Signal Processing 6 (1984) 267–278
North-Holland

SIMPLE FFT AND DCT ALGORITHMS WITH REDUCED NUMBER OF OPERATIONS

Martin VETTERLI, member EURASIP, and Henri J. NUSSBAUMER

Laboratoire d'Informatique Technique, Ecole Polytechnique Fédérale de Lausanne, 16 Chemin de Bellerive, CH-1007 Lausanne, Switzerland

Received 2 November 1983
Revised 20 February 1984

Abstract. A simple algorithm for the evaluation of discrete Fourier transforms (DFT) and discrete cosine transforms (DCT) is presented. This approach, based on the divide and conquer technique, achieves a substantial decrease in the number of additions when compared to currently used FFT algorithms (30% for a DFT on real data, 15% for a DFT on complex data and 25% for a DCT) and keeps the same number of multiplications as the best known FFT algorithms. The simple structure of the algorithm and the fact that it is best suited for real data (one does not have to take a transform of two real sequences simultaneously anymore) should lead to efficient implementations and to a wide range of applications.

Zusammenfassung. Ein einfacher Algorithmus zur Berechnung von diskreten Fourier Transformationen (DFT) und diskreten Cosinus Transformationen (DCT) wird vorgeschlagen. Diese Methode, basierend auf der "Teilen und Lösen" Technik, erlaubt eine Verkleinerung der Anzahl Additionen gegenüber gebraüchlichen FFT Algorithmen (30% für eine DFT von einem reellen Signal, 15% für eine DFT von einem complexen Signal und 25% für eine DCT) und braucht gleichviel Multiplikationen wie die besten bekannten FFT Algorithmen. Die einfache Struktur des Algorithmus und der Fakt dass er am besten für reelle Signale geeignet ist (man braucht nicht mehr gleichzeitig zwei reelle Signale zu transformieren) sollten zu effizienter Implementierung und zu zahlreichen Applikationen führen.

Résumé. Un algorithme simple pour l'évaluation de la transformée de Fourier discrète (DFT) et de la transformée en cosinus discrète (DCT) est proposé. Cette approche, basée sur la méthode de la "division et solution", permet une diminution substantielle du nombre d'additions par rapport aux algorithmes de FFT courants (30% pour une DFT de signaux réels, 15% pour une DFT de signaux complexes et 25% pour une DCT) tout en gardant un nombre de multiplications égal à celui des meilleurs algorithmes de FFT connus. La structure simple de l'algorithme ainsi que le fait qu'il s'applique bien aux signaux réels (il n'y a plus besoin de prendre la transformée de deux signaux réels simultanément) devraient conduire à une implantation efficace ainsi qu'à un large champ d'applications.

Keywords. Fast Fourier transform, fast cosine transform, transforms of real data.

1. Introduction

Since the rediscovery of the fast Fourier transform (FFT) algorithm [1, 2] for the evaluation of discrete Fourier transforms, several improvements have been made to the basic divide and conquer scheme as for example the mixed radix FFT [3] and the real factor FFT [4, 5]. The introduction of the Winograd Fourier transform (WFTA) [6], although a beautiful result in complexity theory, did not bring the expected improvements once implemented on real life computers [7], essentially due to the large total number of operations and to the structural complexity of the algorithm.

The fact that most FFT's are taken on real data is seldom fully taken into account. The algorithm using a FFT of half dimension for the computation of a DFT on a real sequence [8] uses substantially more operations than the method of computing a single FFT on two real sequences simultaneously [9]. The

0165-1684/84/$3.00 © 1984, Elsevier Science Publishers B.V. (North-Holland)

latter method has the disadvantage that one has to take two DFT's at once and that the sorting of the output uses additional adds. The fact that the input and output sequences are real is used explicitly in a real convolution algorithm [10] where the DFT and inverse DFT are computed with a single complex FFT.

Another transform that is mostly applied to real data is the discrete cosine transform. Since the introduction of the DCT [11], the search for a fast algorithm followed two main different approaches. One was to compute the DCT through a FFT of same dimension [12], where one is bound to take two transforms simultaneously. The other was a direct approach, leading to rather involved algorithms [13]. It should be noted that the former technique outperforms all the latter ones when using optimal FFT's, a fact often left in the dark [14].

