A REVIEW OF THE DISCRETE FOURIER TRANSFORM
Part 2: non-radix algorithms, real transforms and noise

Gerard M Blair

ABSTRACT

Since fast algorithms for the discrete Fourier transform (DFT) were first introduced thirty years ago, they have had a major impact on signal processing and are now a basic part of every electrical engineer's education. However, some of the options, and particularly the recent advances, are not as widely known as they deserve. Over two articles, we review the fast algorithms for the DFT. In the first article, we considered why the DFT works and looked at the various fast algorithms for transforms whose orders are a power of two. In this article, we look at techniques for other orders, at adapting algorithms for purely real data, at the problems of fixed point noise, and at implementation options in existing hardware.

1 DFTs FOR N $\neq 2^n$

In a previous article[1], we looked at fast algorithms for the discrete Fourier transform (DFT) whose order was a power of two. There were two features which made these effective: 1) the basic building blocks were simple to implement, and 2) the large blocks could be efficiently broken into smaller blocks. However, the techniques were only explored for powers of two; let us now look beyond this. restriction. For reference, we are using the following definition for the DFT:

$$S(k) = \sum_{n=0}^{N-1} W_N^{nk}\, s(n) \quad k = 0 \rightarrow N-1 \quad \text{where } W_X^x = e^{-j2\pi x/X} \tag{1}$$

Cyclic convolution algorithms

The first trick is to find algorithms which are efficient for small DFTs of sizes other than powers of two to extend our range of building blocks. The most effective technique is based upon cyclic convolutions: this may be thought of as a sequence of inner products where the coefficients of the multiplying vector are rotated one step each time. The matrix equation below illustrates this:

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & x_2 \\ x_1 & x_2 & x_0 \\ x_2 & x_0 & x_1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \tag{2}$$

This can also be viewed as a polynomial $(x_0 + x_1 u + x_2 u^2).(a_0 + a_1 u + a_2 u^2) \bmod (u^3 - 1)$ which can then be manipulated in terms of the roots of $(u^3 - 1)$. This formulations allows us to derive algorithms for these convolutions with only a relatively small number of multiplications (for instance a 6 point convolution requires only 8 multiplications). This development can progress by decomposing large convolutions (of order N) into nested convolutions (of order N_1)

of convolutions (of order N_2) where N_1 and N_2 are relatively prime factors† of N, and by developing hierarchical techniques for convolutions whose order is a power of a prime number. The reader may pursue this elsewhere, particularly in Nussbaumer[2], but the important point is that these fast algorithms exist.

DFT algorithms based on cyclic convolution

These algorithms are useful to us if we can manipulate the DFT into a cyclic convolution. In the first instance, this is possible for DFTs whose order is a *prime* number. This restriction is imposed because it guarantees that there exists a permutation of the inputs and outputs which produces a DFT matrix with a convolution matrix embedded within it. For instance, the matrix for the 5-pt DFT can be reordered (by exchanging the last two rows and columns) as:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W_5^1 & W_5^2 & W_5^4 & W_5^3 \\ 1 & W_5^2 & W_5^4 & W_5^3 & W_5^1 \\ 1 & W_5^4 & W_5^3 & W_5^1 & W_5^2 \\ 1 & W_5^3 & W_5^1 & W_5^2 & W_5^4 \end{bmatrix} \tag{3}$$

The first output term is simply the sum of the inputs corresponding to the top row of the matrix. The remaining (permuted) output terms are formed by adding the first input term (corresponding to the column of 1s) to the result of the 4-pt cyclic convolution of $(W_5^1, W_5^2, W_5^4, W_5^3)$ with the remaining four (permuted) inputs. As described above, there are polynomial-based algorithms for this convolution and these require only 5 complex multiplications. Thus we have derived a 5-pt DFT which requires only 6 complex multiplications (including the trivial multiplication by 1 of the first output term).

This ability to permute the DFT matrix can be extended to orders which are powers of (odd) prime numbers by decomposing them into convolutions and smaller DFTs of the same (odd) prime order; the technique can then be applied recursively. Thus a DFT whose order is any odd prime or power of an odd prime can be implemented efficiently using cyclic convolutions.

