INVITED PAPER

FAST FOURIER TRANSFORMS: A TUTORIAL REVIEW AND A STATE OF THE ART

P. DUHAMEL
CNET/PAB/RPE 38-40, Rue du General Leclerc, 92131 Issy les Moulineaux, France

M. VETTERLI
Dept of EE and CTR, S.W. Mudd Building, Columbia University, 500 W 120th Street, New York, NY 10027, U.S.A.

Received 22 December 1988
Revised 30 October 1989

Abstract. The publication of the Cooley-Tukey fast Fourier transform (FFT) algorithm in 1965 has opened a new area in digital signal processing by reducing the order of complexity of some crucial computational tasks like Fourier transform and convolution from N^2 to $N \log_2 N$, where N is the problem size. The development of the major algorithms (Cooley-Tukey and split-radix FFT, prime factor algorithm and Winograd fast Fourier transform) is reviewed. Then, an attempt is made to indicate the state of the art on the subject, showing the standing of research, open problems and implementations.

Zusammenfassung. Die Publikation von Cooley-Tukey's schnellem Fourier Transformations Algorithmus in 1965 brachte eine neue Area in der digitalen Signalverarbeitung weil die Ordnung der Komplexität von gewissen zentralen Berechnungen, wie die Fourier Transformation und die digitale Faltung, von N^2 zu $N \log_2 N$ reduziert wurden (wo N die Problemgrösse darstellt). Die Entwicklung der wichtigsten Algorithmen (Cooley-Tukey und Split-Radix FFT, Prime Factor Algorithmus und Winograd's schneller Fourier Transformation) ist nachvollzogen. Dann wird versucht, den Stand des Feldes zu beschreiben, um zu zeigen wo die Forschung steht, was für Probleme noch offenstehen, wie zum Beispiel in Implementierungen.

Résumé. La publication de l'algorithme de Cooley-Tukey pour la transformation de Fourier rapide a ouvert une nouvelle ère dans le traitement numérique des signaux, en réduisant l'ordre de complexité de problèmes cruciaux, comme la transformation de Fourier ou la convolution, de N^2 à $N \log_2 N$ (où N est la taille du problème). Le dévelopment des algorithmes principaux (Cooley-Tukey, split-radix FFT, algorithmes des facteurs premiers, et transformée rapide de Winograd) est décrit. Ensuite, l'état de l'art est donné, et on parle des problèmes ouverts et des implantations.

Keywords. Fourier transforms, fast algorithms, computational complexity.

1. Introduction

Linear filtering and Fourier transforms are among the most fundamental operations in digital signal processing. However, their wide use makes their computational requirements a heavy burden in most applications. Direct computation of both convolution and discrete Fourier transform (DFT) requires on the order of N^2 operations where N is the filter length or the transform size. The breakthrough of the Cooley-Tukey FFT comes from the fact that it brings the complexity down to an order of $N \log_2 N$ operations. Because of the convolution property of the DFT, this result applies to the convolution as well. Therefore, fast Fourier transform algorithms have played a key role in the widespread use of digital signal processing in a variety of applications like telecommunications, medical electronics, seismic processing, radar or radio astronomy to name but a few.

Among the numerous further developments that followed Cooley and Tukey's original contribution, the fast Fourier transform introduced in 1976 by Winograd [54] stands out for achieving a new theoretical reduction in the order of the multiplicative complexity. Interestingly, the Winograd algorithm uses convolutions to compute DFTs, an approach which is just the converse of the conventional method of computing convolutions by means of DFTs. What might look as a paradox at first sight actually shows the deep interrelationship that exists between convolutions and Fourier transforms.

Recently, the Cooley–Tukey type algorithms have emerged again, not only because implementations of the Winograd algorithm have been disappointing, but also due to some recent developments leading to the so-called split-radix algorithm [27]. Attractive features of this algorithm are both its low arithmetic complexity and its relatively simple structure.

Both the introduction of digital signal processors and the availability of large scale integration has influenced algorithm design. While in the sixties and early seventies, multiplication counts alone were taken into account, it is now understood that the number of addition and memory accesses in software, and the communication costs in hardware are at least as important.

The purpose of this paper is first to look back at twenty years of developments since the Cooley–Tukey paper. Among the abundance of literature (a bibliography of more than 2500 titles has been published [33]), we will try to highlight only the key ideas. Then, we will attempt to describe the state of the art on the subject. It seems to be an appropriate time to do so, since on the one hand, the algorithms have now reached a certain maturity, and on the other hand, theoretical results on complexity allow us to evaluate how far we are from optimum solutions. Furthermore, on some issues, open questions will be indicated.

Let us point out that in this paper we shall concentrate strictly on the computation of the discrete Fourier transform, and not discuss applications. However, the tools that will be developed may be useful in other cases. For example, the polynomial products explained in Section 5.1 can immediately be applied to the derivation of fast running FIR algorithms [73, 81].

The paper is organized as follows.

Section 2 presents the history of the ideas on fast Fourier transforms, from Gauss to the split-radix algorithm.

Section 3 shows the basic technique that underlies all algorithms, namely the divide and conquer approach, showing that it always improves the performance of a Fourier transform algorithm.

Section 4 considers Fourier transforms with twiddle factors, that is, the classic Cooley–Tukey type schemes and the split-radix algorithm. These twiddle factors are unavoidable when the transform length is composite with non-coprime factors. When the factors are coprime, the divide and conquer scheme can be made such that twiddle factors do not appear.

This is the basis of Section 5, which then presents Rader's algorithm for Fourier transforms of prime lengths, and Winograd's method for computing convolutions. With these results established, Section 5 proceeds to describe both the prime factor algorithm (PFA) and the Winograd Fourier transform (WFTA).

Section 6 presents a comprehensive and critical survey of the body of algorithms introduced so far, then shows the theoretical limits of the complexity of Fourier transforms, thus indicating the gaps that are left between theory and practical algorithms.

Structural issues of various FFT algorithms are discussed in Section 7.

Section 8 treats some other cases of interest, like transforms on special sequences (real or symmetric) and related transforms, while Section 9 is specifically devoted to the treatment of multi-dimensional transforms.

Finally, Section 10 outlines some of the important issues of implementations. Considerations on software for general purpose computers, digital signal processors and vector processors are made. Then, hardware implementations are addressed. Some of the open questions when implementing FFT algorithms are indicated.

The presentation we have chosen here is constructive, with the aim of motivating the 'tricks' that are used. Sometimes, a shorter but 'plug-in'-like presentation could have been chosen, but we avoided it, because we desired to insist on the mechanisms underlying all these algorithms. We have also chosen to avoid the use of some mathematical tools, such as tensor products (that are very useful when deriving some of the FFT algorithms) in order to be more widely readable.

Note that concerning arithmetic complexities, all sections will refer to synthetic tables giving the computational complexities of the various algorithms for which software is available. In a few cases, slightly better figures can be obtained, and this will be indicated.

For more convenience, the references are separated between books and papers, the latter being further classified corresponding to subject matters (1-D FFT algorithms, related ones, multi-dimensional transforms and implementations).

2. A historical perspective

The development of the fast Fourier transform will be surveyed below, because, on the one hand, its history abounds in interesting events, and on the other hand, the important steps correspond to parts of algorithms that will be detailed later.

A first subsection describes the pre-Cooley-Tukey area, recalling that algorithms can get lost by lack of use, or, more precisely, when they come too early to be of immediate practical use. The developments following the Cooley-Tukey algorithm are then described up to the most recent solutions. Another subsection is concerned with the steps that lead to the Winograd and to the prime factor algorithm, and finally, an attempt is made to briefly describe the current state of the art.

2.1. *From Gauss to the Cooley-Tukey FFT*

While the publication of a fast algorithm for the DFT by Cooley and Tukey in 1965 [25] is certainly a turning point in the literature on the subject, the divide and conquer approach itself dates back to Gauss as noted in a well documented analysis by Heideman et al. [34]. Nevertheless, Gauss's work on FFTs in the early 19th century (around 1805) remained largely unnoticed because it was only published in Latin and this after his death.

Gauss used the divide and conquer approach in the same way as Cooley and Tukey have published it later in order to evaluate trigonometric series, but his work pre-dates even Fourier's work on harmonic analysis (1807)! Note that his algorithm is quite general, since it is explained for transforms on sequences with lengths equal to any composite integer.

During the 19th century, efficient methods for evaluating Fourier series appeared independently at least three times [33], but were restricted on lengths and number of resulting points. In 1903, Runge derived an algorithm for lengths equal to powers of 2 which was generalized to powers of 3 as well and used in the forties. Runge's work was thus quite well-known, but nevertheless disappeared after the war.

Another important result useful in the most recent FFT algorithms is another type of divide and conquer approach, where the initial problem of length $N_1 \cdot N_2$ is divided into subproblems of lengths N_1 and N_2 without any additional operations, N_1 and N_2 being coprime.

This result dates back to the work of Good [32] who obtained this result by simple index mappings. Nevertheless, the full implication of this result will only appear later, when efficient methods will be derived for the evaluation of small, prime length DFTs. This mapping itself can be seen as an application of the Chinese remainder theorem (CRT), which dates back to 100 years A.D.! [10, 18].

Then, in 1965, appears a brief article by Cooley and Tukey, entitled 'An algorithm for the machine calculation of complex Fourier series' [25], which reduces the order of the number of operations from N^2 to $N \log_2(N)$ for a length $N = 2^n$ DFT.

This will turn out to be a milestone in the literature on fast transforms, and will even be credited [14, 15] of the tremendous increase of interest in DSP beginning in the seventies. The algorithm is suited for DFTs on any composite length, and is thus of the type that Gauss had derived almost 150 years before. Note that all algorithms published in-between were more restrictive on the transform length [34].

Looking back at this brief history, one may wonder why all previous algorithms had disappeared or remained unnoticed, whereas the Cooley-Tukey algorithm had such a tremendous success. A possible explanation is that the growing interest in the theoretical aspects of digital signal processing was motivated by technical improvements in the semiconductor technology. And, of course, this was not a one-way street ....

The availability of reasonable computing power produced a situation where such an algorithm would suddenly allow numerous new applications. Considering this history, one may wonder how many other algorithms or ideas are just sleeping in some notebook or obscure publication ....

The two types of divide and conquer approaches cited above produced two main classes of algorithms. For the sake of clarity, we will now skip the chronological order and consider the evolution of each class separately.

2.2. Development of the twiddle factor FFT

When the initial DFT is divided into sublengths which are not coprime, the divide and conquer approach as proposed by Cooley and Tukey leads to auxiliary complex multiplications, initially named twiddle factors, which cannot be avoided in this case.

While Cooley-Tukey's algorithm is suited for any composite length, and explained in [25] in a general form, the authors gave an example with $N = 2^n$, thus deriving what is now called a radix-2 decimation in time (DIT) algorithm (the input sequence is divided into decimated subsequences having different phases). Later, it was often falsely assumed that the initial Cooley-Tukey FFT was a DIT radix-2 algorithm only.

A number of subsequent papers presented refinements of the original algorithm, with the aim of increasing its usefulness.

The following refinements were concerned:
—with the structure of the algorithm: it was emphasized that a dual approach leads to 'decimation in frequency' (DIF) algorithms,
—or with the efficiency of the algorithm, measured in terms of arithmetic operations: Bergland showed that higher radices, for example radix-8, could be more efficient [21],
—or with the extension of the applicability of the algorithm: Bergland, again, showed that the FFT could be specialized to real input data [60], and Singleton gave a mixed radix FFT suitable for arbitrary composite lengths.

While these contributions all improved the initial algorithm in some sense (fewer operations and/or easier implementations), actually no new idea was suggested.

Interestingly, in these very early papers, all the concerns guiding the recent work were already here: arithmetic complexity, but also different structures and even real-data algorithms.

In 1968, Yavne presents a little known paper [58] that sets a record: his algorithm requires the least known number of multiplications, as well as additions for length-2^n FFTs, and this both for real and complex input data. Note that this record still holds, at least for practical algorithms. The same number of operations was obtained later on by other (simpler) algorithms, but due to Yavne's cryptic style, few researchers were able to use his ideas at the time of publication.

Since twiddle factors lead to most computations in classical FFTs, Rader and Brenner, perhaps motivated by the appearance of the Winograd Fourier transform which possesses the same characteristic, proposed an algorithm that replaces all complex multiplications by either real or imaginary ones, thus substantially reducing the number of multiplications required by the algorithm [44]. This reduction in the number of multiplications was obtained at the cost of an increase in the number of additions, and a greater sensitivity to roundoff noise. Hence, further developments of these 'real factor' FFTs appeared in [24, 42], reducing these problems. Bruun also proposed an original scheme [22] particularly suited for real data. Note that these various schemes only work for radix-2 approaches.

It took more than fifteen years to see again algorithms for length-2^n FFTs that take as few operations as Yavne's algorithm. In 1984, four papers appeared or were submitted almost simultaneously [27, 40, 46, 51] and presented so-called 'split-radix' algorithms. The basic idea is simply to use a different radix for the even part of the transform (radix-2) and for the odd part (radix-4). The resulting algorithms have a relatively simple structure and are well adapted to real and symmetric data while achieving the minimum known number of operations for FFTs on power of 2 lengths.

2.3. *FFTs without twiddle factors*

While the divide and conquer approach used in the Cooley-Tukey algorithm can be understood as a 'false' mono- to multi-dimensional mapping (this will be detailed later), Good's mapping, which can be used when the factors of the transform lengths are coprime, is a true mono- to multi-dimensional mapping, thus having the advantage of not producing any twiddle factor.

Its drawback, at first sight, is that it requires efficiently computable DFTs on lengths which are coprime: For example, a DFT of length 240 will be decomposed as $240 = 16 \cdot 3 \cdot 5$, and a DFT of length 1008 will be decomposed in a number of DFTs of lengths 16, 9 and 7. This method thus requires a set of (relatively) small-length DFTs that seemed at first difficult to compute in less than N_i^2 operations. In 1968, however, Rader showed how to map a DFT of length N, N prime, into a circular convolution of length $N-1$ [43]. However, the whole material to establish the new algorithms was not ready yet, and it took Winograd's work on complexity theory, in particular on the number of multiplications required for computing polynomial products or convolutions [55] in order to use Good's and Rader's results efficiently.