Recently, evaluation of signal processing algorithms has shifted away from multiplication counts alone to the counting of the total number of operations, including data transfers [15]. This is due to the fact that the ratios (multiplication time)/(addition time) and (multiplication time)/(load time) are close to one on most computers and signal processors. Another growing concern has been the generation of time efficient software [16], and finally, the efficiency of an algorithm turns out to be a non-trivial combination of the various operation counts as well as of its structural complexity [17].

In this communication, we address an old problem, namely, the efficient evaluation of DFT's and DCT's of real data. Efficiency is meant in the sense of minimal number of multiplications and additions as well as in the sense of structural simplicity. As it turns out, the two problems are closely related, since a DFT of dimension N can be evaluated with two DCT's of size $N/4$ and since a DCT of size N can be evaluated with a DFT of size N and additional operations. The same technique can be applied again to the reduced DCT of size $N/4$ and to the DFT of size N, and this until only trivial transforms are left over ($N = 1, 2$).

This leads to an elegant recursive formulation of the two algorithms and to a number of multiplications identical to the best FFT's while diminishing substantially the number of additions (typically 30%). Interestingly, this last saving is partly kept when computing complex DFT's, and as an example, the total number of operations for a 1024-point transform is nearly 10% below the number of operations required for a 1008-point WFTA. The prime factor FFT (PFA) requires about the same number of operations [18], but has a more complex structure.

Note that the algorithms below were developed while searching for an efficient way to compute DCT's of real data. The derived FFT algorithm for real data that follows immediately requires a number of multiplications identical to the one found in [19] (which is a variation of the Rader–Brenner algorithm), and a total number of operations that can be found in [20]. While obtaining an identical complexity, the derivations are quite different and the algorithm below seems more suitable for programming.

Section 2 is used to derive the general algorithm and Section 3 evaluates its computational complexity. In Section 4, the results are compared to other algorithms and some implementation considerations are addressed.

2. Derivation of the algorithms

Let us define the following transforms of the length-N real vector x with elements $x(0), x(1) \cdots x(N-1)$:

Discrete Fourier transform

$$\mathrm{DFT}(k, N, x) := \sum_{n=0}^{N-1} x(n) \cdot \mathrm{e}^{-\mathrm{j}2\pi nk/N}, \quad k = 0, \ldots, N-1, \tag{1}$$

where $\mathrm{j} = +\sqrt{-1}$.

Discrete cosine transform

$$\mathrm{DCT}(k, N, x) := \sum_{n=0}^{N-1} x(n) \cdot \cos\left(\frac{2\pi(2n+1)k}{4N}\right), \quad k = 0, \ldots, N-1. \tag{2}$$

Cosine DFT

$$\text{cos-DFT}(k, N, x) := \sum_{n=0}^{N-1} x(n) \cdot \cos\left(\frac{2\pi nk}{N}\right), \quad k = 0, \ldots, N-1. \tag{3}$$

Sine DFT

$$\text{sin-DFT}(k, N, x) := \sum_{n=0}^{N-1} x(n) \cdot \sin\left(\frac{2\pi nk}{N}\right), \quad k = 0, \ldots, N-1. \tag{4}$$

Note that in the definition of the DCT, the normalizing factor $1/\sqrt{2}$ at $k = 0$ is omitted for simplicity. While the transforms are usually only defined for $k = 0, \ldots, N-1$, one will see that other values are useful as well and can easily be obtained from the ones defined above.

Obviously, the following relations hold:

$$\mathrm{DFT}(k, N, x) = \text{cos-DFT}(k, N, x) - \mathrm{j}\,\text{sin-DFT}(k, N, x), \tag{5}$$

$$\mathrm{DCT}(N, N, x) = 0, \tag{6}$$

$$\mathrm{DCT}(-k, N, x) = \mathrm{DCT}(k, N, x), \tag{7}$$

$$\mathrm{DCT}(2N-k, N, x) = -\mathrm{DCT}(k, N, x), \tag{8}$$

$$\text{cos-DFT}(N-k, N, x) = \text{cos-DFT}(k, N, x), \tag{9}$$

$$\text{sin-DFT}(N-k, N, x) = -\text{sin-DFT}(k, N, x). \tag{10}$$

Looking at the evaluation of cos-DFT(k, N, x), we note that since the cosine function is even:

$$\text{cos-DFT}(k, N, x) = \sum_{n=0}^{N/2-1} x(2n) \cdot \cos\left(\frac{2\pi nk}{N/2}\right) + \sum_{n=0}^{N/4-1} (x(2n+1) + x(N-2n-1)) \cdot \cos\left(\frac{2\pi(2n+1)k}{4 \cdot N/4}\right), \tag{11}$$

or, in a more succinct form:

$$\begin{aligned} &\text{cos-DFT}(k, N, x) = \text{cos-DFT}(k, N/2, x_1) + \mathrm{DCT}(k, N/4, x_2), \quad k = 0, \ldots, N-1, \\ &\text{with} \quad x_1(n) = x(2n), \qquad\qquad n = 0, \ldots, N/2-1, \\ &\qquad x_2(n) = x(2n+1) + x(N-2n-1), \quad n = 0, \ldots, N/4-1, \end{aligned} \tag{12}$$

where (8) or (9) are applied when necessary.

Hence, the cosine DFT of dimension N has been mapped into a cosine DFT of $N/2$ and a DCT of $N/4$, at a cost of $N/4$ input and $N/2$ output additions (note that cos-DFT(k, N, x) is evaluated for $k = 0, \ldots, N/2$ but that $k = N/4$ does not require an output addition). The cosine DFT of $N/2$ can be handled similarly (and this until the transform becomes trivial) and the case of the DCT will be treated below.

Turning to the sine DFT, we take a similar approach, using the fact that the sine function is odd.

$$\text{sin-DFT}(k, N, x) = \sum_{n=0}^{N/2-1} x(2n) \cdot \sin\left(\frac{2\pi nk}{N/2}\right) + \sum_{n=0}^{N/4-1} (x(2n+1) - x(N-2n-1)) \cdot \sin\left(\frac{2\pi(2n+1)k}{4 \cdot N/4}\right). \tag{13}$$

Using the following identity:

$$\sin\left(\frac{2\pi(2n+1)k}{N}\right) = (-1)^n \cdot \cos\left(\frac{2\pi(2n+1)(N/4-k)}{N}\right), \tag{14}$$

we can rewrite (13), using (8) or (10) when necessary, as:

$$\text{sin-DFT}(k, N, x) = \text{sin-DFT}(k, N/2, x_1) + \text{DCT}(N/4-k, N/4, x_3), \quad k = 0, \ldots, N-1,$$

$$\text{with} \quad x_3(n) = (-1)^n \cdot (x(2n+1) - x(N-2n-1)). \tag{15}$$

Therefore, the sine DFT of dimension N has been mapped into a sine DFT of $N/2$ and a DCT of $N/4$, at the cost of $N/4$ input and $(N/2)-2$ output additions (note that sin-DFT(k, N, x) is evaluated for $k = 0, \ldots, N/2$ but that $k = 0$, $N/4$ and $N/2$ do not require any output additions). The sine DFT of $N/2$ is handled in a similar fashion until the length is reduced so that the transform becomes trivial.

We now focus our attention on the computation of the DCT. Using the following mapping [12]:

$$\begin{aligned} &x_4(n) = x(2n), \\ &x_4(N-n-1) = x(2n+1), \quad n = 0, \ldots, N/2-1, \end{aligned} \tag{16}$$

the DCT can be evaluated as:

$$\text{DCT}(k, N, x) = \sum_{n=0}^{N-1} x_4(n) \cdot \cos\left(\frac{2\pi(4n+1)k}{4N}\right), \quad k = 0, \ldots, N-1. \tag{17}$$

Using basic trigonometry, (17) becomes:

$$\text{DCT}(k, N, x) = \cos\left(\frac{2\pi k}{4N}\right) \cdot \text{cos-DFT}(k, N, x_4) - \sin\left(\frac{2\pi k}{4N}\right) \cdot \text{sin-DFT}(k, N, x_4), \quad k = 0, \ldots, N-1. \tag{18}$$

With the symmetries of trigonometric functions, (18) can be computed as follows:

$$\begin{bmatrix} \text{DCT}(k, N, x) \\ \text{DCT}(N-k, N, x) \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{2\pi k}{4N}\right) & -\sin\left(\frac{2\pi k}{4N}\right) \\ \sin\left(\frac{2\pi k}{4N}\right) & \cos\left(\frac{2\pi k}{4N}\right) \end{bmatrix} \cdot \begin{bmatrix} \text{cos-DFT}(k, N, x_4) \\ \text{sin-DFT}(k, N, x_4) \end{bmatrix}, \qquad k = 0, \ldots, N/2-1$$

and

$$\text{DCT}(N/2, N, x) = \cos(\pi/4) \cdot \text{cos-DFT}(N/2, N, x_4). \tag{19}$$

