET/PAB/RPE
F-92131 Issy-les-Moulineaux, France

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 one uses generally Good's mapping to obtain a multidimensional transform that can then be evaluated with the Winograd or the prime factor algorithm. 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 [1] 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 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 [3,6,8] and lead to the lowest known number of operations (multiplications and additions) for DFT's on these lengths.

The idea behind the division 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 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.

First, we will investigate the case of length-2^m DFT's. 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 various ways to split a length-2^m DFT. Then, we consider the case of DFT's of length-p^m, $p > 2$. It is shown 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. Then, we apply the general method to derive an efficient radix-3/9 algorithm for length-3^m DFT's. Note that all DFT's are assumed to have complex inputs, that the number of operations is given in terms of real operations and that we use the 3 multiplication/addition algorithm for complex products.

II Split-radix algorithms for length-2^m DFT's

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

$$X_k = \sum_{n=0}^{N-1} x_n \cdot W_N^{nk}, \quad W_N = e^{-j\frac{2\pi}{N}}, \quad k = 0 \ldots N-1. \quad (1)$$

We will consider division in frequency (DIF) algorithms only (division in time algorithms are dual). The idea is to divide the set of output values $\{X_k\}, k = 0 \ldots N-1$, 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:

$$\{X_k\}_{radix-2} = \{X_{2k_1}\} \cup \{X_{2k_1+1}\}, \quad k_1 = 0..N/2-1. \quad (2)$$

The evaluation of each of the subsets, when chosen properly, is done through a DFT of the size of the subset and some auxiliary operations (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:

$$\{X_{k_1}\} \cup \{X_{k_2}\} \cup \ldots \cup \{X_{k_l}\} = \{X_k\} \quad (3)$$

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

$$X_{2k_1} = \sum_{n=0}^{\frac{N}{2}-1} (x_n + x_{n+N/2}) \cdot W_{N/2}^{2nk_1}, \quad k_1 = 0 \ldots \frac{N}{2} - 1 \quad (4)$$

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

CH2561-9/88/0000-1415 $1.00 © 1988 IEEE

trying to develop the best possible algorithm for the length-2^m DFT, then we **must** use that algorithm for the half size DFT as well. Actually, the division in (4) is exactly what is done in optimal algorithms for length-2^m DFT's [4,5]. 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:

$$\{X_{2k_1+1}\} = \{\cup_{j=0}^{2^{(l-1)}-1} x_{2^l k_1+2j+1}\}, l = 1..m,\ k_1 = 0..\frac{2^m}{2^l}-1 \quad (5)$$

Such a decomposition will require, if the subsets are chosen properly:

- $N/2^l$ 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 easely checked out on an example. 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 and complexity of the final DFT's decreases. However, the number of operations for the input DFT's is raised, while the number of twiddle factors remains approximately constant. Therefore, an optimal trade-off has to be found, and this can be done by trial and error. It turns out by inspection that the radix-2/4 algorithm is the best among the radix-$2/2^l$ algorithms, probably due to the fact that the length-4 DFT is the largest multiplication free DFT (the input stage requires no multiplications). Figure 1 depicts schematically a comparison between the radix-2, radix-4 and radix-2/4 algorithms for a length-16 DFT, showing how the radix-2/4 is actually a compromise between the two other algorithms.

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 [5] 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. The results of this section can be summarized in the following two remarks.

Remark 1: The best algorithm for the computation of the length-2^m DFT will be a radix-$2/2^k$ algorithm, k remaining to be found.

For example, the optimal algorithm [4,5] (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 radices are chosen in practice.

Remark 2: As soon as there is a length-2^l algorithm that is better than the radix-2/4 algorithm for the length-2^l DFT, then the radix-$2/2^l$ outperforms the radix-2/4 algorithm.

It seems therefore that the radix-2/4 algorithm is the best 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. As a side result, it was shown that many efficient mappings are possible besides the classical Cooley-Tukey mapping. The understanding of the radix-2/4 algorithm will now be used to develop split-radix algorithms for length-p^m DFT's, $p > 2$.