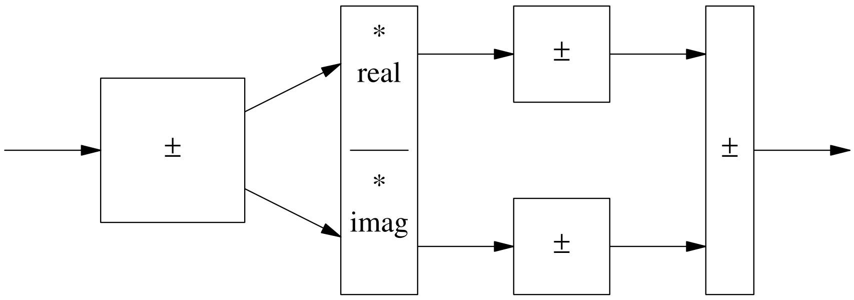


Figure 1: Cyclic-convolution based algorithms

The DFT algorithms based on cyclic convolutions have a specific structure which can be exploited. There are essentially four phases to the calculation as illustrated in figure 1. In phase one, the inputs are combined through various additions and subtractions. In phase two, the new values each undergoes a single complex multiplication by a coefficient which is either purely real or purely imaginary. In phase three, the outputs of the purely real and purely imaginary

† *Two numbers are relatively prime if they share no prime factors: 12 (=2x2x3) and 35 (=5*7) are relatively prime, but 66 (=2*3*11) and 15 (=3*5) are not.*

multiplications are combined *independently* through additions and subtractions, and only in phase four are the two streams combined to form the final outputs.

One consequence of this structure is that since the "complex" multiplications have purely real or purely imaginary coefficients they are of the form: $(a + bj) * c$ or $(a + bj) * dj$, and so require only two real multiplications. Thus the 5-pt DFT described above, with 5 non-trivial complex multiplications, is achieved with only 10 real multiplications - rather than 16 complex multiplications from applying equation 1 (or matrix 3) directly.

A second consequence is that for complex inputs we could implement the first three phases with purely real arithmetic and the real and imaginary data could pass through these phases independently – it is only at the outputs of phase four that both the real and imaginary multiplier streams are combined and at this stage we could combine the results of the real and imaginary data. We will meet other consequences below.

Prime factor algorithms

The second point we want to consider is the effective decomposition of large DFTs. We showed in the previous article ([1], equation 7) that any N-pt DFT with two factors A and C (such that N = AC) can be decomposed into A, C-pt DFTs followed by C, A-pt DFTs with twiddle factors in between. Thus, provided we are careful about the re-ordering, we could decompose the N-pt DFT into layers of N_i-pt DFTs, where the $\{N_i\}$ correspond to the prime factors of N, and perform each of the N_i-pt DFTs using cyclic convolutions. or radix-based algorithms. In fact we can do far better than this by eliminating the twiddle factors as follows.

We start with an N-pt DFT (as in equation 1) where N is non-prime and rewrite the indices k and n in terms of the two factors A and C. In the previous article, we simply replaced k and n with (dA+b) and (aC+c) where a,b = $0 \to A-1$ and c,d = $0 \to C-1$. Now, however, we add the restriction that A and C are relatively prime which allows us to write:

$$k \equiv dA + bC \bmod N, \;\; a, b = 0 \to A - 1 \qquad (4)$$
$$n \equiv aC + cA \bmod N, \;\; c, d = 0 \to C - 1$$

which causes, in effect, a permutation of the indices. Equation 1 then becomes:

$$\begin{aligned} S(dA + bC) &= \sum_{a=0}^{A-1} \sum_{c=0}^{C-1} s(aC + cA) W_N^{((dA+bC)(aC+cA))} \\ &= \sum_{a=0}^{A-1} \sum_{c=0}^{C-1} s(aC + cA) W_N^{(adAC)} W_N^{(cdAA)} W_N^{(abCC)} W_N^{(bcCA)} \\ &= \sum_{c=0}^{C-1} W_C^{(cdA)} \left[\sum_{a=0}^{A-1} W_A^{(abC)} s(aC + cA) \right] \qquad (5) \\ &\text{where } \; b = 0 \to A - 1 \text{ and } d = 0 \to C - 1 \end{aligned}$$

since, for instance, $W_N^{adAC} = e^{-j2\pi adAC/N} = e^{-j2\pi ad} = 1$ as the exponent is an integer multiple of $2\pi j$.

Notice that unlike the decomposition in the previous article, there are now no "twiddle factors". A further permutation allows us to eliminate the A and C terms in the superscripts of $W_C^{(cdA)}$ and $W_A^{(abC)}$; the use of the Chinese Remainder theorem helps with this task[3]. Thus by judicious permutations of the input and outputs (which are only possible if A and C are relatively prime) we arrive at[4]:

$$S(p, q) = \sum_{u=0}^{A-1} W_A^{pu} \sum_{v=0}^{C-1} W_C^{qv} s(u, v) \qquad (6)$$

where $S(p, q)$ are the permuted outputs indexed by p and q and $s(u, v)$ are the permuted inputs indexed by u and v. In this way the N-pt DFT is achieved with A, C-pt DFTs followed by C, A-pt DFTs *without* any intervening twiddle factors. This is illustrated in figure 2 for a 15-pt DFT. Of course this method can be applied recursively and so the N-pt DFT can be evaluated as layers of N_i-pt DFTs where $\{N_i\}$ are *relatively prime* factors of N.