The matrix product in (19) can be evaluated with 3 multiplications and 3 additions (instead of 4 multiplications and 2 additions) [9] as follows:

$$p_1 = \cos\left(\frac{2\pi k}{4N}\right) + \sin\left(\frac{2\pi k}{4N}\right),$$

$$p_2 = \sin\left(\frac{2\pi k}{4N}\right) - \cos\left(\frac{2\pi k}{4N}\right),$$

$$s_1 = \text{cos-DFT}(k, N, x_4) + \text{sin-DFT}(k, N, x_4),$$

$$m_1 = s_1 \cdot \cos\left(\frac{2\pi k}{4N}\right),$$

$$m_2 = \text{sin-DFT}(k, N, x_4) \cdot p_1,$$

$$m_3 = \text{cos-DFT}(k, N, x_4) \cdot p_2,$$

$$\text{DCT}(k, N, x) = m_1 - m_2,$$

$$\text{DCT}(N-k, N, x) = m_1 + m_3, \tag{20}$$

where p_1 and p_2 are precomputed. Using all simplifications, the computation of (19) requires therefore a total of $(3N/2)-2$ multiplications and $(3N/2)-3$ additions.

Thus, we have shown how to map a N dimensional DFT into two DCT's of $N/4$ (5, 12 and 15) and how to map a DCT of N into a DFT of same size (16 and 19). In other words, since the DCT is computed through a DFT, it is shown how to compute a DFT of N with 2 DFT's of $N/4$ plus auxiliary operations, and this operation can be repeated until N has been reduced sufficiently so as to lead to trivial transforms. Figures 1 to 4 try to visualize schematically the interaction of the various transforms. Combination of these figures in appropriate order show how to compute a transform of any dimension.

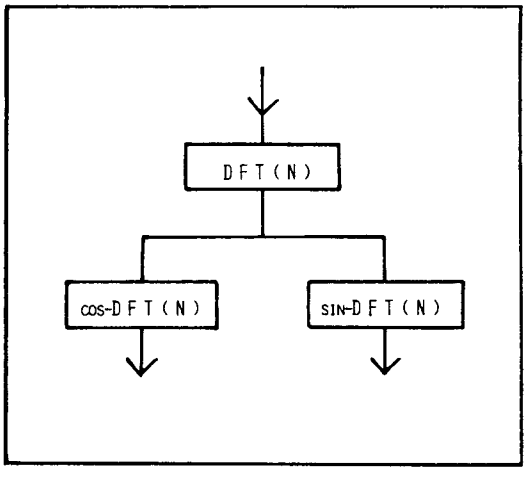

Fig. 1. Decomposition of a DFT of size N into a cosine DFT of size N and a sine DFT of size N.

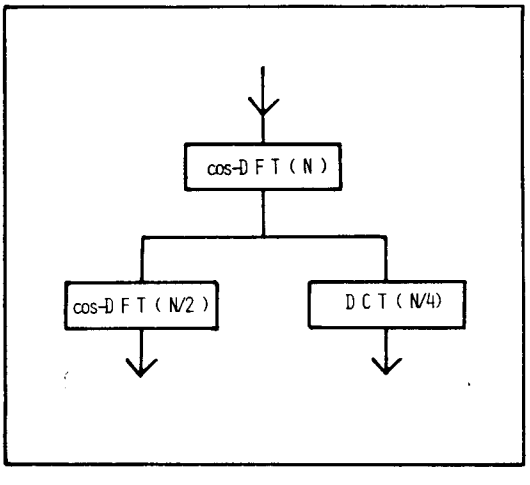

Fig. 2. Decomposition of a cosine DFT of size N into a cosine DFT of size $N/2$ and a DCT of size $N/4$.