III Split-radix algorithms for length-p^m DFT's

The situation in the case $p > 2$ is somewhat simpler, because there are no trivial twiddle factors (except the zero-th 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:

$$\{X_k\} = \{X_{pk_1}\} \cup \{\cup_j \{X_{p^2k_2+j}\}\}, \quad k_1 = 0..N/p-1,$$

$$k_2 = 0..N/p^2-1,\ \ j = 0..p^2-1,\ and\ (j)Mod(p) \neq 0 \quad (6)$$

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. We are going to show 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:

$$O_p(m) = p \cdot O_p(m-1) + p^{m-1} \cdot N_p + (p^{m-1}(p-1) - p + 1) \cdot N_t \quad (7)$$

The initial conditions of this recursion are $O_p(0) = 0$, $O_p(1) = N_p$ and $O_p(2) = 2pN_p + N_t(p-1)^2$. A general solution of (7) is of the form:

$$O_p(m) = a_1 \cdot m \cdot p^m + a_2 \cdot p^m + a_3 \quad (8)$$

Solving (8) to meet the initial conditions gives the following leading term a_1:

$$a_1 = (N_p + N_t(p-1))/p \quad (9)$$

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:

$$N_{pp} = 2pN_p + N_t(p-1)^2 - \alpha \quad (10)$$

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 (9) in order to get the complexity of the radix-p^2 algorithm. The new leading term, a_1', equals:

$$a_1' = (2pN_p + N_t(p-1)^2 + N_t(p^2-1) - \alpha)/p^2 \quad (11)$$

The complexity of the radix-p^2 algorithm, denoted by $O_{p^2}(m)$, has to be compared with the complexity of the radix-p algorithm at $2m$, that is, $O_p(2m)$. Comparing the leading terms of the recursion, it is clear that for m large enough:

$$O_{p^2}(m) < O_p(2m) \iff a_1' < 2a_1 \tag{12}$$

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

$$a_1'/a_1 = 2 - \alpha/(p(N_t(p-1)+N_p)) \tag{13}$$

Clearly, the radix-p^2 algorithm outperforms the radix-p algorithm as soon as α is greater than zero (see (10)), 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 (6) and leads to the following expression:

$$X_{pk_1} = \sum_{n_1=0}^{(N/p)-1} W_{N/p}^{n_1 k_1} \cdot \sum_{n_2=0}^{p-1} x_{n_1+n_2 \cdot N/p}, \; k_1 = 0..(N/p)-1 \tag{14a}$$

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

$$k_2 = 0..p^2-1 \; and \; (k_2) Mod(p) \neq 0, \; k_3 = 0..(N/p^2)-1 \tag{14b}$$

The above equations can be verified by deriving a radix-p algorithm for X_{pk_1} in (14a) and a radix-p^2 algorithm for $X_{p^2k_3+k_2}$ in (14b). Considering the computational complexity of the above equations, we note that equation (14a) corresponds to a DFT of length-p^{m-1}. The right sum in (14b) 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 (14b) 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 non-trivial 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 (10)) 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:

$$\begin{aligned} O_{p/p^2}(m) =& O_{p/p^2}(m-1) + p(p-1) \cdot O_{p/p^2}(m-2) \\ &+ p^{m-2}(N_{pp} - N_p + p(p-1)N_t) - p(p-1)N_t \end{aligned} \tag{15a}$$

The general solution of (15a) is of the form [9]:

$$O_{p/p^2}(m) = a_1 \cdot m \cdot p^m + a_2 \cdot p^m + a_3 \cdot (1-p)^m + a_4 \tag{15b}$$

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. With the appropriate initial conditions one is lead to 4 equations for the unknowns a_1, a_2, a_3 and a_4. The solution can be found in [9] from where we take the leading term that we rename a_1'' for clarity:

$$a_1'' = \frac{N_t \cdot p^2 - N_t \cdot p + N_{pp} - N_p}{p(2p-1)} \tag{16}$$

Similarly to (12), we can state that, for large enough m:

$$O_{p/p^2}(2m) < O_{p^2}(m) \iff 2a_1'' < a_1' \tag{17}$$

Now, one can verify that $2a_1''/a_1'$ is of the form:

$$\frac{2a_1''}{a_1'} = \frac{f(p, N_t, N_p, \alpha)}{f(p, N_t, N_p, \alpha) + \alpha} \tag{18}$$

Therefore, whenever α is greater than zero, then the ratio in (18) is smaller than one, which means that the radix-p/p^2 approach performs better than the radix-p^2 algorithm. Recall that α is the gain or improvement of a length-p^2 DFT algorithm over an equivalent radix-p version. Therefore, with (13) and (18), we have proven that for large m and $\alpha > 0$ the following relation holds:

$$O_{p/p^2}(2m) < O_{p^2}(m) < O_p(2m) \tag{19}$$

which is the central result of this section. An interpretation of this result seems 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 efficient 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 discuss 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. Thus, there is an improvement factor 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. Using these various complexities, one can verify that [9]:

$$O_3(m) = \frac{10}{3} \cdot m \cdot 3^m - 3 \cdot 3^m + 3 \tag{20a}$$

$$O_9(m) = \frac{44}{9} \cdot m \cdot 9^m - 3 \cdot 9^m + 3 \tag{20b}$$

$$O_{3/9}(m) = \frac{34}{15} \cdot m \cdot 3^m - \frac{59}{25} \cdot 3^m - \frac{16}{25} \cdot (-2)^m + 3 \tag{20c}$$