All these results were considered as curiosities when they were first published, but their combination, first done by Winograd and then by Kolba and Parks [39] raised a lot of interest in that class of algorithms. Their overall organization is as follows:

After mapping the DFT into a true multi-dimensional DFT by Good's method and using the fast convolution schemes in order to evaluate the prime length DFTs, a first algorithm makes use of the intimate structure of these convolution schemes to obtain a nesting of the various multiplications. This algorithm is known as the Winograd Fourier transform algorithm (WFTA) [54], an algorithm requiring the least known number of multiplications among practical algorithms for moderate lengths DFTs. If the nesting is not used, and the multi-dimensional DFT is performed by the row-column method, the resulting algorithm is known as the prime factor algorithm (PFA) [39] which, while using more multiplications, has less additions and a better structure than the WFTA.

From the above explanations, one can see that these two algorithms, introduced in 1976 and 1977 respectively, require more mathematics to be understood [19]. This is why it took some effort to translate the theoretical results, especially concerning the WFTA, into actual computer code.

It is even our opinion that what will remain mostly of the WFTA are the theoretical results, since although a beautiful result in complexity theory, the WFTA did not meet its expectations once implemented, thus leading to a more critical evaluation of what 'complexity' meant in the context of real life computers [41, 108, 109].

The result of this new look at complexity was an evaluation of the number of additions and data transfers as well (and no longer only of multiplications). Furthermore, it turned out recently that the theoretical knowledge brought by these approaches could give a new understanding of FFTs with twiddle factors as well.

2.4. Multi-dimensional DFTs

Due to the large amount of computations they require, the multi-dimensional DFTs as such (with common factors in the different dimensions, which was not the case in the multi-dimensional translation of a mono-dimensional problem by PFA) were also carefully considered.

The two most interesting approaches are certainly the vector radix FFT (a direct approach to the multi-dimensional problem in a Cooley–Tukey mood) proposed in 1975 by Rivard [91] and the polynomial transform solution of Nussbaumer and Quandalle in 1978 [87, 88].

Both algorithms substantially reduce the complexity over traditional row–column computational schemes.

2.5. State of the art

From a theoretical point of view, the complexity issue of the discrete Fourier transform has reached a certain maturity. Note that Gauss, in his time, did not even count the number of operations necessary in his algorithm. In particular, Winograd's work on DFTs whose lengths have coprime factors both sets lower bounds (on the number of multiplications) and gives algorithms to achieve these [35, 55], although they are not always practical ones. Similar work was done for length-2^n DFTs, showing the linear multiplicative complexity of the algorithm [28, 35, 105] but also the lack of practical algorithms achieving this minimum (due to the tremendous increase in the number of additions [35]).

Considering implementations, the situation is of course more involved since many more parameters have to be taken into account than just the number of operations.

Nevertheless, it seems that both the radix-4 and the split-radix algorithm are quite popular for lengths which are powers of 2, while the PFA, thanks to its better structure and easier implementation, wins over the WFTA for lengths having coprime factors.

Recently, however, new questions have come up because in software on the one hand, new processors may require different solutions (vector processors, signal processors), and on the other hand, the advent of VLSI for hardware implementations sets new constraints (desire for simple structures, high cost of multiplications versus additions).

3. Motivation (or: why dividing is also conquering)

This section is devoted to the method that underlies all fast algorithms for DFT, that is the 'divide and conquer' approach.

The discrete Fourier transform is basically a matrix–vector product. Calling $(x_0, x_1, \ldots, x_{N-1})^{\mathrm{T}}$ the vector of the input samples,

$$(X_0, X_1, \ldots, X_{N-1})^{\mathrm{T}}$$

the vector of transform values and W_N the primitive Nth root of unity ($W_N = \mathrm{e}^{-\mathrm{j}2\pi/N}$), the DFT can be written as

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & W_N^3 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & W_N^6 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{N-1} \end{bmatrix}. \quad (1)$$

The direct evaluation of the matrix-vector product in (1) requires of the order of N^2 complex multiplications and additions (we assume here that all signals are complex for simplicity).

The idea of the 'divide and conquer' approach is to map the original problem into several subproblems in such a way that the following inequality is satisfied:

$$\sum \text{cost(subproblems)} + \text{cost(mapping)} < \text{cost(original problem)}. \tag{2}$$

But the real power of the method is that, often, the division can be applied recursively to the subproblems as well, thus leading to a reduction of the order of complexity.

Specifically, let us have a careful look at the DFT transform in (3) and its relationship with the z-transform of the squence $\{x_n\}$ as given in (4).

$$X_k = \sum_{i=0}^{N-1} x_i W_N^{ik}, \quad k = 0, \dots, N-1, \tag{3}$$

$$X(z) = \sum_{i=0}^{N-1} x_i z^{-i}. \tag{4}$$

$\{X_k\}$ and $\{x_i\}$ form a transform pair, and it is easily seen that X_k is the evaluation of $X(z)$ at point $z = W_N^{-k}$:

$$X_k = X(z)_{z=W_N^{-k}}. \tag{5}$$

Furthermore, due to the sampled nature of $\{x_n\}$, $\{X_k\}$ is periodic, and vice versa: since $\{X_k\}$ is sampled, $\{x_n\}$ must also be periodic.

From a physical point of view, this means that both sequences $\{x_n\}$ and $\{X_k\}$ are repeated indefinitely with period N.

This has a number of consequences as far as fast algorithms are concerned.

All fast algorithms are based on a divide and conquer strategy, we have seen this in Section 2. But how shall we divide the problem (with the purpose of conquering it)?

The most natural way is, of course, to consider subsets of the initial sequence, take the DFT of these subseqnences, and reconstruct the DFT of the initial sequence from these intermediate results.

Let I_l, $l = 0, \dots, r-1$ be the partition of $\{0, 1, \dots, N-1\}$ defining the r different subsets of the input sequence. Equation (4) can now be rewritten as

$$X(z) = \sum_{i=0}^{N-1} x_i z^{-i} = \sum_{l=0}^{r-1} \sum_{i \in I_l} x_i z^{-i}, \tag{6}$$

and, normalizing the powers of z with respect to some x_{0l} in each subset $\mathbf{I}_l$:

$$X(z) = \sum_{l=0}^{r-1} z^{-i_{0l}} \sum_{i \in I_l} x_i z^{-i+i_{0l}}. \tag{7}$$

From the considerations above, we want the replacement of z by W_N^{-k} in the innermost sum of (7) to define an element of the DFT of $\{x_i \mid i \in I_l\}$. Of course, this will be possible only if the subset $\{x_i \mid i \in I_l\}$, possibly permuted, has been chosen in such a way that it has the same kind of periodicity as the initial sequence. In what follows, we show that the three main classes of FFT algorithms can all be casted into the form given by (7).

—In some cases, the second sum will also involve elements having the same periodicity, and hence will define DFTs as well. This corresponds to the case of Good's mapping: all the subsets I_l have the same number of elements $m = N/r$ and $(m, r) = 1$.

—If this is not the case, (7) will define one step of an FFT with twiddle factors: when the subsets I_l all have the same number of elements, (7) defines one step of a radix-r FFT.

—If $r = 3$, one of the subsets having $N/2$ elements, and the other ones having $N/4$ elements, (7) is the basis of a split-radix algorithm.

Furthermore, it is already possible to show from (7) that the divide and conquer approach will always improve the efficiency of the computation.

To make this evaluation easier, let us suppose that all subsets I_l have the same number of elements, say N_1. If $N = N_1 \cdot N_2$, $r = N_2$, each of the innermost sums of (7) can be computed with N_1^2 multiplications, which gives a total of $N_2 N_1^2$, when taking into account the requirement that the sum over $i \in I_l$ defines a DFT. The outer sum will need $r = N_2$ multiplications per output point, that is $N_2 \cdot N$ for the whole sum.

Hence, the total number of multiplications needed to compute (7) is

$$N_2 \cdot N + N_2 \cdot N_1^2 = N_1 \cdot N_2(N_1+N_2) < N_1^2 \cdot N_2^2 \quad \text{if } N1, N2 > 2, \tag{8}$$

which shows clearly that the divide and conquer approach, as given in (7), has reduced the number of multiplications needed to compute the DFT.

Of course, when taking into account that, even if the outermost sum of (7) is not already in the form of a DFT, it can be rearranged into a DFT plus some so-called twiddle-factors, this mapping is always even more favorable than is shown by (8), especially for small N_1, N_2 (for example, the length-2 DFT is simply a sum and difference).

Obviously, if N is highly composite, the division can be applied again to the subproblems, which results in a number of operations generally several orders of magnitude better than the direct matrix-vector product.

The important point in (2) is that two costs appear explicitly in the divide and conquer scheme: the cost of the mapping (which can be zero when looking at the number of operations only) and the cost of the subproblems. Thus, different types of divide and conquer methods attempt to find various balancing schemes between the mapping and the subproblem costs. In the radix-2 algorithm, for example, the subproblems end up being quite trivial (only sum and differences), while the mapping requires twiddle factors that lead to a large number of multiplications. On the contrary, in the prime factor algorithm, the mapping requires no arithmetic operation (only permutations), while the small DFTs that appear as subproblems will lead to substantial costs since their lengths are coprime.

4. FFTs with twiddle factors

The divide and conquer approach reintroduced by Cooley and Tukey [25] can be used for any composite length N but has the specificity of always introducing twiddle factors. It turns out that when the factors of N are not coprime (for example if $N = 2^n$), these twiddle factors cannot be avoided at all. This section will be devoted to the different algorithms in that class.

The difference between the various algorithms will consist in the fact that more or fewer of these twiddle factors will turn out to be trivial multiplications, such as 1, −1, j, −j.

4.1. *The Cooley-Tukey mapping*

Let us assume that the length of the transform is composite: $N = N_1 \cdot N_2$.

As we have seen in Section 3, we want to partition $\{x_i \mid i = 0, \ldots, N-1\}$ into different subsets $\{x_i \mid i \in I_l\}$ in such a way that the periodicities of the involved subsequences are compatible with the periodicity of the input sequence, on the one hand, and allow to define DFTs of reduced lengths on the other hand.

Hence, it is natural to consider decimated versions of the initial sequence:

$$I_{n_1} = \{n_2 N_1 + n_1\}, \quad n_1 = 0, \ldots, N_1 - 1, \quad n_2 = 0, \ldots, N_2 - 1, \tag{9}$$

which, introduced in (6) gives

$$X(z) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_2 N_1 + n_1} z^{-(n_2 N_1 + n_1)}, \tag{10}$$

and, after normalizing with respect to the first element of each subset,

$$X(z) = \sum_{n_1=0}^{N_1-1} z^{-n_1} \sum_{n_2=0}^{N_2-1} x_{n_2 N_1 + n_1} z^{-n_2 N_1},$$

$$X_k = X(z)\big|_{z = W_N^{-k}} = \sum_{n_1=0}^{N_1-1} W_N^{n_1 k} \sum_{n_2=0}^{N_2-1} x_{n_2 N_1 + n_1} W_N^{n_2 N_1 k}. \tag{11}$$

Using the fact that

$$W_N^{i N_1} = e^{-j 2\pi N_1 i / N} = e^{-j2\pi i / N_2} = W_{N_2}^i, \tag{12}$$

(11) can be rewritten as

$$X_k = \sum_{n_1=0}^{N_1-1} W_N^{n_1 k} \sum_{n_2=0}^{N_2-1} x_{n_2 N_1 + n_1} W_{N_2}^{n_2 k}. \tag{13}$$

Equation (13) is now nearly in its final form, since the right-hand sum corresponds to N_1 DFTs of length N_2, which allows the reduction of arithmetic complexity to be achieved by reiterating the process. Nevertheless, the structure of the Cooley-Tukey FFT is not fully given yet.

Call $Y_{n_1,k}$ the kth output of the n_1th such DFT:

$$Y_{n_1,k} = \sum_{n_2=0}^{N_2-1} x_{n_2 N_1 + n_1} W_{N_2}^{n_2 k}. \tag{14}$$

Note that in $Y_{n_1,k}$, k can be taken modulo N_2, because

$$W_{N_2}^{k} = W_{N_2}^{N_2+k'} = W_{N_2}^{N_2} \cdot W_{N_2}^{k'} = W_{N_2}^{k'}. \tag{15}$$

With this notation, X_k becomes

$$X_k = \sum_{n_1=0}^{N_1-1} Y_{n_1,k} W_N^{n_1 k}. \tag{16}$$

At this point, we can notice that all the X_k for ks being congruent modulo N_2 are obtained from the same group of N_1 outputs of $Y_{n_1,k}$. Thus, we express k as

$$k = k_1 N_2 + k_2$$
$$k_1 = 0, \ldots, N_1 - 1, \quad k_2 = 0, \ldots, N_2 - 1. \tag{17}$$

Obviously, $Y_{n_1,k}$ is equal to Y_{n_1,k_2} since k can be taken modulo N_2 in this case (see (12) and (15)). Thus, we rewrite (16) as

$$X_{k_1 N_2 + k_2} = \sum_{n_1=0}^{N_1-1} Y_{n_1,k_2} W_N^{n_1(k_1 N_2 + k_2)}, \tag{18}$$

which can be reduced, using (12), to

$$X_{k_1 N_2 + k_2} = \sum_{n_1=0}^{N_1-1} Y_{n_1,k_2} W_N^{n_1 k_2} W_{N_1}^{n_1 k_1}. \tag{19}$$

Calling Y'_{n_1,k_2} the result of the first multiplication (by the twiddle factors) in (19) we get

$$Y'_{n_1,k_2} = Y_{n_1,k_2} W_N^{n_1 k_2}. \tag{20}$$

We see that the values of $X_{k_1 N_2 + k_2}$ are obtained from N_2 DFTs of length N_1 applied on Y'_{n_1,k_2}:

$$X_{k_1 N_2 + k_2} = \sum_{n_1=0}^{N_1-1} Y'_{n_1,k_2} W_{N_1}^{n_1 k_1}. \tag{21}$$

We recapitulate the important steps that lead to (21). First, we evaluated N_1 DFTs of length N_2 in (14). Then, N multiplications by the twiddle factors were performed in (20). Finally, N_2 DFTs of length N_1 lead to the final result (21).

A way of looking at the change of variables performed in (9) and (17) is to say that the one-dimensional vector x_i has been mapped into a two-dimensional vector x_{n_1,n_2} having N_1 lines and N_2 columns. The computation of the DFT is then divided into N_1 DFTs on the lines of the vector x_{n_1,n_2}, a point by point multiplication with the twiddle factors and finally N_2 DFTs on the columns of the preceding result.

Until recently, this was the usual presentation of FFT algorithms, by the so-called 'index mappings' [4, 23]. In fact, (9) and (17), taken together, are often referred to as the 'Cooley-Tukey mapping' or 'common factor mapping'. However, the problem with the two-dimensional interpretation is that it does not include all algorithms (like the split-radix algorithm that will be seen later). Thus, while this interpretation helps the understanding of some of the algorithms, it hinders the comprehension of others. In our presentation, we tried to enhance the role of the periodicities of the problem, which result from the initial choice of the subsets.