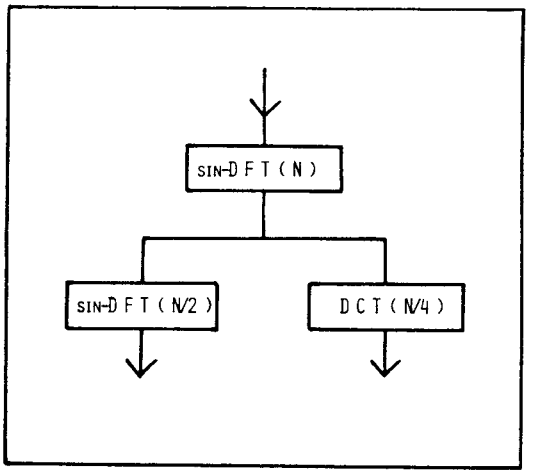

Fig. 3. Decomposition of a sine DFT of size N into a sine DFT of size $N/2$ and a DCT of size $N/4$.

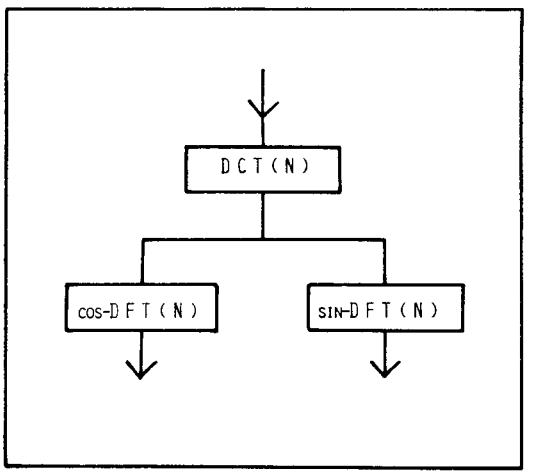

Fig. 4. Decomposition of a DCT of size N into a cosine DFT of size N and a sine DFT of size N plus auxiliary operations.

The algorithm is best suited for DFT's on real data, but if the input is complex, one simply takes the real and imaginary parts separately (thus doubling the computational load) and evaluates the output with $2N-4$ auxiliary adds ($k=0$ and $N/2$ require no additions).

3. Computational complexity

Even if the derivation above used only the fact that N was a multiple of 4, the algorithm, as other divide and conquer schemes, performs best when N is highly composite, typically a power of 2.

Below, we restate the number of operations required for the various steps needed in the evaluation of the transforms, where $O_M[\cdot]$ and $O_A[\cdot]$ stand for the number of multiplies and adds respectively.

DFT on length-N real data:

$$O_M[\text{R-DFT}(N)] = O_M[\text{cos-DFT}(N)] + O_M[\text{sin-DFT}(N)],$$
$$O_A[\text{R-DFT}(N)] = O_A[\text{cos-DFT}(N)] + O_A[\text{sin-DFT}(N)]. \tag{21}$$

DFT on length-N complex data:

$$O_M[\text{C-DFT}(N)] = 2 \cdot (O_M[\text{cos-DFT}(N)] + O_M[\text{sin-DFT}(N)]),$$
$$O_A[\text{C-DFT}(N)] = 2 \cdot (O_A[\text{cos-DFT}(N)] + O_A[\text{sin-DFT}(N)]) + 2N - 4. \tag{22}$$

DCT on length-N real data:

$$O_M[\text{DCT}(N)] = O_M[\text{cos-DFT}(N)] + O_M[\text{sin-DFT}(N)] + (3N/2) - 2,$$
$$O_A[\text{DCT}(N)] = O_A[\text{cos-DFT}(N)] + O_A[\text{sin-DFT}(N)] + (3N/2) - 3. \tag{23}$$

Cosine DFT on length-N real data:

$$O_M[\text{cos-DFT}(N)] = O_M[\text{DCT}(N/4)] + O_M[\text{cos-DFT}(N/2)],$$
$$O_A[\text{cos-DFT}(N)] = O_A[\text{DCT}(N/4)] + O_A[\text{cos-DFT}(N/2)] + (3N/4). \tag{24}$$

Sine DFT on length-N real data:

$$O_M[\text{sin-DFT}(N)] = O_M[\text{DCT}(N/4)] + O_M[\text{sin-DFT}(N/2)],$$
$$O_A[\text{sin-DFT}(N)] = O_A[\text{DCT}(N/4)] + O_A[\text{sin-DFT}(N/2)] + (3N/4) - 2. \tag{25}$$

From (21)–(25) one can compute recursively the number of operations needed for the various transform types and sizes greater than 2. For $N = 1$, no operations are required, and the values for $N = 2$ are given in (26), thus defining the initial conditions for the above recursions.