Table I lists the number of multiplications for the various length-3^m DFT's. The asymptotic gain (18) of the radix-3/9 over the radix-9 algorithm is equal to 0.92727. Both the radix-3 and the radix-9 algorithms can be improved by using so-called small DFT's with scaled output in which case the asymptotic gain is not as good (this is to be expected since the gain α is smaller than in the non-scaled version).

Several other algorithms achieving low arithmetic complexity have been proposed for the length-3^m DFT [2,6,7]. Two of them [2,6] makes use of the $(1, \mu)$ plane $[\mu^3 = 1]$ in order to reduce the number of multiplications. When compared with the algorithms making use of the $(1, \mu)$ plane [2,6], the radix-3/9 algorithm has both a lower number of multiplications and additions (if the conversion from and to the ordinary $(1, j)$ plane is counted), while it achieves less multiplications than the scheme in [7]. The improvements are not substantial, but they open new possibilities for efficient algorithms of length-$2^k \cdot 3^l$ DFT's.

V Conclusion

First, the split-radix algorithm 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 (leading to a better radix-2/32 algorithm for ex.). 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. 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.

Acknowledgements: The authors thank Dr. M.T Heideman and Prof. M.Kunt for helpful discussions. The first author acknowledges support from NSF under grant CDR-84-21402.

References

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

[2] E.Dubois and A.N.Venetsanopoulos, "A new algorithm for the radix-3 FFT," IEEE Trans. on ASSP, Vol.26, Jun.1978, pp.222-225.

[3] P.Duhamel, and H. Hollmann, "Split-Radix FFT Algorithm," Electronics Letters, Vol.20, No.1, Jan.1984.

[4] P.Duhamel, and H. Hollmann, "Existence of a 2^n FFT Algorithm with a Number of Multiplications lower than $2^{(n+1)}$," Electronics Letters, Vol.20, No.17, Aug.1984.

[5] M.T.Heideman, and C.S.Burrus, "On the number of multiplications necessary to compute a length-2^n DFT," IEEE Trans. on ASSP, Vol.34, No.1, Feb.1986, pp.91-95.

[6] J.B.Martens, "Recursive Cyclotomic Factorization - A New Algorithm for calculating the Discrete Fourier Transform," IEEE Trans. on ASSP, Vol.32, No.4, Aug.1984, pp.750-761.

[7] Y.Suzuki, T.Sone, and K.Kido, "A new FFT algorithm of radix 3, 6, and 12," IEEE Trans. on ASSP, Vol.34, No.2, Apr.1986, pp.380-383.

[8] 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.

[9] M.Vetterli and P.Duhamel, "Split-radix algorithms for length-p^m DFT's," submitted to IEEE Trans. on ASSP.

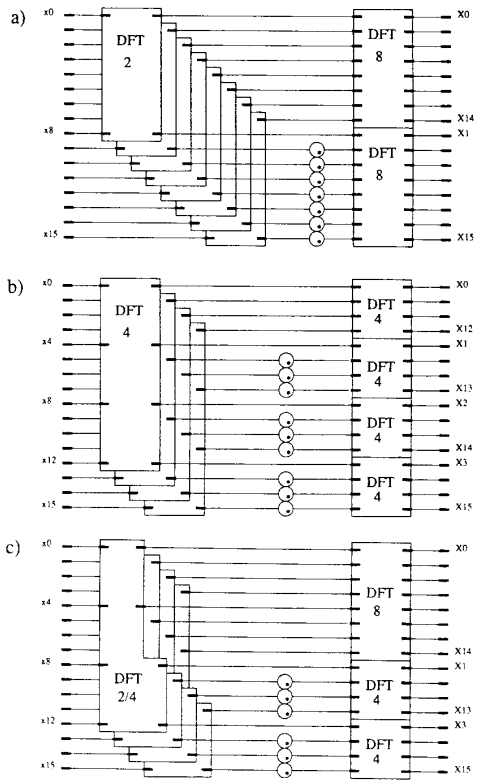

Figure 1: Schematic comparison of radix-2 (a), radix-4 (b) and radix-2/4 (c) algorithms for the length-16 DFT.

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

Table I: Number of multiplications for the length-3^m DFT.