Nevertheless, we illustrate pictorially a length-15 DFT using the two-dimensional view with $N_1 = 3$, $N_2 = 5$ (see Fig. 1), together with the Cooley-Tukey mapping in Fig. 2, to allow a precise comparison with Good's mapping that leads to the other class of FFTs: the FFTs without twiddle factors. Note that for the case where N_1 and N_2 are coprime, the Good's mapping will be more efficient as shown in the next section, and thus this example is for illustration and comparison purpose only. Because of the twiddle factors in (20), one cannot interchange the order of DFTs once the input mapping

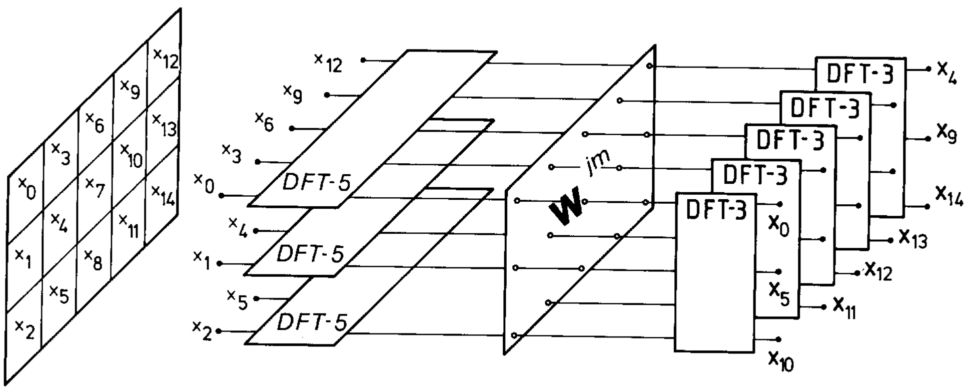


Fig. 1. 2-D view of the length-15 Cooley-Tukey FFT.

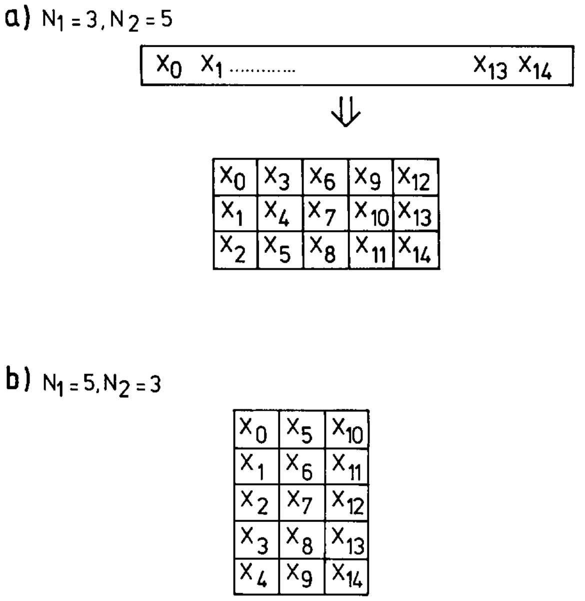


Fig. 2. Cooley-Tukey mapping. (a) $N_1 = 3$, $N_2 = 5$; (b) $N_1 = 5$, $N_2 = 3$.

has been chosen. Thus, in Fig. 2(a), one has to begin with the DFTs on the rows of the matrix. Choosing $N_1 = 5$, $N_2 = 3$ would lead to the matrix of Fig. 2(b), which is obviously different from just transposing the matrix of Fig. 2(a). This shows again that the mapping does not lead to a true two-dimensional transform (in that case, the order of row and column would not have any importance).

4.2. Radix-2 and radix-4 algorithms

The algorithms suited for lengths equal to powers of 2 (or 4) are quite popular since sequences of such lengths are frequent in signal processing (they make full use of the addressing capabilities of computers or DSP systems).

We assume first that $N = 2^n$. Choosing $N_1 = 2$ and $N_2 = 2^{n-1} = N/2$ in (9) and (10) divides the imput sequence into the sequence of even and odd numbered samples, which is the reason why this approach is called 'decimation in time' (DIT). Both sequences are decimated versions, with different phases, of the original sequence. Following (17), the output consists of $N/2$ blocks of 2 values. Actually, in this simple case, it is easy to rewrite (14), (21) exhaustively:

$$X_{k_2} = \sum_{n_2=0}^{N/2-1} x_{2n_2} W_{N/2}^{n_2k_2} + W_N^{k_2} \sum_{n_2=0}^{N/2-1} x_{2n_2+1} W_{N/2}^{n_2k_2}, \tag{22a}$$

$$X_{N/2+k_2} = \sum_{n_2=0}^{N/2-1} x_{2n_2} W_{N/2}^{n_2k_2} - W_N^{k_2} \sum_{n_2=0}^{N/2-1} x_{2n_2+1} W_{N/2}^{n_2k_2}. \tag{22b}$$

Thus, X_m and $X_{N/2+m}$ are obtained by 2-point DTFs on the outputs of the length-$N/2$ DFTs of the even and odd-numbered sequences, one of which is weighted by twiddle factors. The structure

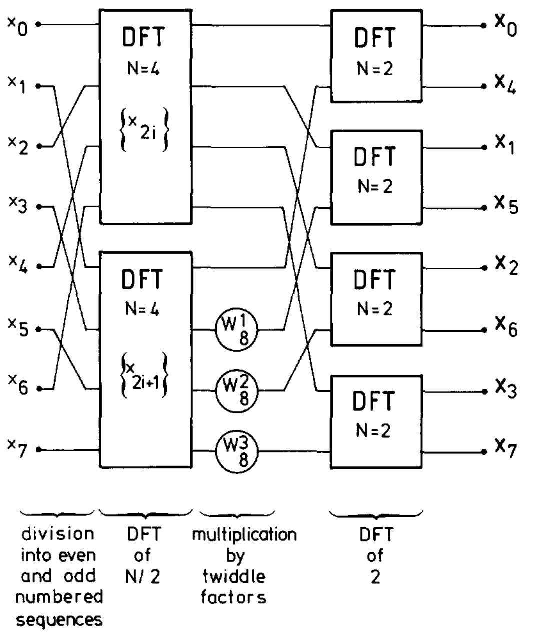


Fig. 3. Decimation in time radix-2 FFT.

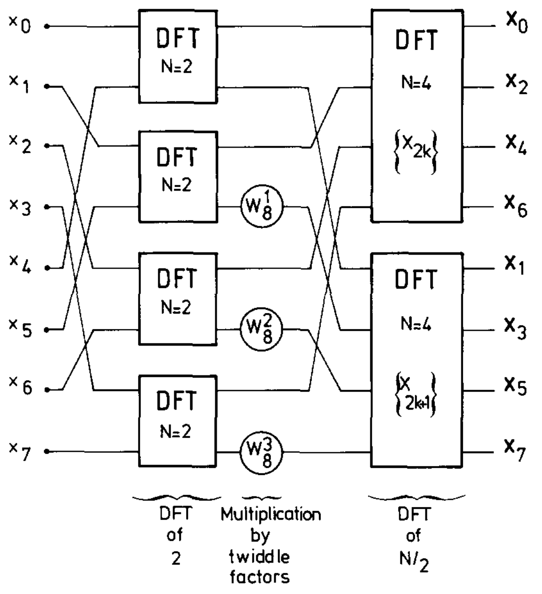


Fig. 4. Decimation in frequency radix-2 FFT.

made by a sum and difference followed (or preceded) by a twiddle factor is generally called a 'butterfly'. The DIT radix-2 algorithm is schematically shown in Fig. 3.

Its implementation can now be done in several different ways. The most natural one is to reorder the input data such that the samples of which the DFT has to be taken lie in subsequent locations. This results in the bit-reversed input, in-order output decimation in time algorithm. Another possibility is to selectively compute the DFTs over the input sequence (taking only the even and odd numbered samples), and perform an in-place computation. The output will now be in bit-reversed order. Other implementation schemes can lead to constant permutations between the stages (constant geometry algorithm [15]).

If we reverse the role of N_1 and N_2, we get the decimation in frequency (DIF) version of the algorithm. Inserting $N_1 = N/2$ and $N_2 = 2$ into (9), (10) leads to (again from (14) and (21))

$$X_{2k_1} = \sum_{n_1=0}^{N/2-1} W_{N/2}^{n_1k_1}(x_{n_1} + x_{N/2+n_1}), \tag{23a}$$

$$X_{2k_1+1} = \sum_{n_1=0}^{N/2-1} W_{N/2}^{n_1k_1} W_N^{n_1}(x_{n_1} - x_{N/2+n_1}). \tag{23b}$$

This first step of a DIF algorithm is represented in Fig. 5(a), while a schematic representation of the full DIF algorithm is given in Fig. 4. The duality between division in time and division in frequency is obvious, since one can be obtained from the other by interchanging the role of $\{x_i\}$ and $\{X_k\}$.

Let us now consider the computational complexity of the radix-2 algorithm (which is the same for the DIF and DIT version because of the duality indicated above). From (22) or (23), one sees that a DFT of length N has been replaced by two DFTs of length $N/2$, and this at the cost of $N/2$ complex multiplications as well as N complex additions. Iterating the scheme $\log_2 N - 1$ times in order to obtain trivial transforms (of length 2) leads to the following order of magnitude of the number of operations:

$$\mathrm{O_M}[\mathrm{DFT}_{\text{radix-2}}] \approx N/2(\log_2 N - 1)$$

complex multiplications, (24a)

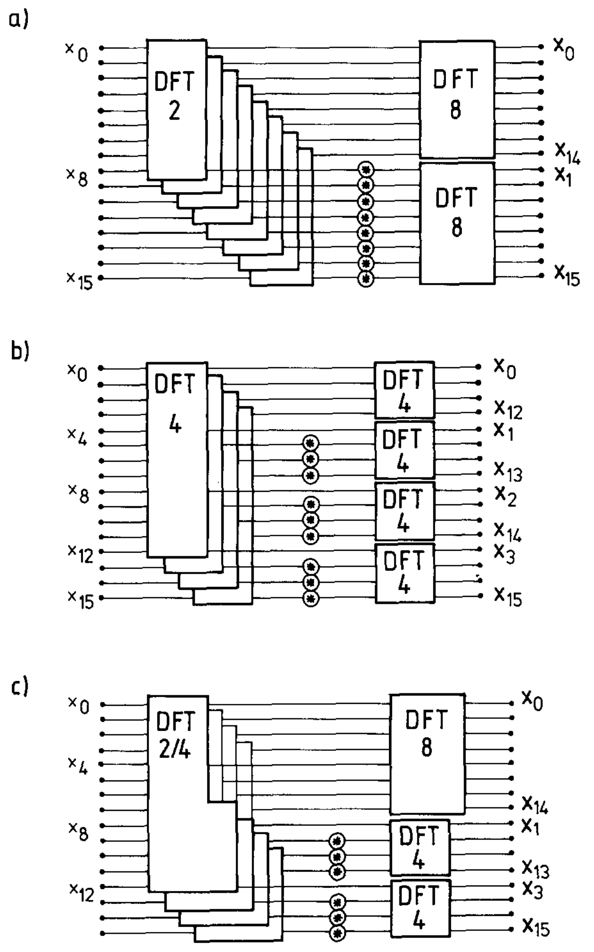


Fig. 5. Comparison of various DIF algorithms for the length-16 DFT. (a) Radix-2; (b) radix-4; (c) split-radix.

$$O_A[\mathrm{DFT}_{\text{radix-2}}] \approx N(\log_2 N - 1) \quad \text{complex additions.} \tag{24b}$$

A closer look at the twiddle factors will enable us to still reduce these numbers. For comparison purposes, we will count the number of real operations that are required, provided that the multiplication of a complex number x by W_N^i is done using 3 real multiplications and 3 real additions [12]. Furthermore, if i is a multiple of $N/4$, no arithmetic operation is required, and only 2 real multiplications and additions are required if i is an odd multiple of $N/8$. Taking into account these simplifications results in the following total number of operations [12]:

$$M[\mathrm{DFT}_{\text{radix-2}}] = 3N/2 \log_2 N - 5N + 8, \tag{25a}$$

$$A[\mathrm{DFT}_{\text{radix-2}}] = 7N/2 \log_2 N - 5N + 8. \tag{25b}$$

Nevertheless, it should be noticed that these numbers are obtained by the implementation of 4 different butterflies (1 general plus 3 special cases), which reduces the regularity of the programs. An evaluation of the number of real operations for other number of special butterflies, is given in [4], together with the number of operations obtained with the usual 4-mult, 2-adds complex multiplication algorithm.

Another case of interest appears when N is a power of 4. Taking $N_1 = 4$ and $N_2 = N/4$, (13) reduces the length-N DFT into 4 DFTs of length $N/4$, about $3N/4$ multiplications by twiddle factors, and $N/4$ DFTs of length 4. The interest of this case lies in the fact that the length-4 DFTs do not cost any multiplication (only 16 real additions). Since there are $\log_4 N - 1$ stages and the first set of twiddle factors (corresponding to $n_1 = 0$ in (20)) is trivial, the number of complex multiplications is about

$$O_M[\mathrm{DFT}_{\text{radix-4}}] \approx 3N/4(\log_4 N - 1). \tag{26}$$

Comparing (26) to (24a) shows that the number of multiplications can be reduced with this radix-4 approach by about a factor of 3/4. Actually, a detailed operation count using the simplifications indicated above gives the following result [12]:

$$M[\mathrm{DFT}_{\text{radix-4}}] = 9N/8 \log_2 N - 43N/12 + 16/3, \tag{27a}$$

$$A[\mathrm{DFT}_{\text{radix-4}}] = 25N/8 \log_2 N - 43N/12 + 16/3. \tag{27b}$$

Nevertheless, these operation counts are obtained at the cost of using six different butterflies in the programming of the FFT. Slight additional gains can be obtained when going to even higher radices (like 8 or 16) and using the best possible algorithms for the small DFTs. Since programs with a regular structure are generally more compact, one often uses recursively the same decomposition at each stage, thus leading to full radix-2 or radix-4 programs, but when the length is not a power of the radix (for example 128 for a radix-4 algorithm), one can use smaller radices towards the end of the decomposition. A length-256 DFT could use 2 stages of radix-8 decomposition, and finish with one stage of radix-4. This approach is called 'mixed-radix' approach [45] and achieves low arithmetic complexity while allowing flexible transform length (not restricted to powers of 2, for example), at the cost of a more involved implementation.