$$O_M[\text{R-DFT}(2)] = O_M[\text{C-DFT}(2)] = O_M[\text{cos-DFT}(2)] = O_M[\text{sin-DFT}(2)] = 0,$$
$$O_M[\text{DCT}(2)] = 1,$$
$$O_A[\text{R-DFT}(2)] = O_A[\text{DCT}(2)] = O_A[\text{cos-DFT}(2)] = 2, \tag{26}$$
$$O_A[\text{C-DFT}(2)] = 4,$$
$$O_A[\text{sin-DFT}(2)] = 0.$$

But, when N is a power of 2 and that the recursions are therefore applied $\text{Log}_2 N$ times, the operation counts for the DCT reduces simply to:

$$O_M[\text{DCT}(N)] = N/2 \cdot \text{Log}_2(N),$$
$$O_A[\text{DCT}(N)] = N/2 \cdot (3\ \text{Log}_2(N) - 2) + 1. \tag{27}$$

From (27) it follows immediately with (21)–(23) that:

$$O_M[\text{R-DFT}(N)] = N/2 \cdot (\text{Log}_2(N) - 3) + 2,$$
$$O_A[\text{R-DFT}(N)] = N/2 \cdot (3\ \text{Log}_2(N) - 5) + 4, \tag{28}$$

and

$$O_M[\text{C-DFT}(N)] = N \cdot (\text{Log}_2(N) - 3) + 4,$$
$$O_A[\text{C-DFT}(N)] = 3N \cdot (\text{Log}_2(N) - 1) + 4. \tag{29}$$

The above closed form expressions can now easily be compared to existing algorithms.

4. Comparison with existing algorithms

First, the introduced algorithm (which is called in the following fast Fourier-cosine transform or FFCT) is compared to other algorithms which work directly on real data. Then we look at the case where two transforms on real data are taken simultaneously with an optimal FFT algorithm. The case of the FFT on complex data is investigated as well, and the issue of algorithm structure is addressed. The operation counts for FFT's are taken from [9] unless otherwise specified.

For the computation of a DFT on real data, one can use the algorithm based on an FFT of $N/2$ [8]. Using again the 3 mult/3 adds approach for the auxiliary output operations, one gets the following complexity:

$$O_M[\text{FFT}(N)] = O_M[\text{FFT}(N/2)] + (3N/2) - 2,$$
$$O_A[\text{FFT}(N)] = O_A[\text{FFT}(N/2)] + (9N/2) - 2. \tag{30}$$

Table 1 compares this result with the FFCT and shows the substantial savings that are obtained.

Table 1

Comparison of direct computation of a DFT on real data with a FFT of $N/2$ or the FFCT algorithm (the FFT of $N/2$ is computed with the Rader–Brenner algorithm)

size	FFT of $N/2$ + ops.		FFCT algorithm	
N	real mults	real adds	real mults	real adds
8	10	42	2	20
16	26	122	10	60
32	66	290	34	164
64	162	710	98	420
128	386	1 678	258	1 028
256	898	3 870	642	2 436
512	2 050	8 766	1 538	5 636
1 024	4 610	19 582	3 586	12 804
2 048	10 242	43 262	8 194	28 676

Turning to the DCT, one can compare the FFCT to fast discrete cosine algorithms which perform the transform directly, without going to the FFT (and therefore to the need to perform 2 transforms on real data at the same time). The various operation counts are shown in Table 2. The number of multiplications for the FFCT is always below the other algorithms, and even if the number of additions is slightly above the Chen *et al.* version, the total number of operations is always substantially less.

Table 2

Comparison of direct computation of a DCT on real data with the algorithms of Chen *et al.* [13], Wang *et al.* [14] and the FFCT (the operation counts for the two first algorithms are taken from [14])

size	Chen *et al.*		Wang *et al.*		FFCT	
N	*r.* mults	*r.* adds	*r.* mults	*r.* adds	*r.* mults	*r.* adds
8	16	26	13	29	12	29
16	44	74	35	83	32	81
32	116	194	91	219	80	209
64	292	482	227	547	192	513
128	708	1 154	547	1 315	448	1 217
256	1 668	2 690	1 283	3 075	1 024	2 817
512	3 844	6 146	2 947	7 043	2 304	6 401
1 024	8 708	13 826	6 659	15 875	5 120	14 337
2 048	19 460	30 722	14 851	35 331	11 264	31 745

The other general approach to transforms on real data is to compute simultaneously the transform of two real sequences. Beside the drawback that one has always to compute two transforms (otherwise the approach is suboptimal), the sorting of the output requires about N output additions. The operation count for a DFT on real data are given below:

$$O_M[\text{R-DFT}(N)] = 1/2 \cdot O_M[\text{FFT}(N)],$$
$$O_A[\text{R-DFT}(N)] = 1/2 \cdot O_A[\text{FFT}(N)] + N - 2. \tag{31}$$

Table 3 compares (31) with the FFCT, and, as can be seen, the multiplication count is identical while a saving of about 30% is made with regard to additions.