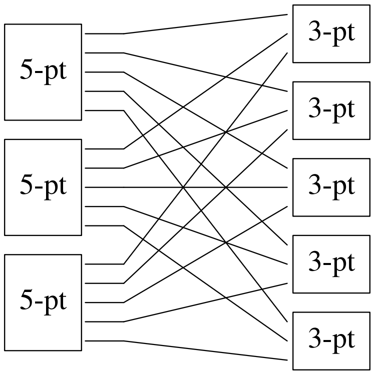


Figure 2: Prime factor decomposition of 15-pt DFT

Winograd's algorithm

We can think of the prime-factor algorithm as:

a) reorder the inputs into an A*C matrix,

b) perform C-pt DFTs on each of the A columns

c) perform A-pt DFTs on each of the transformed C rows

d) reorder the results into the frequency outputs.

The final finesse is Winograd's algorithm[5] which starts with the same decomposition and reordering as the prime-factor algorithm but which exploits the structure of the cyclic-convolution based algorithm (see figure 1) to combine the A- and C-pt DFT algorithms so that their multiplications coincide. We can think of Winograd's algorithm as:

a) reorder the inputs as before into an A*C matrix

b) perform the initial additions for the C-pt DFT on the A columns – in general this results in more than C values so the columns of the matrix grow

c) rather than multiply by the C-pt DFT coefficients, pause this algorithm and perform instead the initial additions of the A-pt DFT on the rows of the (expanded) matrix - in general, this causes the rows to grow

The next stage in both DFT algorithms is a multiplication: each row would be multiplied by one of the coefficients from the C-pt algorithm, and each column by one of the coefficients from the A-pt algorithm. Winograd's algorithm combines these by precalculating the product of these two coefficients for each element of the (expanded) matrix.

d) multiply each element of the expanded matrix by the precalculated combined coefficient

e) perform the post additions for the A-pt DFT on the rows of the matrix

f) perform the post additions for the C-pt DFT on the resulting A columns

g) perform simple additions and reorder the results into complex frequency outputs.

The paper by Kolba and Parks[4] is particularly recommended for a comparison between the prime factor and Winograd algorithms.

Comparisons

Let M_X denote the number of multiplications, and A_X the number of additions, for an X-pt DFT cyclic-convolution based algorithm. Table 1 shows the combined numbers of operations for two factors X and Y such that XY=N using the prime-factor and Winograd algorithms. The calculation is complicated by the fact that in Winograd's algorithm the multipliers of one DFT are multiplied by those of the other, thus the non-trivial multipliers in each must also be considered. However, with these details, table 2 compares the number of real multiplications and additions for various order-N transforms using these and the split-radix algorithms.

	Prime Factor	Winograd
Multiplications	$XM_Y + YM_X$	$M_X\ M_Y$
Additions	$XA_Y + YA_X$	$XA_Y + M_Y A_X$

Table 1: Comparison of prime-factor and Winograd's algorithms

N-pt	Split-radix		Winograd		Prime factor	
DFT	M	A	M	A	M	A
120			276	2076	460	2076
128	516	2308				
240			632	5016	1100	4812
256	1284	5380				
504			1572	14540	2524	13388
512	3076	12292				
1008			3548	34668	5804	29548
1024	7122	27652				

Table 2: Comparison of radix and non-radix DFTs

In terms of multiplications only, the Winograd algorithm appears best; however, in terms of the total number of operations, it is beaten for large DFT sizes by both of the other two algorithms. Thus in choosing an algorithm for a given task, these figures would have to be re-examined and weighted by the cost of the multiply, add, and multiply-add operations.

Let us consider storage requirements. In each of the primitive convolution-based DFT modules, the number of multiplications is slightly larger than the number of inputs. Thus each small DFT requires more work-space than that required to store the inputs. Of course this is true of the radix based algorithms also (for the 2-pt DFT, try calculating a+b and a-b with only storage for a and b) but this is viewed as a small amount of temporary storage with the outputs of each stage overwriting the inputs. In a similar way, the prime-factor algorithms are also in-place since each primitive stage transforms independent sets of inputs into the same number of outputs. In the Winograd algorithm however, there are more inputs to the inner algorithm than to the outer one (since there are more outer multiplications than inputs) and, because of this, the algorithm is not "in-place". However, the storage overheads are not large and so should not preclude consideration of this algorithm.