4.3. Split-radix algorithm

As already noted in Section 2, the lowest known number of both multiplications and additions for length-2^n algorithms was obtained as early as 1968 and was again achieved recently by new algorithms. Their power was to show explicitly that the improvement over fixed- or mixed-radix algorithms can be obtained by using a radix-2 and a radix-4 simultaneously on different parts of the transform. This allowed the emergence of new compact and computationally efficient programs to compute the length-2^n DFT.

Below, we will try to motivate (a posteriori!) the split-radix approach and give the derivation of the algorithm as well as its computational complexity.

When looking at the DIF radix-2 algorithm given in (23), one notices immediately that the even indexed outputs X_{2k_1} are obtained without any further multiplicative cost from the DFT of a length-$N/2$ sequence, which is not so well-done in the radix-4 algorithm for example, since relative to that length-$N/2$ sequence, the radix-4 behaves like a radix-2 algorithm. This lacks logical sense, because it is well-known that the radix-4 is better than the radix-2 approach.

From that observation, one can derive a first rule: the even samples of a DIF decomposition X_{2k} should be computed separately from the other ones, with the same algorithm (recursively) as the DFT of the original sequence (see [53] for more details).

However, as far as the odd indexed outputs X_{2k+1} are concerned, no general simple rule can be established, except that a radix-4 will be more efficient than a radix-2, since it allows to compute the samples through two $N/4$ DFTs instead of a single $N/2$ DFT for a radix-2, and this at the same multiplicative cost, which will allow the cost of the recursions to grow more slowly. Tests showed that computing the odd indexed output through radices higher than 4 was inefficient.

The first recursion of the corresponding 'split-radix' algorithm (the radix is split in two parts) is obtained by modifying (23) accordingly:

$$X_{2k_1}=\sum_{n_1=0}^{N/2-1} W_{N/2}^{n_1k_1}(x_{n_1}+x_{N/2+n_1}), \tag{28a}$$

$$X_{4k_1+1}=\sum_{n_1=0}^{N/4-1} W_{N/4}^{n_1k_1}W_N^{n_1} \times[(x_{n_1}-x_{N/2+n_1}) +\mathrm{j}(x_{n_1+N/4}-x_{n_1+3N/4})], \tag{28b}$$

$$X_{4k_1+3}=\sum_{n_1=0}^{N/4-1} W_{N/4}^{n_1k_1}W_N^{3n_1} \times[(x_{n_1}+x_{N/2+n_1}) -\mathrm{j}(x_{n_1+N/4}-x_{n_1+3N/4})]. \tag{28c}$$

The above approach is a DIF SRFFT, and is compared in Fig. 5 with the radix-2 and radix-4 algorithms. The corresponding DIT version, being dual, considers separately the subsets $\{x_{2i}\}$, $\{x_{4i+1}\}$ and $\{x_{4i+3}\}$ of the initial sequence.

Taking $I_0=\{2i\}$, $I_1=\{4i+1\}$, $I_2=\{4i+3\}$ and normalizing with respect to the first element of the set in (7) leads to

$$X_k=\sum_{I_0} x_{2i}W_N^{k(2i)}+W_N^k\sum_{I_1} x_{4i+1}W_N^{k(4i+1)-k} + W_N^{3k}\sum_{I_2} x_{4i+3}W_N^{k(4i+3)-3k}, \tag{29}$$

which can be explicitly decomposed in order to make the redundancy between the computation of

X_k, $X_{k+N/4}$, $X_{k+N/2}$ and $X_{k+3N/4}$ more apparent:

$$X_k = \sum_{i=0}^{N/2-1} x_{2i} W_{N/2}^{ik} + W_N^k \sum_{i=0}^{N/4-1} x_{4i+1} W_{N/4}^{ik} + W_N^{3k} \sum_{i=0}^{N/4-1} x_{4i+3} W_{N/4}^{ik}, \tag{30a}$$

$$X_{k+N/4} = \sum_{i=0}^{N/2-1} x_{2i} W_{N/2}^{ik} + \mathrm{j} W_N^k \sum_{i=0}^{N/4-1} x_{4i+1} W_{N/4}^{ik} - \mathrm{j} W_N^{3k} \sum_{i=0}^{N/4-1} x_{4i+3} W_{N/4}^{ik}, \tag{30b}$$

$$X_{k+N/2} = \sum_{i=0}^{N/2-1} x_{2i} W_{N/2}^{ik} - W_N^k \sum_{i=0}^{N/4-1} x_{4i+1} W_{N/4}^{ik} - W_N^{3k} \sum_{i=0}^{N/4-1} x_{4i+3} W_{N/4}^{ik}, \tag{30c}$$

$$X_{k+3N/4} = \sum_{i=0}^{N/2-1} x_{2i} W_{N/2}^{ik} - \mathrm{j} W_N^k \sum_{i=0}^{N/4-1} x_{4i+1} W_{N/4}^{ik} + \mathrm{j} W_N^{3k} \sum_{i=0}^{N/4-1} x_{4i+3} W_{N/4}^{ik}. \tag{30d}$$

The resulting algorithms have the minimum known number of operations (multiplications plus additions) as well as the minimum number of multiplications among practical algorithms for lengths which are powers of 2. The number of operations can be checked as being equal to

$$M[\mathrm{DFT}_{\text{split-radix}}] = N \log_2 N - 3N + 4, \tag{31a}$$

$$A[\mathrm{DFT}_{\text{split-radix}}] = 3N \log_2 N - 3N + 4. \tag{31b}$$

These numbers of operations can be obtained with only 4 different building blocks (with a complexity slightly lower than the one of a radix-4 butterfly), and are compared with the other algorithms in Tables 1 and 2.

Of course, due to the asymmetry in the decomposition, the structure of the algorithm is slightly more involved than for fixed-radix algorithms. Nevertheless, the resulting programs remain fairly simple [113] and can be highly optimized. Furthermore, this approach is well suited for applying FFTs on real data. It allows an in-place, butterfly style implementation to be performed [65, 77].

The power of this algorithm comes from the fact that it provides the lowest known number of operations for computing length-2^n FFTs, while

Table 1

Number of non trivial real multiplications for various FFTs on complex data

N	Radix 2	Radix 4	SRFFT	PFA	Winograd
16	24	20	20		
30				100	68
32	88		68		
60				200	136
64	264	208	196		
120				460	276
128	712		516		
240				1100	632
256	1800	1392	1284		
504				2524	1572
512	4360		3076		
1008				5804	3548
1024	10248	7856	7172		
2048	23560		16388		
2520				17660	9492

Table 2

Number of real additions for various FFTs on complex data

N		Radix 2	Radix 4	SRFFT	PFA	Winograd
16		152	148	148		
	30				384	384
32		408		388		
	60				888	888
64		1032	976	964		
	120				2076	2076
128		2504		2308		
	240				4812	5016
256		5896	5488	5380		
	504				13388	14540
512		13566		12292		
	1008				29548	34668
1024		30728	28336	27652		
2048		68616		61444		
	2520				84076	99628

being implemented with compact programs. We shall see later that there are some arguments tending to show that it is actually the best possible compromise.

Note that the number of multiplications in (31a) is equal to the one obtained with the so-called 'real-factor' algorithms [44, 24]. In that approach, a linear combination of the data, using additions only, is made such that all twiddle factors are either pure real or pure imaginary. Thus, a multiplication of a complex number by a twiddle factor requires only 2 real multiplications. However, the real factor algorithms are quite costly in terms of additions, and are numerically ill-conditioned (division by small constants).

4.4. *Remarks on FFTs with twiddle factors*

The Cooley-Tukey mapping in (9) and (17) is generally applicable, and actually the only possible mapping when the factors on N are not coprime. While we have paid particular attention to the case $N=2^n$, similar algorithms exist for $N=p^m$ (p an arbitrary prime). However, one of the elegances of the length-2^n algorithms comes from the fact that the small DFTs (lengths 2 and 4) are multiplication-free, a fact that does not hold for other radices like 3 or 5, for instance. Note, however, that it is possible, for radix-3, either to completely remove the multiplication inside the butterfly by a change of base [26], at the cost of a few multiplications and additions, or to merge it with the twiddle factor [49] in the case where the implementation is based on the 4-mult 2-add complex multiplication scheme. It was also recently shown that, as soon as a radix p^2 algorithm was more efficient than a radix-2 algorithm, a split-radix p/p^2 was more efficient than both of them [53]. However, unlike the 2^n case, efficient implementations for these p^n split-radix algorithms have not yet been reported. More efficient mixed radix algorithms also remain to be found (initial results are given in [40]).

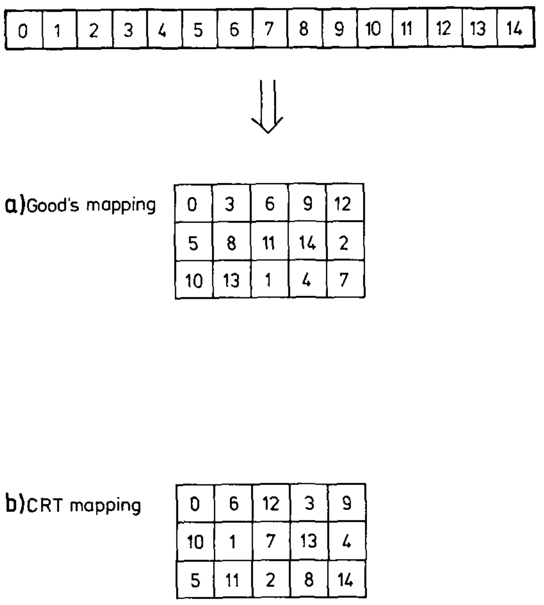
5. FFTs based on costless mono- to multidimensional mapping

The divide and conquer strategy, as explained in Section 3, has few requirements for feasibility: N needs only to be composite, and the whole DFT is computed from DFTs on a number of points which is a factor of N (this is required for the redundancy in the computation of (11) to be apparent). This requirement allows the expression of the innermost sum of (11) as a DFT, provided that the subsets I_l have been chosen in such a way that x_i, $i \in I_l$ is periodic. But, when N factors into relatively prime factors, say $N = N_1 \cdot N_2$, $(N_1, N_2) = 1$, a very simple property will allow a stronger requirement to be fulfilled:

Starting from any point of the sequence x_i, you can take as a first subset with compatible periodicity either $\{x_{i+N_1 \cdot n_2} | n_2 = 1, \ldots, N_2 - 1\}$ or, equivalently $\{x_{i+N_2 \cdot n_1} | n_1 = 1, \ldots, N_1 - 1\}$, and both subsets only have one common point x_i (by compatible, it is meant that the periodicity of the subsets divides the periodicity of the set). This allows a rearrangement of the input (periodic) vector into a matrix with a periodicity in both dimensions (rows and columns), both periodicities being compatible with the initial one (see Fig. 6).

5.1. Basic tools

FFTs without twiddle factors are all based on the same mapping, which is explained in Section 5.1.1. This mapping turns the original transform into sets of small DFTs, the lengths of which are coprime. It is therefore necessary to find efficient ways of computing these short-length DFTs. Section 5.1.2 explains how to turn them into cyclic convolutions for which efficient algorithms are described in Section 5.1.3.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

a) Good's mapping

0	3	6	9	12
5	8	11	14	2
10	13	1	4	7

b) CRT mapping

0	6	12	3	9
10	1	7	13	4
5	11	2	8	14

Fig. 6. The prime factor mappings for $N = 15$.

5.1.1. *The mapping of Good* [32]

Performing the selection of subsets described in the introduction of Section 5 for any index i is equivalent to writing i as

$$\begin{aligned} &i = \langle n_1 \cdot N_2 + n_2 \cdot N_1 \rangle_N, \\ &\quad n_1 = 1, \ldots, N_1 - 1, \quad n_2 = 1, \ldots, N_2 - 1, \\ &\quad N = N_1 N_2, \end{aligned} \tag{32}$$

and, since N_1 and N_2 are coprime, this mapping is easily seen to be one to one. (It is obvious from the right-hand side of (32) that all congruences modulo N_1 are obtained for a given congruence modulo N_2, and vice versa.)

This mapping is another arrangement of the 'Chinese Remainder Theorem' mapping, which can be explained as follows on index k.

The Chinese Remainder Theorem (CRT) states that if we know the residue of some number k modulo two relatively prime numbers N_1 and N_2, it is possible to reconstruct $\langle k \rangle_{N_1 N_2}$ as follows:

Let $\langle k \rangle_{N_1} = k_1$ and $\langle k \rangle_{N_2} = k_2$. Then the value of $k \bmod N$ $(N = N_1 \cdot N_2)$ can be found by

$$k = \langle N_1 t_1 k_2 + N_2 t_2 k_1 \rangle_N, \tag{33}$$

t_1 being the multiplicative inverse of N_1 mod N_2, that is $\langle t_1, N_1 \rangle_{N_2} = 1$, and t_2 the multiplicative inverse of N_2 mod N_1 (these inverses always exist, since N_1 and N_2 are coprime: $(N_1, N_2) = 1$).

Taking into account these two mappings in the definition of the DFT (3) leads to

$$X_{N_1 t_1 k_2 + N_2 t_2 k_1} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1 N_2 + n_2 N_1} W_N^{(n_1 N_2 + N_1 n_2)(N_1 t_1 k_2 + N_2 t_2 k_1)}, \tag{35}$$

but

$$W_N^{N_2} = W_{N_1} \tag{36}$$

and

$$W_{N_1}^{N_2 t_2} = W_{N_1}^{\langle N_2 t_2 \rangle_{N_1}} = W_{N_1}, \tag{37}$$

which implies

$$X_{N_1 t_1 k_2 + N_2 t_2 k_1} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1 N_2 + n_2 N_1} \times W_{N_1}^{n_1 k_2} W_{N_2}^{n_2 k_2}, \tag{38}$$

which, with

$$x'_{n_1,n_2} = x_{n_1 N_2 + n_2 N_1}$$

and

$$X'_{k_1,k_2} = X_{N_1 t_1 k_2 + N_2 t_2 k_1},$$

leads to a formulation of the initial DFT into a true bidimensional transform:

$$X'_{k_1 k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x'_{n_1 n_2} W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2}. \tag{39}$$

An illustration of the prime factor mapping is given in Fig. 6(a) for the length $N = 15 = 3 \cdot 5$, and Fig. 6(b) provides the CRT mapping. Note that these mappings, which were provided for a factorization of N into two coprime numbers easily generalizes to more factors, and that reversing the roles of N_1 and N_2 results in a transposition of the matrices of Fig. 6.