The situation is quite similar when computing a DCT with an FFT [12]. It is assumed that 2 DCT's on real data are evaluated at the same time. Thus, the operation count is:

$$O_M[\text{R-DFT}(N)] = 1/2 \cdot O_M[\text{FFT}(N)] + (3N/2) - 2,$$
$$O_A[\text{R-DFT}(N)] = 1/2 \cdot O_A[\text{FFT}(N)] + (5N/2) - 5. \tag{32}$$

Table 3

Comparison of the computation of a DFT on real data with an FFT of N on two data sequences simultaneously and output additions or the FFCT algorithm (the FFT of N is computed with the Rader–Brenner algorithm)

size	FFT of N+adds		FFCT algorithm	
N	real mults	real adds	real mults	real adds
8	2	32	2	20
16	10	88	10	60
32	34	242	34	164
64	98	614	98	420
128	258	1 486	258	1 028
256	642	3 486	642	2 436
512	1 538	7 998	1 538	5 636
1 024	3 586	18 046	3 586	12 804
2 048	8 194	40 190	8 194	28 676

A technique similar to the one used in (23) was used in the computation of the output of the DCT. The results are compared in Table 4 where it is seen that the additions are reduced in the FFCT by about 25% with an identical number of multiplies.

Table 4

Comparison of the computation of a DCT on real data with the algorithm using an FFT on two data sequences simultaneously or the FFCT (the FFT is the Rader–Brenner version)

size	FFT of N + ops.		FFCT algorithm	
N	real mults	real adds	real mults	real adds
8	12	41	12	29
16	32	109	32	81
32	80	287	80	209
64	192	707	192	513
128	448	1 675	448	1 217
256	1 024	3 867	1 024	2 817
512	2 304	8 763	2 304	6 401
1 024	5 120	19 579	5 120	14 337
2 048	11 264	43 259	11 264	31 745

Turning to the computation of a DFT on a complex input, one can use (29) for the FFCT. The comparison to the Rader–Brenner FFT is given in Table 5 where the number of multiplies turns out to be identical while the number of additions is reduced by nearly 20%.

Looking at last at the Winograd Fourier transform, one sees that even if the number of multiplies is larger in the FFCT case, the total number of operations is smaller for large transforms (for example 5% for the WFTA(504)/FFCT(512) and 9% for the WFTA(1008)/FFCT(1024) comparison). Note that the gain is less in the PFA case (5% and 2% respectively).

Concerning the WFTA and the PFA, one recalls that its main drawback is the involved structure of the algorithm. Even if the structure of the FFCT is not as straightforward as the radix-2 FFT, it remains simple (it is similar to the structure of the Rader–Brenner algorithm). In the FFCT, the reduction in the number of operations is obtained at the cost of additional permutations and data transfers. Even if these

Table 5

Comparison of the Rader–Brenner FFT with the FFCT when applied to complex data

size	FFT of N		FFCT algorithm	
N	real mults	real adds	real mults	real adds
8	4	52	4	52
16	20	148	20	148
32	68	424	68	388
64	196	1 104	196	964
128	516	2 720	516	2 308
256	1 284	6 464	1 284	5 380
512	3 076	14 976	3 076	12 292
1 024	7 172	34 048	7 172	27 652
2 048	16 388	76 288	16 388	61 444

permutations can be grouped with previous ones (for example at the entry of the DCT), the resulting structure is more complex. But this increase in topological complexity is annihilated by the substantial decrease in arithmetic complexity.

Looking at the implementation, one sees the importance of using very efficient small transforms. Together with the explicit coding technique [16], this should lead to fast code for real transforms, especially on micro and signal processors, where the number of arithmetic and data registers is small (leading to unefficient implementations of the complex transform on two real sequences version).