In some applications, code size is critical in terms of program storage, the initial code writing and the subsequent maintenance; in such applications the simplicity of the radix algorithms has an obvious advantage. However, many real world applications do not fit neatly into powers of two, yet the tendency is to manipulate the overall data processing to accommodate this restriction. When viewed in the context of these coding overheads, a non-radix DFT may be the better choice.

2 REAL VALUED DFTS

The fastest way to write a real valued DFT algorithm is to take an existing complex DFT module and set the imaginary inputs to zero. The algorithm typically uses twice the storage and twice the number of operations as are necessary, but it is really fast to write. In many situations this is the optimal solution. However, since much of engineering data is real, it is worth considering algorithms which are written specifically for purely real data.

One popular method rests upon the property that the even components of the real data (the cosine part) are transformed to purely real and even values while the odd components are transformed to purely imaginary and odd values, and that there is a dual property for purely-imaginary input data. Using this, it is possible to transform two real sequences of length N (s(n) and t(n)) simultaneously using an N-pt complex DFT by setting the inputs to z(n) = s(n) + jt(n). The individual transforms can then be extracted by separating the value of their joint transform into the four components according to real/imaginary and odd/even categories at a cost of about 2N complex additions. This is quite attractive if you already have a complex transform and need to transform two sequences.

A clever variation of this technique is to create two real-valued sequences from one as part of the DFT algorithm. Specifically, the first stage of the radix-2 decimation-in-time algorithm ([1], equation 8) decomposes the N-pt DFT into two N/2-pt DFTs which are recombined by multiplication by twiddle factors and 2-pt DFTs (butterflies). Thus we can perform an N-pt real DFT as two N/2-pt real DFTs using one N/2-pt complex DFT as above, and then recombine them according to the decimation-in-time formula.

However, as explained in the paper by Sorenson *et al*[6], while these approaches are well known they are *never* as good as the *direct* implementations of real-valued DFT algorithms in terms of the number of operations or the simplicity of the code.

The direct implementations are based upon one simple property:

$$\boxed{\text{if } s(n) \text{ are real then } S(k) = S^*(N-k)}$$

where * represents the complex conjugate. Notice that S(0) is real since the definition of the DFT implies that S(0) = S(N), and if N is even then S(N/2) is purely real since S(N/2) = S*(N/2). There are two basic tricks from this property:

(1) only one of the two values needs to be actually calculated

(2) as real data becomes complex in the algorithm, the imaginary value may be stored in the space freed by (1) and so an N-pt real DFT requires only N words of storage.

Real radix based algorithms

The split-radix decimation-in-frequency formula ([1], equation 17) decomposes the DFT into three sets of frequencies: S(2d), S(4d+1) and S(4d+3) where d - 0 $\rightarrow N/4-1$. By applying the first trick, we can see that it is unnecessary to calculate both $S(4d+1)$ and $S(4d+3)$ since

$$S(4d+1) = S*(N-(4d+1)) = S*(4d'+3) \tag{7}$$

where $d' = (N/4 - 1 - d)$ and $d, d' = 0 \rightarrow (N/4 - 1)$; and this applies at each level of the algorithm. Thus the N-pt real DFT is achieved by an N/4-pt complex DFT and an N/2-pt real DFT; and the latter is achieved by an N/8-pt complex DFT and an N/4-pt real DFT; and so on.

By applying the second trick, also at each stage, it is possible to utilize the storage associated with the uncomputed complex DFT to store the imaginary components of the complex DFT which is computed. Thus the N-pt real DFT is achieved with only N words of storage using the mesh shown in figure 3.

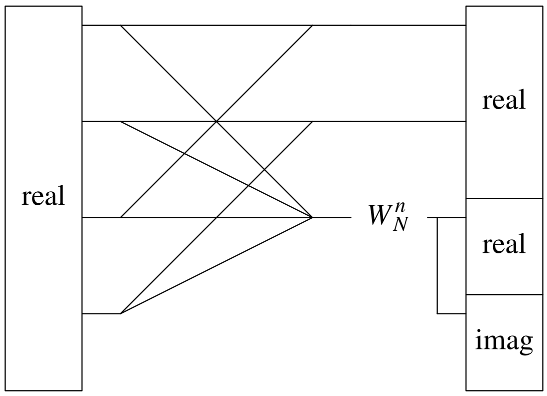


Figure 3: Split-radix real number transformation

For radix algorithms in general the N-pt real DFT requires exactly half the multiplications and storage used for the corresponding N-pt complex algorithm. The only disadvantage of the split-radix algorithm is that the output data is presented in a strange order making unscrambling difficult. Details of the radix-2 decimation-in-time algorithm which avoids this problem may be found in Sorensen *et al*[6].