5.1.2. DFT computation as a convolution

With the aid of Good's mapping, the DFT computation is now reduced to that of a multidimensional DFT, with the characteristic that the lengths along each dimension are coprime. Furthermore, supposing that these lengths are small is quite reasonable, since Good's mapping can provide a full multi-dimensional factorization when N is highly composite.

The question is now to find the best way of computing this M-D DFT and these small-length DFTs. A first step in that direction was obtained by Rader [43] who showed that a DFT of prime length could be obtained as the result of a cyclic convolution: Let us rewrite (1) for a prime length $N = 5$:

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W_5^1 & W_5^2 & W_5^3 & W_5^4 \\ 1 & W_5^2 & W_5^4 & W_5^1 & W_5^3 \\ 1 & W_5^3 & W_5^1 & W_5^4 & W_5^2 \\ 1 & W_5^4 & W_5^3 & W_5^2 & W_5^1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}. \tag{40}$$

Obviously, removing the first column and first row of the matrix will not change the problem, since they do not involve any multiplication. Furthermore, careful examination of the remaining part of the matrix shows that each column and each row involves every possible power of W_5, which is the first condition to be met for this part of the DFT to become a cyclic convolution. Let us now permute the last two rows and last two columns of the reduced matrix:

$$\begin{bmatrix} X'_1 \\ X'_2 \\ X'_4 \\ X'_3 \end{bmatrix} = \begin{bmatrix} W_5^1 & W_5^2 & W_5^4 & W_5^3 \\ W_5^2 & W_5^4 & W_5^3 & W_5^1 \\ W_5^4 & W_5^3 & W_5^1 & W_5^2 \\ W_5^3 & W_5^1 & W_5^2 & W_5^4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_4 \\ x_3 \end{bmatrix}. \tag{41}$$

Equation (41) is then a cyclic correlation (or a convolution with the reversed sequence).

It turns out that this a general result.

It is well-known in number theory that the set of numbers lower than a prime p admits some primitive elements g such that the successive powers of g modulo p generate all the elements of the set. In the example above, $p = 5$, $g = 2$, and we observe that

$$g^0 = 1, \quad g^1 = 2, \quad g^2 = 4, \quad g^3 = 8 = 3 \pmod 5).$$

The above result (41) is only the writing of the DFT in terms of the successive powers of w_p^g:

$$X'_k = \sum_{i=1}^{p-1} x_i W_p^{ik}, \quad k = 1, \ldots, p-1, \tag{42}$$

$$\langle ik \rangle_p = \langle \langle i \rangle_p \cdot \langle k \rangle_p \rangle_p = \langle \langle g^{u_i} \rangle_p \langle g^{v_k} \rangle_p \rangle_p,$$

$$X'_{g^{v_i}} = \sum_{u_i=0}^{p-2} x_{g^{u_i}} \cdot (W_p^g)^{u_i + v_i}, \quad v_i = 0, \ldots, p-2, \tag{43}$$

and the length-p DFT turns out to be a length $(p-1)$ cyclic correlation:

$$\{X'_g\} = \{x_g\} * \{W_p^g\}. \tag{44}$$

5.1.3. Computation of the cyclic convolution

Of course (43) has changed the problem, but it is not solved yet. And in fact, Rader's result was considered as a curiosity up to the moment when Winograd [55] obtained some new results on the computation of cyclic convolution.

And, again, this was obtained by application of the CRT. In fact, the CRT, as explained in (33), (34) can be rewritten in the polynomial domain: if we know the residues of some polynomial $K(z)$ modulo two mutually prime polynomials

$$\begin{aligned}\langle K(z)\rangle_{P_1(z)} &= K_1(z),\\ \langle K(z)\rangle_{P_2(z)} &= K_2(z),\end{aligned} \quad (P_1(z), P_2(z)) = 1, \tag{45}$$

we shall be able to obtain

$$K(z) \bmod P_1(z) \cdot P_2(z) = P(z)$$

by a procedure similar to that of (33).

This fact will be used twice in order to obtain Winograd's method of computing cyclic convolutions:

A first application of the CRT is the breaking of the cyclic convolution into a set of polynomial products. For more convenience, let us first state (44) in polynomial notation:

$$X'(z) = x'(z) \cdot w(z) \bmod (z^{p-1}-1). \tag{46}$$

Now, since $p-1$ is not prime (it is at least even), $z^{p-1}-1$ can be factorized at least as

$$z^{p-1}-1 = (z^{(p-1)/2}+1)(z^{(p-1)/2}-1), \tag{47}$$

and possibly further, depending on the value of p. These polynomial factors are known and named cyclotomic polynomials $\varphi_q(z)$. They provide the full factorization of any z^N-1:

$$z^N - 1 = \prod_{q|N} \varphi_q(z). \tag{48}$$

A useful property of these cyclotomic polynomials is that the roots of $\varphi_q(z)$ are all the qth primitive roots of unity, hence degree $\{\varphi_q(z)\} = \varphi(q)$, which is by definition the number of integers lower than q and coprime with it. Namely, if $w_q = e^{-j2\pi/q}$, the roots of $\varphi_q(z)$ are $\{W_q^r \mid (r, q) = 1\}$.

As an example, for $p=5$, $z^{p-1}-1 = z^4-1$,

$$\begin{aligned}z^4-1 &= \varphi_1(z)\cdot\varphi_2(z)\cdot\varphi_4(z)\\ &= (z-1)(z+1)(z^2+1).\end{aligned}$$

The first use of the CRT to compute the cyclic convolution (46) is then as follows:

(1) compute $\left.\begin{aligned}x'_q(z) &= x'(z) \bmod \varphi_q(z),\\ w'_q(z) &= w(z) \bmod \varphi_q(z),\end{aligned}\right\} \quad q \mid p-1,$

(2) then obtain

$$X'_q(z) = x'_q(z)\cdot w'_q(z) \bmod \varphi_q(z)$$

(3) and reconstruct $X'(z) \bmod z^{p-1}-1$ from the polynomials $X'_q(z)$ using the CRT.

Let us apply this procedure to our simple example:

$$x'(z) = x_1 + x_2 z + x_4 z^2 + x_3 z^3,$$

$$w(z) = W_5^1 + W_5^2 z + W_5^4 z^2 + W_5^3 z^3.$$

Step 1.

$$\begin{aligned}w_4(z) &= w(z) \bmod \varphi_4(z)\\ &= (W_5^1 - W_5^4) + (W_5^2 - W_5^3)z,\\ w_2(z) &= w(z) \bmod \varphi_2(z)\\ &= (W_5^1 + W_5^4 - W_5^2 - W_5^3),\\ w_1(z) &= w(z) \bmod \varphi_1(z)\\ &= (W_5^1 + W_5^4 + W_5^2 + W_5^3) \quad [=-1],\end{aligned}$$

$$\begin{aligned}x'_4(z) &= (x_1 - x_4) + (x_2 - x_3)z,\\ x'_2(z) &= (x_1 + x_4 - x_2 - x_3),\\ x'_1(z) &= (x_1 + x_4 + x_2 + x_3).\end{aligned}$$

Step 2.

$$\begin{aligned}X'_4(z) &= x'_4(z)\cdot w_4(z) \bmod \varphi_4(z),\\ X'_2(z) &= x'_2(z)\cdot w_2(z) \bmod \varphi_2(z),\\ X'_1(z) &= x'_1(z)\cdot w_1(z) \bmod \varphi_1(z).\end{aligned}$$

Step 3.

$$X'(z)=[X_1'(z)(1+z)/2+X_2'(z)(1-z)/2]$$
$$\times(1+z^2)/2+X_4'(z)(1-z^2)/2.$$

Note that all the coefficients of $w_q(z)$ are either real or purely imaginary. This is a general property due to the symmetries of the successive powers of W_p.

The only missing tool needed to complete the procedure now is the algorithm to compute the polynomial products modulo the cyclotomic factors. Of course, a straightforward polynomial product followed by a reduction modulo $\varphi_q(z)$ would be applicable, but a much more efficient algorithm can be obtained by a second application of the CRT in the field of polynomials.

It is already well-known that knowing the values of an Nth degree polynomial at $N+1$ different points can provide the value of the same polynomial anywhere else by Lagrange interpolation. The CRT provides an analogous way of obtaining its coefficients.

Let us first recall the equation to be solved:

$$X_q'(z)=x_q'(z)\cdot w_q(z) \bmod \varphi_q(z), \tag{49}$$

with

$$\deg \varphi_q(z)=\varphi(q).$$

Since $\varphi_q(z)$ is irreducible, the CRT cannot be used directly. Instead, we choose to evaluate the product $X_q''(z)=x_q'(z)\cdot w_q(z)$ modulo an auxiliary polynomial $A(z)$ of degree greater than the degree of the product. This auxiliary polynomial will be chosen to be fully factorizable. The CRT hence applies, providing

$$X_q''(z)=x_q'(z)\cdot w_q(z),$$

since the mod $A(z)$ is totally artificial, and the reduction modulo $\varphi_q(z)$ will be performed afterwards.

The procedure is then as follows.

Let us evaluate both $x_q'(z)$ and $w_q(z)$ modulo a number of different monomials of the form

$$(z-a_i), \quad i=1,\dots,2\varphi(q)-1.$$

Then compute

$$X_q''(a_i)=x_q^i(a_i)w_q(a_i), \quad i=1,\dots,2\varphi(q)-1. \tag{50}$$

The CRT then provides a way of obtaining

$$X_q''(z) \bmod A(z), \tag{51}$$

with

$$A(z)=\prod_{i=1}^{2\varphi(q)-1}(z-a_i),$$

which is equal to $X_q''(z)$ itself, since

$$\deg X_q''(z)=2\varphi(q)-2. \tag{52}$$

Reduction of $X_q''(z)$ mod $\varphi_z(z)$ will then provide the desired result.

In practical cases, the points $\{a_i\}$ will be chosen in such a way that the evaluation of $w_q'(a_i)$ involves only additions {i.e.: $a_i=0,\pm1,\dots$).

This limits the degree of the polynomials whose products can be computed by this method. Other suboptimal methods exist [12], but are nevertheless based on the same kind of approach (the 'dot products' (50) become polynomial products of lower degree, but the overall structure remains identical).

All this seems fairly complicated, but results in extremely efficient algorithms that have a low number of operations. The full derivation of our example ($p=5$) then provides the following algorithm:

5 point DFT:

$u=2\pi/5$,

(reduction modulo z^2-1:)

$t_1=x_1+x_4, \quad t_2=x_2+x_3,$

(reduction modulo z^2+1:)

$t_3=x_1-x_4, \quad t_4=x_3-x_2,$

$t_5=t_1+t_2$ (reduction modulo $z-1$),

$t_6=t_1-t_2$ (reduction modulo $z+1$),

($X_1'(z)=x_1'(z)\cdot w_1(z) \bmod \varphi_1(z)$:)

$m_1=[(\cos u+\cos 2u)/2]t_5,$

($X_2'(z)=x_2'(z)\cdot w_2(z) \bmod \varphi_2(z)$:)

$m_2 = [(\cos u - \cos 2u)/2]t_6,$

polynomial product modulo z^2+1,

$X'_4(z) = x'_4(zj) \cdot w_4(z) \bmod \varphi_u(z)$:)

$m_3 = -\mathrm{j}(\sin u)(t_3 + t_4),$

$m_4 = -\mathrm{j}(\sin u + \sin 2u)t_4,$

$m_5 = \mathrm{j}(\sin u - \sin 2u)t_3,$

$s_1 = m_3 - m_4,$

$s_2 = m_3 + m_5,$

(reconstruction following Step 3, the 1/2 terms have been included into the polynomial products:)

$s_3 = x_0 + m_1,$

$s_4 = s_3 + m_2,$

$s_5 = s_3 - m_2,$

$X_0 = x_0 + t_5,$

$X_1 = s_4 + s_1,$

$X_2 = s_5 + s_2,$

$X_3 = s_5 - s_2,$

$X_4 = s_4 - s_1.$

When applied to complex data, this algorithm requires 10 real multiplications and 34 real additions, vs. 48 real multiplications and 88 real additions for a straightforward algorithm (matrix-vector product).

In matrix form, and slightly changed, this algorithm may be written as follows:

$$(X'_0, X'_1, \ldots, X'_4)^{\mathrm{T}} = C \cdot D \cdot B \cdot (x_0, x_1, \ldots, x_4)^{\mathrm{T}}, \tag{53}$$

with

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & -1 & 0 \\ 1 & 1 & -1 & 1 & 0 & 1 \\ 1 & 1 & -1 & -1 & 0 & -1 \\ 1 & 1 & 1 & -1 & 1 & 0 \end{bmatrix},$$

$$D = \mathrm{diag}[1, ((\cos u + \cos 2u)/2 - 1), (\cos u - \cos 2u)/2, -\mathrm{j}\sin u, -\mathrm{j}(\sin u + \sin 2u), \mathrm{j}(\sin u - \sin 2u)],$$

$$B = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & -1 & 1 & -1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

By construction, D is a diagonal matrix, where all multiplications are grouped, while C and B only involve additions (they correspond to the reductions and reconstructions in the applications of the CRT).

It is easily seen that this structure is a general property of the short-length DFTs based on CRT: all multiplications are 'nested' at the center of the algorithms. By construction, also, D has dimension M_p, which is the number of multiplications required for computing the DFT, some of them being trivial (at least one, needed for the computation of X_0). In fact, using such a formulation, we have $M_p \geqslant p$. This notation looks awkward, at first glance (why include trivial multiplications in the total number?), but Section 5.3 will show that it is necessary in order to evaluate the number of multiplications in the Winograd FFT.

It can also be proven that the methods explained in this section are essentially the only ways of obtaining FFTs with the minimum number of multiplications. In fact, this gives the optimum structure, mathematically speaking. These methods always provide a number of multiplications lower than twice the length of the DFT:

$$M_{N_1} < 2N_1.$$

This shows the linear complexity of the DFT in this case.

5.2. Prime factor algorithms [95]

Let us now come back to the initial problem of this section: the computation of the bidimensional transform given in (39). Rearranging the data in matrix form, of size N_1N_2, and F_1 (resp. F_2) denoting the Fourier matrix of size N_1 (resp. N_2), results in the following notation, often used in the context of image processing:

$$X = F_1 x F_2^{\mathrm{T}}. \tag{54}$$

Performing the FFT algorithm separately along each dimension results in the so-called prime factor algorithm (PFA).

To summarize, PFA makes use of Good's mapping (Section 5.1.1) to convert the length $N_1 \cdot N_2$ 1-*D* DFT into a size $N_1 \times N_2$ 2-D DFT, and then computes this 2-D DFT in a row-column fashion, using the most efficient algorithms along each dimension.