5. Concluding remarks

We have introduced a simple, recursive algorithm for the computation of the discrete Fourier transform and the discrete cosine transform. First, this algorithm can be applied directly to real data, providing therefore an attractive alternative to the method using a complex transform of two real sequences. Secondly, it uses the same number of multiplies as the best structured FFT algorithms but decreases the number of additions by about 25 to 30% for DFT's and DCT's on real data and by nearly 20% for DFT's on complex data when compared to currently used algorithms. At last, its structure is simple enough so that it should lead to efficient implementations.

It is worth noting that the proposed algorithm, while leading to the same computational complexity for the FFT as the one in [20], is not isomorphic. Interestingly, the algorithm in [20] has been around for 15 years but was seldom referenced and even less used. Meanwhile, literature was published on ameliorations of the Cooley–Tukey FFT which leads to less efficient algorithms than the one in [20].... More recently, actually just as the final copy was being sent to the publisher, yet another algorithm appeared that leads to the same complexity for the complex FFT [21].

Investigations are under way in order to generalize the above ideas to other sinusoidal transforms and to higher dimensions, as well as to prove the efficiency of an implementation.

Acknowledgements

The authors would like to thank one of the anonymous reviewers for bringing [20] and [21] to their attention as well as the "Fonds National Suisse de la Recherche Scientifique" for supporting this research.

References

[1] C. Runge, and H. Konitz, *Vorlesungen über Numerisches Rechnen*, Springer Verlag, Berlin, 1924.

[2] 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, April 1965.

[3] R. Singleton, "An algorithm for computing the mixed radix fast fourier transform", *IEEE Trans. on Audio. and Electroacoustics*, Vol. AU-17, pp. 93–103, June 1969.

[4] 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.

[5] G. Bruun, "z-transform DFT filters and FFT's", *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-26, pp. 56–63, Feb. 1978.

[6] S. Winograd, "On computing the discrete fourier transform", *Proc. Nat. Acad. Sci. USA*, Vol. 73, pp. 1005–1006, April 1976.

[7] L.R. Morris, "A comparative study of time efficient FFT and WFTA programs for general purpose computers", *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-26, pp. 141–150, April 1978.

[8] E.O. Brigham, *The Fast Fourier Transform*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1974.
[9] H.J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*, Springer Verlag, Berlin, 1982.
[10] G.K. McAuliffe, "Fourier digital filter or equalizer and method of operation therefore", US Patent No. 3 679 882, July 25, 1972.
[11] N. Ahmed, T. Natarajan and K.R. Rao, "Discrete cosine transform", *IEEE Trans. on Computers*, Vol. C-23, pp. 88–93, Jan. 1974.
[12] M.J. Narasimha, and A.M. Peterson, "On the computation of the discrete cosine transform", *IEEE Trans. on Communications*, Vol. COM-26, pp. 934–936, June 1978.
[13] W-H. Chen, C.H. Smith and S.C. Fralick, "A fast computational algorithm for the discrete cosine transform", *IEEE Trans. on Communications*, Vol. COM-25, pp. 1004–1009, Sept. 1977.
[14] Z. Wang, and B.R. Hunt, "The discrete cosine transform—A new version", *Int. Conf. on Acoust., Speech, Signal Processing*, ICASSP-83, Boston, 1983.
[15] H. Nawab, and J.H. McClellan, "Bounds on the minimum number of data transfers in WFTA and FFT programs", *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-27, pp. 394–398, Aug. 1979.
[16] L.R. Morris, "Automatic generation of time efficient digital signal processing software", *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-25, pp. 74–78, Feb. 1977.
[17] H.J. Nussbaumer, "Efficient algorithms for signal processing", *Second European Signal Processing Conference*, EUSIPCO-83, Erlangen, 1983.
[18] D.P. Kolba, and T.W. Parks, "A prime factor FFT algorithm using high-speed convolution", *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-25, pp. 281–294, Aug. 1977.
[19] R.D. Preuss, "Very fast computation of the Radix-2 discrete Fourier transform", *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-30, pp. 595–607, Aug. 1982.
[20] R. Yavne, "An economical method for calculating the discrete Fourier transform", *AFIPS Proc.* Vol. 33, pp. 115–125, 1968 Fall Joint Computer Conference, Washington.
[21] P. Duhamel, and H. Hollmann, "Split Radix FFT algorithm", *Electronics Letters*, Vol. 20, No. 1, 5th Jan. 1984.