Real cyclic-convolution based algorithms

Let us consider a single N-pt DFT where N is prime by recalling the structure in figure 1. Because of this structure, a purely real input can be processed by purely real arithmetic throughout phases 1-3 (thus using half the real additions and multiplications of the complex version); phase 4 is redundant in that the outputs of phase 3 are the separated real and imaginary part of the final output.

Of course the same comments hold true for the combination of these modules using the Winograd algorithm which retains the same basic structure.

To see how the prime factor algorithms can be adapted we consider a simple example of a 15-pt DFT decomposed into 3, 5-pt algorithms and 5, 3-pt algorithms as shown in figure 4 (which can be compared with figure 2). The first layer of DFTs can all be constructed as above for real inputs only, thus saving over half the operations as compared with the complex implementation. Of the N outputs of each module, the first is also a purely real number. Thus one DFT in the second layer will also receive purely real inputs and it too can be designed accordingly. Of the other outputs, half are complex conjugates of the others: thus only one of each pair is output and so half of the remaining DFTs in the second layer are redundant. In further layers, the outputs of the remaining "real" DFT modules can have their redundancy removed in the same manner.

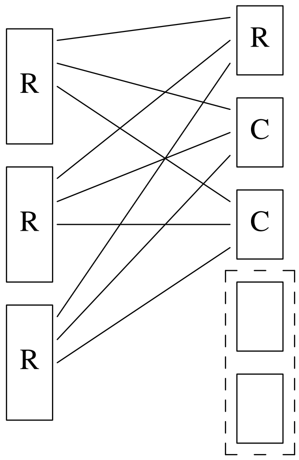


Figure 4: Real redundancy for prime-factor algorithms

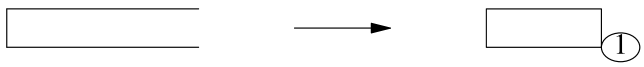
3 FIXED POINT NOISE

Any implementation of these algorithms in fixed point arithmetic must consider the errors which arise. For a discussion of statistical models for error analysis the reader is referred to the classic paper by Tran-Thong and Liu[7]. However, statistical analysis (and the text which follows) is only a guide; the real test for any fixed point algorithm must be performed in exhaustive simulation.

In the following we use Sod's Law analysis on a radix-2 DFT algorithm. Despite popular believe, nothing will ever be this bad but by assuming worst case we will see how the errors arise, how they propagate, and how they relate to each other. We will look at three sources of error and consider the *worst case error*, initially with respect to the least significant bit of the word as stored in computer memory.

Quantization of coefficients and data

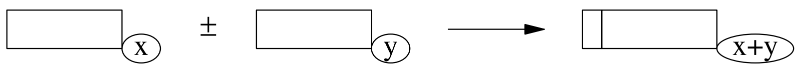
Quantization is the representation of a possibly irrational number in a fixed number of bits. A simple way to achieve this is to discard all the less significant bits (truncation) so that:



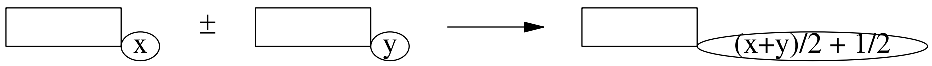
The value in the circle represents the possible error of the last bit of the remaining word. With simple truncation this is "1" since all the discarded bit could (in worst case) themselves have been 1 (and 0.1' = 1.0). If we use rounding instead, then the error would be 1/2.

Scaling of sums

Scaling occurs when two n-bit numbers are added together since they potentially form an n+1 bit sum. In the first instance their worst-case errors add:



however, to maintain a fixed size, we need to shift the result one bit to the right (i.e. discard the least significant bit) giving:



where the summed error is halved (by the shift right) and the addition of 1/2 corresponds to the possible value of the (single) discarded bit.

Truncation of products

The truncation of products occurs because the product of an n- by m-bit number is n+m and so to maintain the fixed point data words we need to discard the least significant bits corresponding to the size of the coefficient. Since we here considering the DFT, we will assume that the error in the coefficient arises only from its quantization and assume that this error is "1". Let the data word be represented by d+1 bits and the coefficient by c+1 bits (where one bit is to represent the sign).