Of course, this applies recursively to more than two factors, the constraints being that they must be mutually coprime. Nevertheless, this constraint implies the availability of a whole set of efficient small DFTs ($N_i = 2, 3, 4, 5, 7, 8, 16$ is already sufficient to provide a dense set of feasible lengths).

A graphical display of PFA for length $N = 15$ is given in Fig. 7. Since there are N_2 applications of length N_1 FFT and N_1 applications of length N_2 FFTs, the computational costs are as follows:

$$\begin{aligned} M_{N_1N_2} &= N_1M_2 + N_2M_1, \\ A_{N_1N_2} &= N_1A_2 + N_2A_1, \end{aligned} \tag{55}$$

or, equivalently, the number of operations to be performed per output point is the sum of the individual number of operations in each short algorithm: let m_N and a_N be these reduced numbers

$$\begin{aligned} m_{N_1N_2N_3N_4} &= m_{N_1} + m_{N_2} + m_{N_3} + m_{N_4}, \\ a_{N_1N_2N_3N_4} &= a_{N_1} + a_{N_2} + a_{N_3} + a_{N_4}. \end{aligned} \tag{56}$$

An evaluation of these figures is provided in Tables 1 and 2.

5.3. *Winograd's Fourier transform algorithm (WFTA)* [56]

Winograd's FFT makes full use of all the tools explained in Section 5.1.

Good's mapping is used to convert the length $N_1 \cdot N_2$ 1-D DFT into a length $N_1 \times N_2$ 2-D DFT, and the intimate structure of the small-length algorithms is used to nest all the multiplications at the center of the overall algorithm as follows.

Reporting (53) into (54) results in

$$X = C_1 D_1 B_1 x B_2^{\mathrm{T}} D_2 C_2^{\mathrm{T}}. \tag{57}$$

Since C and B do not involve any multiplication, the matrix $(B_1 x B_2^{\mathrm{T}})$ is obtained by only adding properly chosen input elements. The resulting matrix now has to be multiplied on the left and on the right by diagonal matrices D_1 and D_2, of respective dimensions M_1 and M_2. Let M_1' and M_2' be the numbers of trivial multiplications involved.

Premultiplying by the diagonal matrix D_1 multiplies each row by some constant, while

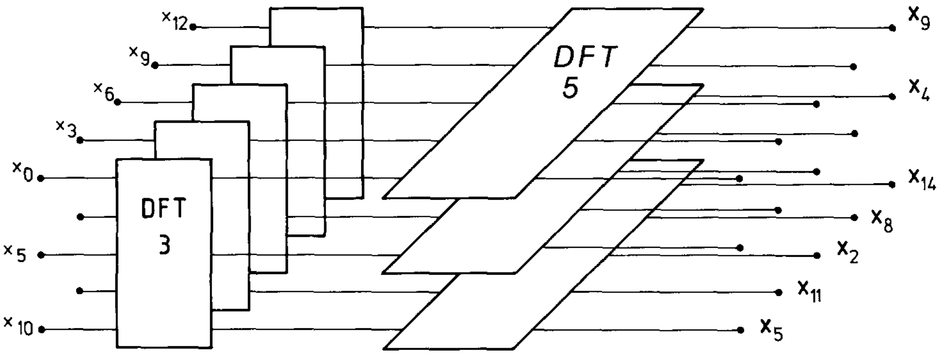


Fig. 7. Schematic view of PFA for $N = 15$.

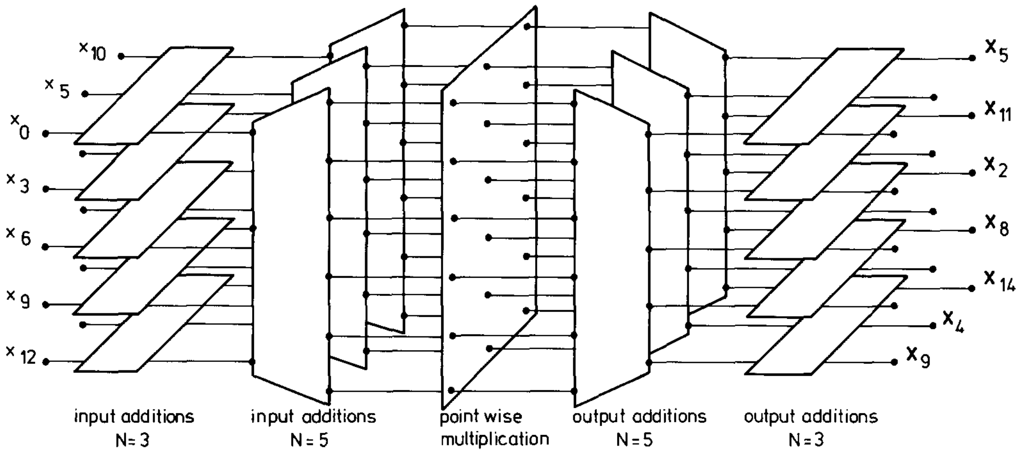


Fig. 8. Schematic view of WFTA for $N=15$.

postmultiplying does it for each column. Merging both multiplications leads to a total number of

$$M_{N_1N_2}=M_{N_1}\cdot M_{N_2} \tag{58}$$

out of which $M'_{N_1}\cdot M'_{N_2}$ are trivial.

Pre- and postmultiplying by C_1 and C_2^{T} will then complete the algorithm.

A graphical display of WFTA for length $N=15$ is given in Fig. 8, which clearly shows that this algorithm cannot be performed in place.

The number of additions is more intricate to obtain.

Let us consider the pictorial representation of (57) as given in Fig. 8.

Let C_1 involve A_1^1 additions (output additions) and B_1 involve A_2^1 additions (input additions). (Which means that there exists an algorithm for multiplying C_1 by some vector involving A_1^1 additions. This is different from the number of ± 1's in the matrix—see the $p=5$ example.)

Under these conditions, obtaining xB_2 will cost $A_2^2\cdot N_1$ additions, $B_1(xB_2^{\mathrm{T}})$ will cost $A_1^2\cdot M_2$ additions, $C_1(D_1B_1xB_2^{\mathrm{T}})$ will cost $A_1^1\cdot M_2$ additions and $(C_1D_1B_1xB_2^{\mathrm{T}})C_2$ will cost $A_2^1\cdot N_1$ additions, which gives a total of

$$A_{N_1N_2}=N_1A_2+M_2A_1. \tag{59}$$

This formula is not symmetric in N_1 and N_2. Hence, it is possible to interchange N_1 and N_2, which does not change the number of multiplications. This is used to minimize the number of additions.

Since $M_2\geq N_2$, it is clear that WFTA will always require at least as many additions as PFA, while it will always need fewer multiplications, as long as optimum short length DFTs are used. The demonstration is as follows.

Let

$$M_1=N_1+\varepsilon_1,\qquad M_2=N_2+\varepsilon_2,$$

$$M_{\mathrm{PFA}}=N_1M_2+N_2M_1$$
$$=2N_1N_2+N_1\varepsilon_2+N_2\varepsilon_1,$$

$$M_{\mathrm{WFTA}}=M_1\cdot M_2$$
$$=N_1N_2+\varepsilon_1\varepsilon_2+N_1\varepsilon_2+N_2\varepsilon_1.$$

Since ε_1 and ε_2 are strictly smaller than N_1 and N_2 in optimum short-length DFTs, we have, as a result

$$M_{\mathrm{WFTA}}<M_{\mathrm{PFA}}.$$

Note that this result is not true if suboptimal short-length FFTs are used. The numbers of operations to be performed per output point (to be compared with (56)) are as follows in the WFTA:

$$m_{N_1N_2}=m_{N_1}\cdot M_{N_2},\qquad a_{N_1N_2}=a_{N_2}+m_{N_2}a_{N_1}. \tag{60}$$

These numbers are given in Tables 1 and 2.

Note that the number of additions in the WFTA was reduced later by Nussbaumer with a scheme called 'split nesting' [12], leading to the algorithm with the least known number of operations (multiplications + additions).

5.4. *Other members of this class [38]*

PFA and WFTA are seen to be both described by the following equation:

$$X = C_1 D_1 B_1 x B_2^{\mathrm{T}} D_2 C_2^{\mathrm{T}}. \qquad (61)$$

Each of them is obtained by different ordering of the matrix products.

—The PFA multiplies $(C_1 D_1 B_1)x$ first, and then the result is postmultiplied by $(B_2^{\mathrm{T}} D_2 C_2^{\mathrm{T}})$.

—The WFTA starts with xB_2^{T}, then $(D_1 \times D_2)$, then C_1 and finally C_2^{T}.

Nevertheless, these are not the only ways of obtaining X: C and B can be factorized as two matrices each, to fully describe the way the algorithms are implemented. Taking this fact into account allows a great number of different algorithms to be obtained. Johnson and Burrus [38] systematically investigated this whole class of algorithms, obtaining interesting results, such as

—some WFTA-type algorithms, with reduced number of additions.

—algorithms with lower number of multiplications than both PFA and WFTA in the case where the short-length algorithms are not optimum.

5.5. *Remarks on FFTs without twiddle factors*

It is easily seen that members of this class of algorithms differ fundamentally from FFTs with twiddle factors.

Both classes of algorithms are based on a divide and conquer strategy, but the mapping used to eliminate the twiddle factors introduced strong constraints on the type of lengths that were possible with Good's mapping.

Due to those constraints, the elaboration of efficient FFTs based on Good's mapping required considerable work on the structure of the short FFTs. This resulted in a better understanding of the mathematical structure of the problem, and a better idea of what was feasible and what was not.

This new understanding has been applied to the study of FFTs with twiddle factors. In this study, issues, such as optimality, distance (in cost) of the practical algorithms from the best possible ones and the structural properties of the algorithms, have been prominent in the recent evolution of the field of algorithms.

6. State of the art

FFT algorithms have now reached a great maturity, at least in the 1-D case, and it is now possible to make strong statements about what eventual improvements are feasible and what are not.

In fact, lower bounds on the number of multiplications necessary to compute a DFT of given length can be obtained by using the techniques described in Section 5.1.

6.1. *Multiplicative complexity*

Let us first consider the FFTs with lengths that are powers of two.

Winograd [57] was first able to obtain a lower bound on the number of complex multiplications necessary to compute length 2^n DFTs. This work was then refined in [28], which provided realizable lower bounds, with the following multiplicative complexity:

$$\mu_{\mathrm{c}}[\mathrm{DFT}\ 2^n] = 2^{n+1} - 2n^2 + 4n - 8. \qquad (62)$$

This means that there will never exist any algorithm computing a length 2^n DFT with a lower number of non-trivial complex multiplications than the one in (62).

Furthermore, since the demonstration is constructive [28], this optimum algorithm is known. Unfortunately, it is of no practical use for lengths greater than 64 (it involves much too many additions).

The lower part of Fig. 9 shows the variation of this lower bound and of the number of complex multiplications required by some practical

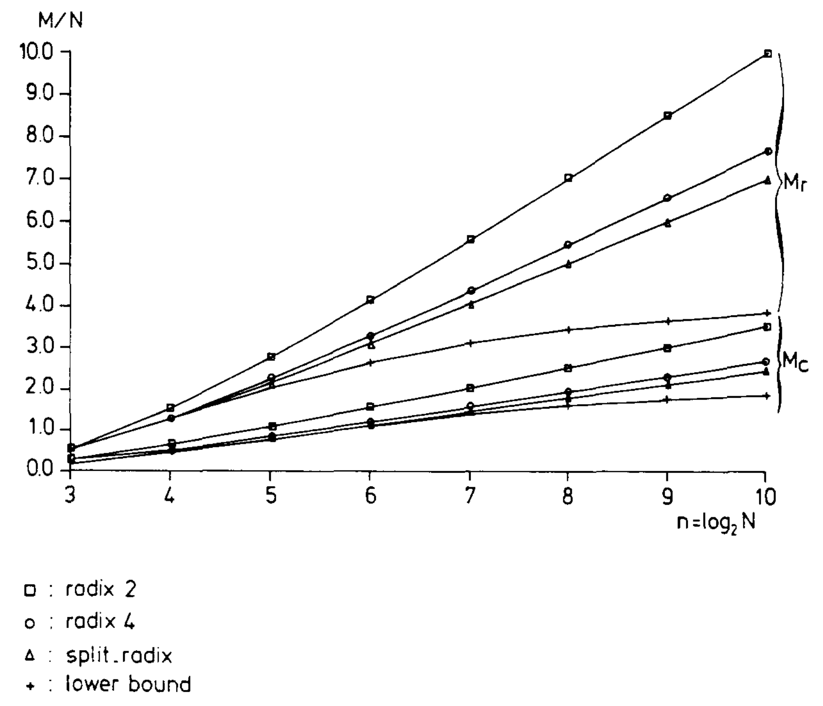


Fig. 9. Number of non-trivial real or complex multiplications per output point.

algorithms (radix 2, radix 4, SRFT). It is clearly seen that SRFFT follows this lower bound up to $N = 64$, and is fairly close for $N = 128$. Divergence is quite fast afterwards.

It is also possible to obtain a realizable lower bound on the number of real multiplications [35, 36].

$$\mu_r[\text{DFT}\ 2^n] = 2^{n+2} - 2n^2 - 2n + 4. \tag{63}$$

The variation of this bound, together with that of the number of real multiplications required by some practical algorithms is provided on the upper part of Fig. 9. Once again, this realizable lower bound is of no practical use above a certain limit. But, this time, the limit is much lower: SRFFT, together with radix 4, meets the lower bound on the number of real multiplications up to $N = 16$, which is also the last point where one can use an optimal polynomial product algorithm (modulo u^2+1) which is still practical. ($N = 32$ would require an optimal product modulo u^4+1 that requires a large number of additions).

It was also shown [31, 76] that all of the three following algorithms: optimum algorithm minimizing complex multiplications, optimum algorithm minimizing real multiplications and SRFFT, had exactly the same structure. They performed the decomposition into polynomial products exactly in the same manner, and they differ only in the way the polynomial products are computed.

Another interesting remark is as follows: the same number of multiplications as in SRFFT could also be obtained by so-called 'real factor radix-2 FFTs' [24, 42, 44] (which were, on another respect, somewhat numerically ill-conditioned and needed about 20% more additions). They were obtained by making use of some computational trick to replace the complex twiddle factors by purely real or purely imaginary ones. Now, the question is: is it possible to do the same kind of thing with radix 4, or even SRFFT? Such a result would provide algorithms with still fewer operations. The knowledge of the lower bound tells us that it is impossible since, for some points ($N = 16$, for example) this would produce an algorithm with better performance than the lower bound. The challenge of eventually improving SRFFT is now as follows:

Comparison of SRFFT with $\mu_c[\mathrm{DFT}\ 2^n]$ tells us that no algorithm using complex multiplications will be able to improve significantly SRFFT for lengths <512. Furthermore, the trick allowing real factor algorithms to be obtained cannot be applied to radices greater than 2 (or at least not in the same manner).