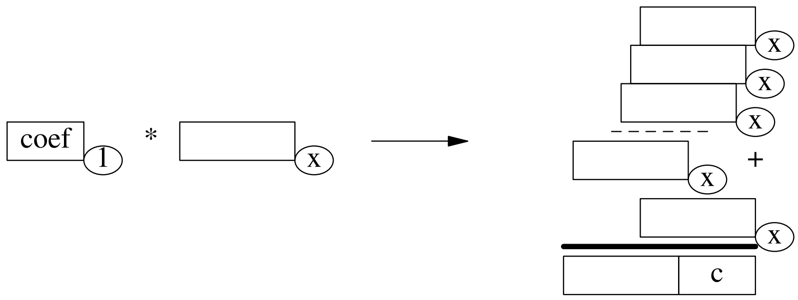
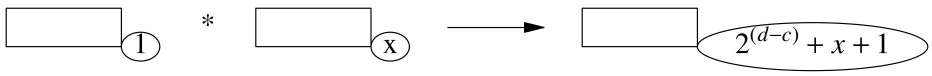


Figure 5: Multiplication and the addition of noise

To explain the effect, we depict the multiplication in figure 5 as a sum of the data word repeated and shifted according to the bits of the coefficient; there is also a final addition of the data word corresponding to its multiplication by the error in the coefficient. The line drawn in the sum corresponds to the boundary at which truncation occurs.

Consider the summation of the data errors. Including the final addition, these sum to exactly "x" on the truncated boundary if (worst case) all the coefficient bits are 1. If the quantization of the coefficient had been by rounding (i.e. the error value of 1/2) then the final value added in the summation would be shifted one bit to the right and the value of "x" for the summation of errors would be an approximate upper bound (to 0.xxx...x). The truncation itself adds a further "1" to the error on the truncated boundary.

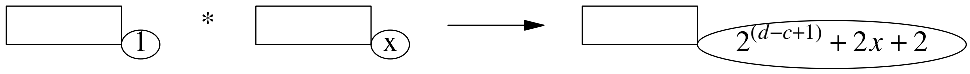
However, there is a further source of error from the coefficient quantization since the extra addition of the data word can overlap the bits of the truncated product leading to an error value of $2^{(d-c)}$. Of course this is really worst case: it assumes that the data word is at its maximum value and that the error in the coefficient is also maximum. However, we proceed anyway and adding up these errors gives:



Complex multiplication

The above analysis holds for the convolution-based algorithms of the DFT since the complex multiplications are all by a purely real or purely imaginary coefficient. However, in the radix-based algorithms the complex multiplication is actually a rotation of the data in the imaginary plain. Thus if the original data can be represent in real and imaginary words of size n-bits, then so too must the rotated values. Thus although the operation involves real additions, there is no need for scaling of the sum.

There are various methods of performing the complex multiplication including different orders of truncation and summation - however, we will assume that each output word is the unscaled sum of two truncated products. The reader might verify that this gives:



Theoretical analysis

Having looked at the individual errors, we now consider how they evolve within a 512-pt radix-2 DFT algorithm. We will look at three cases:

1) Fixed point with quantization of coefficients, truncation of products and shifting of sums.
2) The same without the shifting of sums - therefore assuming that the data word can grow.
3) Quantization of data and coefficients only - to examine the effect of the truncation of products.

The radix-2 DFT is composed of complex additions and subtractions, and the worst case error path is the one with the most multipliers - thus we consider the total errors for a path with nine sums and eight multipliers (in calculating S[511]). Since the length of the data word grows in cases 2 and 3, we will not consider the error at the least significant bit but rather at the bit *d-1* places below the most significant bit where *d* is the number of bits in the original data word. The individual and total worst-case errors are given in table 3.

data error = x, coeff error = 1, initial data error = 1		
Quantization, truncation of products and shifting of sums		
addition	multiplication	total for 512-pt DFT
$x + 1/2$	$2x + 2^{(d-c+1)} + 2$	$\sum_{i=1}^{8} 2^{(d-c+i)} + 1021.5$
Quantization and truncation of products		
addition	multiplication	total for 512-pt DFT
x	$2x + 2^{(d-c+1)} + 2$	$\sum_{i=1}^{8} 2^{(d-c+i)} + 766$
Quantization only		
addition	multiplication	total for 512-pt DFT
x	$2x + 2^{(d-c+1)}$	$\sum_{i=1}^{8} 2^{(d-c+i)} + 256$

Table 3: Worst-case noise for a 512-pt DFT

The similarity between the errors for the individual operations is striking; the totals are quite frightening to a designer. They indicate that even if $d = c$, there are 9-bits of error above the bit under consideration: 16-bit data would lead to an output with only 7-bits accuracy. Fortunately, these are worst cases and practically impossible.

One point is very clear, the choice of the coefficient data length in relation to the initial data length is critical. This analysis suggests that $c \geq d$ will keep this error close to, or below, that due to the truncation of the product − however careful simulations are, of course, necessary to check the effects for specific choices.

Simulation analysis

To illustrate the merits of simulation, we present the results of simple simulations to find the number of reliable bits under the three schemes (compared against 32-bit floating point arithmetic using the same algorithm). Two signal sources were used: a pure sine wave and a noisy pattern built up from 100 sine waves. In both cases all the frequencies, phases, and relative amplitudes were chosen by a random number generator. Table 4 indicates the number of bits which remained reliable over *all* outputs and *all* 10,000 separate trials for different coefficient bit lengths.

Radix-2 Decimation in Frequency DFT order-512 Data length = 16 bits Results from 10,000 trials with 1 (100) random sine wave(s)							
Coef bit length (bits)							
8	10	12	14	16	18	20	22
Quantization of data and coefs with truncation of products and shifting of sums							
5 (9)	7 (11)	9 (12)	10 (12)	11 (12)	12 (12)	12 (12)	12 (12)
Quantization of data and coefs with truncation of products							
5 (9)	7 (11)	9 (13)	10 (14)	12 (15)	13 (15)	14 (15)	14 (15)
Quantization of data and coefs only							
5 (9)	7 (12)	9 (13)	11 (15)	12 (16)	14 (16)	15 (17)	15 (17)

Table 4: Simulation of noise in 512-pt radix-2 decimation-in-frequency algorithm

Reading the table from left to right, the number of bits of accuracy increases with the coefficient bit length; this corresponds to the $2^{(d-c)}$ term in the theoretical analysis. This increase continues until the other, constant terms begin to dominate. The differences in the final values reflect the difference in the corresponding constant terms. The effect of the $2^{(d-c)}$ term is worse on pure sine waves because the maximum frequency amplitude is higher than in the noisy case and so the overlap of the final error-sum term in the multiplications will give rise to greater errors.

Most real world signals are noisy (a mixture of many sine waves) which is fortunate since the algorithm performs far better with this type of input. In this case we see that algorithms with shifting of sums can use coefficient bit lengths shorter than the data without affecting performance. However without shifting of sums, the word length of the coefficients should be the same as the data. Without truncation, the results from noisy data are actually more accurate than the input data − this effect is undoubtedly helped by the fact that the final multiplications in decimation-in-frequency are trivial and therefore exact.

Finally we should note that these values are the worst of the worst errors over 10,000 signals - the mean error (across all signals and all runs) for the "real world" signals was two or three bits lower in all cases. Only by carefully considering the application *and its requirements* can an engineer determine the required precision for the necessary performance.

4 HARDWARE SUPPORT

Hardware support for the DFT is limited to the radix algorithms. There are two approaches: the application specific integrated circuits (ASICs) which try to be flexible enough to cover a range of applications (to achieve a sufficient market size); and the general, programmable digital signal processing (DSP) chips which add specific submodules to accelerate radix DFTs (since it is so commonly used).

The ASIC solution has the advantage that the whole architecture can be optimized for the one problem, but in commercial terms the cost of the device (and the design effort) is influenced by the fact that the potential market is small. DSP chips, however, are destined for a larger market and so can merit higher investment (and require lower margins per chip).

Let us take two examples of ASIC components. The first is the Sharp LH9124 Digital Signal Processor[8] (now marketed by Butterfly DSP as the BDSP9124[9]) which requires a separate component for address generation. It is optimized for fixed point block orientated algorithms and array processing including FIR filters and 2D Complex FFTs. For the DFT algorithms in particular, the performance can be enhanced by cascading components. Data and coefficients can be 8-24 bits wide and are stored off-chip. For comparison, a single-LH9124 1k complex FFT requires 80.7μs and three LH9124s in parallel achieve the same result in 25.6μs (with a 40MHz sampling rate).

The second example is the PDSP16510A from GEC Plessey which is a dedicated stand-alone FFT processor[10]. This has internal coefficient ROM and data RAM for up to 1k complex data points. Data and coefficients are 16 bit long but the internal operations use block floating point to increase the dynamic range. Multiple devices can increase the sampling rate. For comparison, a single PDSP16510A 1k complex FFT requires 98μs. six PDSP16510As in parallel achieve the same result is 25.6μs (with a 40MHz sampling rate).