The above discussion thus shows that there remain very few approaches (yet unknown) that could eventually improve the best known length 2^n FFT.

And what is the situation for FFTs based on Good's mapping?

Realizable lower bounds are not so easily obtained. For a given length $N=\prod N_i$, they involve a fairly complicated number theoretic function [8], and simple analytical expressions cannot be obtained. Nevertheless, programs can be written to compute $\mu_r\{\mathrm{DFT}_N\}$, and are given in [36]. Table 3 provides numerical values for a number of lengths of interest.

Careful examination of Table 3 provides a number of interesting conclusions.

First, one can see that, for comparable lengths (since SRFFT and WFTA cannot exist for the same lengths), a classification depending on the efficiency is as follows: WFTA always requires the lowest number of multiplications, followed by PFA, and followed by SRFFT, all fixed or mixed-radix FFTs being next. Nevertheless, none of these algorithms attains the lower bound, except for very small lengths.

Another remark is that the number of multiplications required by WFTA is always smaller than the lower bound for the corresponding length that is a power of 2. This means on the one hand that transform lengths for which Good's mapping can be applied are well suited for a reduction in the number of multiplications, and on the other hand, that they are very efficiently computed by WFTA, from this point of view.

And this states the problem of the relative efficiencies of these algorithms: How close are they to their respective lower bound?

The last column of Table 3 shows that the relative efficiency of SRFFT decreases almost linearly with the length (it requires about twice the minimum number of multiplications for $N=2048$), while the relative efficiency of WFTA remains almost

Table 3
Practical algorithms vs. lower bounds (number of non-trivial real multiplications for FFTs on real data)

N	SRFFT	WFTA	Lower bound (L.B.)	$\frac{\text{SRFFT}}{\text{L.B.}}$	$\frac{\text{WFTA}}{\text{L.B.}}$
16	20		20	1	
30		68	56		1.21
32	68		64	1.06	
60		136	112		1.21
64	196		168	1.16	
120		276	240		1.15
128	516		396	1.3	
240		632	548		1.15
256	1284		876	1.47	
504		1572	1320		1.19
512	3076		1864	1.64	
1008		3548	2844		1.25
1024	7172		3872	1.85	
2048	16388		7876	2.08	
2520		9492	7440		1.27

constant for all the lengths of interest (it would not be the same result for much greater N). Lower bounds for Winograd-type lengths are also seen to be smaller than for the corresponding power of 2 lengths.

All these considerations result in the following conclusion: lengths for which Good's mapping is applicable allow a greater reduction of the number of multiplications (which is due directly to the mathematical structure of the problem). And, furthermore, they allow a greater relative efficiency of the actual algorithms vs. the lower bounds (and this is due indirectly to the mathematical structure).

6.2. Additive complexity

Nevertheless, the situation is not the same as regards the number of additions.

Most of the work on optimality was concerned with the number of multiplications. Concerning the number of additions, one can distinguish between additions due to the complex multiplications and the ones due to the butterflies. For the case $N=2^n$, it was shown in [106, 110], that the latter number which is achieved in actual algorithms is also the optimum. Differences between the various algorithms is thus only due to varying numbers of complex multiplications. As a conclusion, one can see that the only way to decrease the number of additions is to decrease the number of true complex multiplications (which is close to the lower bound).

Figure 10 gives the variation of the total number of operations (multiplications plus additions) for these algorithms, showing that SRFFT has the lowest operation count. Furthermore, its more regular structure results in faster implementations.

Note that all the numbers given here concern the initial versions of SRFFT, PFA and WFTA, for which FORTRAN programs are available. It is nevertheless possible to improve the number of additions in WFTA by using the so-called split-nesting technique [12] (which is used in Fig. 10), and the number of multiplications of PFA by using

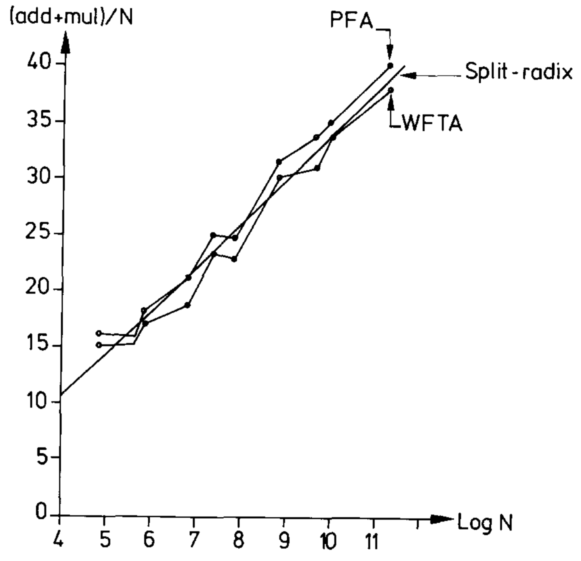


Fig. 10. Total number of operations per output point for different algorithms.

small-length FFTs with scaled output [12], resulting in an overall scaled DFT.

As a conclusion, one can realize that we now have practical algorithms (mainly WFTA and SRFFT) that follow the mathematical structure of the problem of computing the DFT with the minimum number of multiplications, as well as a knowledge of their degree of suboptimality.

7. Structural considerations

This section is devoted to some points that are important in the comparison of different FFT algorithms, namely easy obtention of inverse FFT, in-place computation, regularity of the algorithm, quantization noise and parallelization, all of which are related to the structure of the algorithms.

7.1. Inverse FFT

FFTs are often used regardless of their 'frequency' interpretation for computing FIR filtering in blocks, which achieves a reduction in arithmetic complexity compared to the direct algorithm. In that case, the forward FFT has to be followed, after pointwise multiplication of the result, by an inverse FFT. It is of course possible to rewrite a program along the same lines as the forward one, or to reorder the outputs of a forward FFT. A simpler way of computing an inverse FFT by using a forward FFT program is given (or reminded) in [99], where it is shown that, if CALL FFT (XR, XI, N) computes a forward FFT of the sequence $\{XR(i)+jXI(i) \mid i=0,\ldots,N-1\}$, CALL FFT(XI, XR, N) will compute an inverse FFT of the same sequence, whatever the algorithm is. Thus, all FFT algorithms on complex data are equivalent in that sense.

7.2. *In-place computation*

Another point in the comparison of algorithms is the memory requirement: most algorithms (Cooley-Tukey, SRFFT, PFA) allow in-place computation (no auxiliary storage of size depending on N is necessary), while WFTA does not. And this may be a drawback for WFTA when applied to rather large sequences.

Cooley-Tukey and split-radix FFTs also allow rather compact programs [4, 113], the size of which is independent of the length of the FFT to be computed.

On the contrary, PFA and WFTA will require longer and longer programs when the upper limit on the possible lengths is increased: an 8-module program ($n=2, 4, 8, 16, 3, 5, 7, 9$) allows to obtain a rather dense set of lengths up to $N=5040$ only. Longer transforms can only be obtained either by the use of rather 'exotic' modules that can be found in [37], or by some kind of mixture between Cooley-Tukey FFT (or SRFFT) and PFA.

7.3. *Regularity, parallelism*

Regularity has been discussed for nearly all algorithms when they were described. Let us recall here that Cooley-Tukey FFT (CTFFT) is very regular (based on repetitive use of a few modules). SRFFT follows (repetitive use of very few modules in a slightly more involved manner). Then, PFA requires repetitive use (more intricate than CTFFT) of more modules, and finally WFTA requires some combining of parts of these modules, which means that, even if it has some regularity, this regularity is more hidden ....

Let us point out also that the regularity of an algorithm cannot really be seen from its flowgraph. The equations describing the algorithm, as given in (13) or (39) do not fully define the implementations, which is partially done in the flowgraph. The reordering of the nodes of a flowgraph may provide a more regular one (the classical radix 2 and 4 CTFFT can be reordered into a constant geometry algorithm. See also [30] for SRFFT).

Parallelization of CTFFT and SRFFT is fairly easy, since the small modules are applied on sets of data that are separable and contiguous, while it is slightly more difficult with PFA, where the data required by each module are not in contiguous locations.

Finally, let us point out that mathematical tools such as tensor products can be used to work on the structure of the FFT algorithms [50, 101], since the structure of the algorithm reflects the mathematical structure of the underlying problem.

7.4. *Quantization noise*

Roundoff noise generated by finite precision operations inside the FFT algorithm is also of importance. Of course, fixed point implementations of CTFFT for lengths 2^n were studied first, and it was shown that the error-to-signal ratio of the FFT process increases as $\sqrt{N}$ (which means 1/2 bit per stage) [117]. SRFFT and radix-4 algorithms were also reported to generate less roundoff than radix-2 [102].

Although the WFTA requires fewer multiplications than the CTFFT (hence has less noise sources), it was soon recognized that proper scaling was difficult to include in the algorithm, and that the resulting noise-to-signal ratio was higher. It is usually thought that two more bits are necessary for representing data in the WFTA to give an error of the same order as CTFFT (at least for practical lengths). A floating point analysis of PFA is provided in [104].

8. Particular cases and related transforms

The previous sections have been devoted exclusively to the computation of the matrix-vector product involving the Fourier matrix. In particular, no assumption has been made on the input or output vector. In the following subsections, restrictions will be put on these vectors, showing how the previously described algorithms can be applied when the input is e.g. real valued, or when only a part of the output is desired. Then, transforms closely related to the DFT will be discussed as well.

8.1. *DFT algorithms for real data*

Very often in applications, the vector to be transformed is made up of real data. The transformed vector then has an hermitian symmetry, that is,

$$X_{N-k} = X_k^*, \tag{64}$$

as can be seen from the definition of the DFT. Thus, X_0 is real, and when N is even, $X_{N/2}$ is real as well. That is, the N input values map to 2 real and $N/2-1$ complex conjugate values when N is even, or 1 real and $(N-1)/2$ complex conjugate values when N is odd (which leaves the number of free variables unchanged).

This redundancy in both input and output vectors can be exploited in the FFT algorithms in order to reduce the complexity and storage by a factor of 2. That the complexity should be half can be shown by the following argument. If one takes a real DFT of the real and imaginary parts of a complex vector separately, then $2N$ additions are sufficient in order to obtain the result of the complex DFT [3]. Therefore, the goal is to obtain a real DFT that uses half as many multiplications and less than half as many additions. If one could do better, then it would improve the complex FFT as well by the above construction.

For example, take the DIF SRFFT algorithm (28). First, X_{2k} requires a half length DFT on real data, and thus the algorithm can be reiterated. Then, because of the hermitian symmetry property (64):

$$X_{4k+1} = X^*_{4(N/4-k-1)+3}, \tag{65}$$

and therefore (28c) is redundant and only one DFT of size $N/4$ on complex data needs to be evaluated for (28b). Counting operations, this algorithm requires exactly half as many multiplications and slightly less than half as many additions as its complex counterpart, or [30]

$$M(\text{R-DFT}(2^m)) = 2^{n-1}(n-3)+2, \tag{66}$$

$$A(\text{R-DFT}(2^m)) = 2^{n-1}(3n-5)+4. \tag{67}$$

Thus, the goal for the real DFT stated earlier has been achieved. Similar algorithms have been developed for radix-2 and radix-4 FFTs as well. Note that even if DIF algorithms are more easily explained, it turns out that DIT ones have a better structure when applied to real data [29, 65, 77].

In the PFA case, one has to evaluate a multidimensional DFT on real input. Because the PFA is a row-column algorithm, data become hermitian after the first 1-D FFTs, hence an accounting has to be made of the real and conjugate parts so as to divide the complexity by 2 [77]. Finally, in the WFTA case, the input addition matrix and the diagonal matrix are real, and the output addition matrix has complex conjugate rows, showing again the saving of 50% when the input is real. Note, however, that these algorithms generally have a more involved structure than their complex counterparts (especially in the PFA and WFTA case). Some algorithms have been developed which are inherently 'real', like the real factor FFTs [44, 22] or the FFCT algorithm [51], and do not require substantial changes for real input.

A closely related question is how to transform (or actually back transform) data that possess hermitian symmetry. An actual algorithm is best derived by using the transposition principle: since the Fourier transform is unitary, its inverse is equal to its hermitian transpose, and the required algorithm can be obtained simply by transposing the flow graph of the forward transform (or by transposing the matrix factorization of the algorithm). Simple graph theoretic arguments show that both the multiplicative and additive complexity are exactly conserved.

Assume next that the input is real and that only the real (or imaginary) part of the output is desired. This corresponds to what has been called a cosine (or sine) DFT, and obviously, a cosine and a sine DFT on a real vector can be taken altogether at the cost of a single real DFT. When only a cosine DFT has to be computed, it turns out that algorithms can be derived so that only half the complexity of a real DFT (that is, the quarter of a complex DFT) is required [30, 52], and the same holds for the sine DFT as well [52]. Note that the above two cases correspond to DFTs on real and symmetric (or antisymmetric) vectors.

8.2. DFT pruning

In practice, it may happen that only a small number of the DFT outputs are necessary, or that only a few inputs are different from zero. Typical cases appear in spectral analysis, interpolation and fast convolution applications. Then, computing a full FFT algorithm can be wasteful, and advantage should be taken of the inputs and outputs that can be discarded.

We will not discuss 'approximate' methods which are based on filtering and sampling rate changes [2, pp. 317-319] but only consider 'exact' methods. One such algorithm is due to Goertzel [68] which is based on the complex resonator idea. It is very efficient if only a few outputs of the FFT are required. A direct approach to the problem consists in pruning the flowgraph of the complete FFT so as to disregard redundant paths (corresponding to zero inputs or unwanted outputs). As an inspection of a flow graph quickly shows, the achievable gains are not spectacular, mainly because of the fact that data communication is not local (since all arithmetic improvements in the FFT over the DFT are achieved through data shuffling).

More complex methods are therefore necessary in order to achieve the gains one would expect. Such methods lead to an order of $N \log_2 K$ operations, where N is the transform size and K the number of active inputs or outputs [48]. Reference [78] also provides a method combining Goertzel's method with shorter FFT algorithms. Note that the problems of input and output pruning are dual, and that algorithms for one problem can be applied to the other by transposition.

8.3. Related transforms

Two transforms which are intimately related to the DFT are the discrete Hartley transform (DHT) [61, 62] and the discrete cosine transform (DCT) [1, 59]. The former has been proposed as an alternative for the real DFT and the latter is widely used in image processing.