On the other hand, the DSP components also have specialized hardware[9]. The Motorola DSP56001 is a 24-bit fixed point processor with a built-in 256-word sine ROM (sufficient for 1k DFT) and modulo, increment and bit-reversed addressing. Less well known is the Zoran ZR38001 from a family designed specifically for applications at Dolby Laboratories. This includes a special radix-2 butterfly FFT unit and achieves a complex 1k DFT in 67μs.

The differences in performance figures given here are not significant in terms of assigning victory to either the ASIC or the DSP solutions, which is not surprising since they are converging as DSPs add application specific support modules and ASICs use essentially the same general modules (memory, multipliers, adders) as the DSPs. In theory, the ASIC components are to some extent pre-programmed and so should be easier to incorporate within a larger system; and on the other hand, the programmable DSP may also undertake other functions within the system and so can lead to a simpler overall architecture. Ultimately, an ASIC design should lead to a superior solution since it allows all modules, memory sizes and data buses to be optimized for one algorithm – but whether the design costs are merited by the gains this achieves over DSP-chips (optimized for a larger but similar class of algorithms) has yet to be demonstrated.

5 CONCLUSION

The fast DFT algorithms have now reached maturity. The simple radix fast Fourier transform (FFT) has been refined to a possibly optimal variation; small building blocks of prime number orders (and powers of prime numbers) are used to construct efficient algorithms for many other orders; real number transforms are properly seen as non-redundant versions of the complex algorithms.

Aided by the increasing availability of processing power, the DFT has been applied to areas as diverse as detecting pulsars, speech encoding and seismic monitoring of nuclear tests. With modern DSPs, we can now perform large DFTs in "real-time" and the rising levels of integration are continuously pushing for even higher orders. Such advances motivate applications as well as support them. For instance, a recently reported single-chip DFT[11] achieves an 8k-pt complex DFT in 400μs (and a 1k complex in 50μs), and was designed to validate and support a particular scheme for digital television broadcasting − this is surely an impressive outcome for a little paper[12] published thirty years ago.

References

1. Gerard M Blair, ''A review of the discrete Fourier transform: Part 1 retarding waves and the powers of two'', *IEE Electronics & Communication Engineering Journal*, vol. 7, no. 4, pp. XX-XX, Aug 1995.
2. H J Nussbaumer, *Fast Fourier Transform and Convolution Algorithms,* Springer-Verlag, 1981.
3. C Sidney Burrus, ''Index mappings for multidimensional formulation of the DFT and convolution'', *IEEE Trans Acoustics, speech and signal processing*, vol. ASSP-25, no. 3, pp. 239-242, June 1977.
4. Dean P Kolba and Thomas V Parks, ''A prime factor FFT algorithm using high-speed convolution'', *IEEE Trans Acoustics, Speech, and Signal Processing*, vol. ASSP-25, no. 4, pp. 281-294, Aug 1977.
5. S Winograd, ''On computing the discrete Fourier transform'', *Mathematics of Computation*, vol. 32, no. 141, pp. 175-199, Jan 1978.
6. H V Sorensen, D L Jones, M T Heideman, and C S Burrus, ''Real-valued fast Fourier transform algorithms'', *IEEE Trans Acoustics Speech and Signal Processing*, vol. ASSP-35, no. 6, pp. 849-863, Jun 1987.
7. Tran-Thong and Bede Liu, ''Fixed-point fast fourier transform error analysis'', *IEEE Trans Acoustics Speech and Signal Processing*, vol. ASSP-24, no. 6, pp. 563-573, Dec 1976.
8. ''LH9124, Digital Signal Processor'', *Data Sheet*, Sharp, 1992.
9. *EDN*, May 11th 1995.
10. ''PDSP16510A, Stand alone FFT processor'', *Data Sheet*, GEC Plessey, Aug 1994.
11. E Bidet, D Castelain, C Joanblanq, and P Senn, ''A fast single-chip implementation of 8192 complex point FFT'', *IEEE JSSC*, vol. 30, no. 3, pp. 300-305, Mar 1995.
12. J W Cooley and J W Tukey, ''An algorithm for machine computation of complex Fourier series'', *Math Comput*, vol. 19, pp. 297-301, Apr 1965.

Acknowledgement: the author gratefully acknowledges the assistance of Peter Grant and the encouragement of his supervisor Yuuji Hatano.

Dr Gerard M Blair is currently a Visiting Senior Researcher at Hitachi Ltd, Central Research Laboratory, Tokyo, Japan. He returns this October as a Senior Lecturer in the Department of Electrical Engineering, The University of Edinburgh, The King's Buildings, Edinburgh EH9 3JL.