The DHT is defined by

$$X_k = \sum_{n=0}^{N-1} x_n(\cos(2\pi nk/N) + \sin(2\pi nk/N)) \tag{68}$$

and is self-inverse, provided that X_0 is further weighted by $1/\sqrt{2}$. Initial claims for the DHT were

—improved arithmetic efficiency. This was soon recognized to be false, when compared to the real DFT. The structures of both programs are very similar and their arithmetic complexities are equivalent (DHTs actually require slightly more additions than real-valued FFTs).

—self-inverse property. It has been explained above that the inverse real DFT on hermitian data has exactly the same complexity as the real DFT (by transposition). If the transposed algorithm is not available, it can be found in [65] how to compute the inverse of a real DFT with a real DFT with only a minor increase in additive complexity.

Therefore, there is no computational gain in using a DHT, and only a minor structural gain if an inverse real DFT cannot be used.

The DCT, on the other hand, has found numerous applications in image and video processing. This has led to the proposal of several fast algorithms for its computation [51, 64, 70, 72]. The DCT is defined by

$$X_k = \sum_{n=0}^{N-1} x_n \cos(2\pi(2k+1)n/4N). \tag{69}$$

A scale factor of $1/\sqrt{2}$ for X_0 has been left out in (69), mainly because the above transform appears as a subproblem in a length-$4N$ real DFT [51]. From this, the multiplicative complexity of the DCT can be related to that of the real DFT as [69]

$$\begin{aligned}&\mu(\mathrm{DCT}(N))\\ &\quad=(\mu(\text{real-DFT}(4N))\\ &\qquad-\mu(\text{real-DFT}(2N)))/2.\end{aligned} \tag{70}$$

Practical algorithms for the DCT depend as expected, on the transform length.
—N odd: the DCT can be mapped through permutations and sign changes only into a same length real DFT [69].
—N even: the DCT can be mapped into a same length real DFT plus $N/2$ rotations [51]. This is not the optimal algorithm [69, 100] but, however, a very practical one.

Other sinusoidal transforms [71], like the discrete sine transform (DST), can be mapped into DCTs as well, with permutations and sign changes only. The main point of this paragraph is that DHTs, DCTs and other related sinusoidal transforms can be mapped into DFTs, and therefore one can resort to the vast and mature body of knowledge that exists for DFTs. It is worth noting that so far, for all sinusoidal transforms that have been considered, a mapping into a DFT has always produced an algorithm that is at least as efficient as any direct factorization. And if an improvement is ever achieved with a direct factorization, then it could be used to improve the DFT as well. This is the main reason why establishing equivalences between computational problems is fruitful, since it allows to improve the whole class when any member can be improved.

Figure 11 shows the various ways the different transforms are related: starting from any transform with the best known number of operations, you may obtain by following the appropriate arrows the corresponding transform for which the minimum number of operations will be obtained as well.

9. Multi-dimensional transforms

We have already seen in Sections 4 and 5 that both types of divide and conquer strategies resulted in a multi-dimensional transform with some particularities: in the case of the Cooley-Tukey mapping, some 'twiddle factors' operations had to be performed between the treatment of both dimensions, while in the Good's mapping, the resulting array had dimensions which were coprime.

Here, we shall concentrate on true 2-D FFTs with the same size along each dimension (generalization to more dimensions is usually straightforward).

Another characteristic of the 2-D case, is the large memory size required to store the data. It is therefore important to work in-place. As a consequence, in-place programs performing FFTs on real data are also more important in the 2-D case, due to this memory size problem. Furthermore, the required memory is often so large that the data are stored in mass memory and brought into core memory when required, by rows or columns. Hence, an important parameter when evaluating

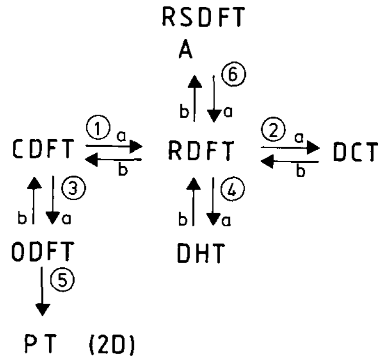


Fig. 11(a). Consistency of the split-radix based algorithms. Path showing the connections between the various transforms.

1) a	Complex DFT 2^n	2 real DFT's 2^n + 2^{n+1} - 4 additions
b	Real DFT 2^n	1 real DFT 2^{n-1} + 1 complex DFT 2^{n-2} + $(3.2^{n-2} - 4)$ multiplications + $(2^n + 3.2^{n-2}-n)$ additions
2) a	Real DFT 2^n	1 real DFT 2^{n-1} + 2 DCT's 2^{n-2} + 3.2^{n-1}-2 additions
b	DCT 2^n	1 real DFT 2^n + $(3.2^{n-1}-2)$ multiplications + $(3.2^{n-1}-3)$ additions
3) a	Complex DFT 2^n	1 odd DFT 2^{n-1}+ 1 complex DFT 2^{n-1} + 2^{n+1} additions
b	Odd DFT 2^{n-1}	2 complex DFT's 2^{n-2} + $2(3.2^{n-2}-4)$ multiplications + $(2^n+3.2^{n-1}-8)$ additions
4) a	Real DFT 2^n	1 DHT 2^n - 2 additions
b	DHT 2^n	1 real DFT 2^n + 2 additions
5)	Complex DFT $2^n x 2^n$	3.2^{n-1} odd DFT 2^{n-1} + 1 complex DFT $2^{n-1}x2^{n-1}$ + $n.2^n$ additions
6) a	Real DFT 2^n	1 real symmetric DFT 2^n + 1 real antisymmetric DFT 2^n + $(6n+10).4^{n-1}$ additions
b	Real symm DFT 2^n	1 real symmetric DFT 2^{n-1} + 1 inverse real DFT + $3(2^{n-3}-1)+1$ multiplications + $(3n-4).2^{n-3}+1$ additions

Fig. 11(b). Consistency of the split-radix based algorithms. Weighting of each connection in terms of real operations.

2-D FFT algorithms is the amount of memory calls required for performing the algorithm.

The 2-D DFT to be computed is defined as follows:

$$X_{k,r} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x_{i,j} W_N^{ik+jr},$$
$$k, r = 0, \ldots, N-1. \tag{71}$$

The methods for computing this transform are distributed in four classes: row-column algorithms, vector-radix algorithms, nested algorithms and polynomial transform algorithms. Among them, only the vector-radix and the polynomial transform were specifically designed for the 2-D case. We shall only give the basic principles underlying these algorithms and refer to the literature for more details.

9.1. Row-column algorithms

Since the DFT is separable in each dimension, the 2-D transform given in (71) can be performed in two steps, as was explained for the PFA.

—First compute N FFTs on the columns of the data.

—Then compute N FFTs on the rows of the intermediate result.

Nevertheless, when considering 2-D transforms, one should not forget that the size of the data becomes huge quickly: a length 1024×1024 DFT requires 10^6 words of storage, and the matrix is therefore stored in mass memory. But, in that case, accessing a single data is not more costly than reading the whole block in which it is stored. An important parameter is then the number of memory accesses required for computing the 2-D FFT.

This is why the row-column FFT is often performed as shown in Fig. 12, by performing a matrix transposition between the FFTs on the columns and the FFTs on the rows, in order to allow an access to the data by blocks. Row-column algorithms are very easily implemented and only require efficient 1-D FFTs, as described before, together with a matrix transposition algorithm (for which an efficient algorithm [84] was proposed). Note, however, that the access problem tends to be reduced with the availability of huge core memories.

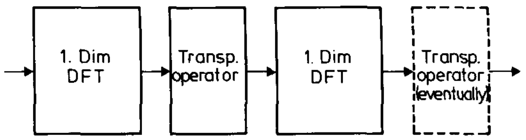


Fig. 12. Row-column implementation of the 2-D FFT.

9.2. Vector-radix algorithms

A computationally more efficient way of performing the 2-D FFT is a direct approach to the multi-dimensional problem: the vector-radix (VR) algorithm [91, 92, 85].

They can easily be understood through an example: the radix-2 DIT VRFFT.

This algorithm is based on the following decomposition:

$$\begin{aligned} X_{k,r} = &\sum_{i=0}^{N/2-1}\sum_{j=0}^{N/2-1} x_{2i,2j} W_{N/2}^{ik+jr} \\ &+ W_N^k \sum_{i=0}^{N/2-1}\sum_{j=0}^{N/2-1} x_{2i+1,2j} W_{N/2}^{ik+jr} \\ &+ W_N^r \sum_{i=0}^{N/2-1}\sum_{j=0}^{N/2-1} x_{2i,2j+1} W_{N/2}^{ik+jr} \\ &+ W_N^{k+r} \sum_{i=0}^{N/2-1}\sum_{j=0}^{N/2-1} x_{2i+1,2j+1} W_{N/2}^{ik+jr}, \end{aligned} \tag{72}$$

and the redundancy in the computation of $X_{k,r}$, $X_{k+N/2,r}$, $X_{k,r+N/2}$ and $X_{k+N/2,r+N/2}$ leads to simplifications which allow to reduce the arithmetic complexity.

This is the same approach as was used in the Cooley–Tukey FFTs, the decomposition being applied to both indices altogether.

Of course, higher radix decompositions or split radix decompositions are also feasible [86], the main difference being that the vector-radix SRFFT, as derived in [86], although being more efficient than the one in [90] is not the algorithm with the lowest arithmetic complexity in that class: For the 2-D case, the best algorithm is not only a mixture of radices 2 and 4.

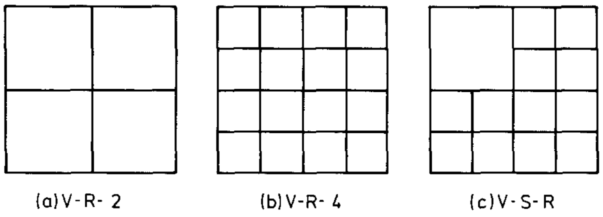


Fig. 13. Decomposition performed in various vector radix algorithms.

Figure 13 shows what kind of decompositions are performed in the various algorithms. Due to the fact that the VR algorithms are true generalizations of the Cooley–Tukey approach, it is easy to realize that they will be obtained by repetitive use of small blocks of the same type (the 'butterflies', by extension). Figure 14 provides the basic butterfly for a vector radix-2 FFT, as derived by (72). It should be clear, also, from Fig. 13 that the complexity of these butterflies increases very quickly with the radix: a radix-2 butterfly involves 4 inputs (it is a 2×2 DFT followed by some 'twiddle factors'), while VR4 and VSR butterflies involve 16 inputs.

Note also that the only VR algorithms that have seriously been considered all apply to lengths that are powers of 2, although other radices are of course feasible.

The number of read/write cycles of the whole set of data needed to perform the various FFTs of this class, compared to the row-column algorithm, can be found in [86].

9.3. Nested algorithms

They are based on the remark that the nesting property used in Winograd's algorithm, as

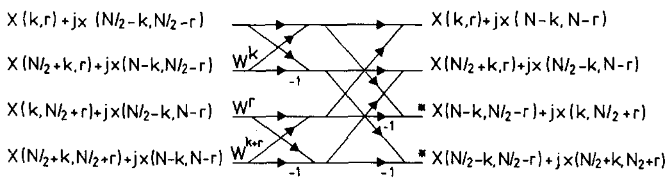


Fig. 14. General vector-radix 2 butterfly.

explained in Section 5.3 is not bound to the fact that the lengths are coprime (this requirement was only needed for Good's mapping). Hence, if the length of the DFT allows the corresponding 1-D DFT to be of a nested type (product of mutually prime factors), it is possible to nest further the multiplications, so that the overall 2-D algorithm is also nested.

The number of multiplications thus obtained are very low (see Table 4), but the main problem deals with memory requirements: WFTA is not performed in-place, and since all multiplications are nested, it requires the availability of a number of memory locations equal to the number of multiplications involved in the algorithms. For a length 1008×1008 FFT, this amounts to about $6 \cdot 10^6$ locations. This restricts the practical usefulness of these algorithms to small or medium length DFTs.

9.4. *Polynomial transform*

Polynomial transforms were first proposed by Nussbaumer [74] for the computation of 2-D cyclic convolutions. They can be seen as a generalization of Fourier transforms in the field of polynomials. Working in the field of polynomials resulted in a simplification of the multiplications by the root of unity, which was changed from a complex multiplication to a vector reordering. This powerful approach was applied in [87, 88] to the computation of 2-D DFTs as follows.

Let us consider the case where $N = 2^n$, which is the most common case.

The 2-D DFT of (71) can be represented by the following three polynomial equations:

$$X_i(z) = \sum_{j=0}^{N-1} x_{i,j} \cdot z^j, \tag{73a}$$

$$\bar{X}_k(z) = \sum_{i=0}^{N-1} X_i(z) W_N^{ik} \bmod (z^N - 1), \tag{73b}$$

$$X_{k,r} = \bar{X}_k(z) \bmod (z - W_N^r). \tag{73c}$$

This set of equations can be interpreted as follows: (73a) writes each row of the data as a polynomial, (73b) computes explicitly the DFTs on the columns, while (73c) computes the DFTs on the rows as a polynomial reduction (it is merely the equivalent of (5). Note that the modulo operation in (73b) is not necessary (no polynomial involved has a degree greater than N), but it will allow a divide and conquer strategy on (73c).

In fact, since $(z^N - 1) = (z^{N/2} - 1)(z^{N/2} + 1)$, the set of two equations (73b), (73c) can be separated into two cases, depending on the parity of r:

$$\bar{X}_k^1(z) = \sum_{i=0}^{N-1} X_i(z) W_N^{ik} \bmod (z^{N/2} - 1), \tag{74a}$$

Table 4

Number of non-trivial real multiplications per output point for various 2-D FFTs on real data

$N \times N$ (WFTA)	$N \times N$ (Others)	R.C.	VR2	VR4	VSR	WFTA	P.T.
	2×2	0	0		0		0
	4×4	0	0	0	0		0
	8×8	0.5	0.375		0.375		0.375
	16×16	1.25	1.25	0.844	0.844		0.844
30×30	32×32	2.125	2.062		1.43	1.435	1.336
	64×64	3.0625	3.094	2.109	2.02		1.834
120×120	128×128	4.031	4.172		2.655	1.4375	2.333
240×240	256×256	5.015	5.273	3.48	3.28	1.82	2.833
504×504	512×512	6.008	6.386		3.92	2.47	3.33
1008×1008	1024×1024	7.004	7.506	4.878	4.56	3.12	3.83

$$X_{k,2r} = \bar{X}^1_k(z) \bmod (z - W_N^{2r}), \quad (74b)$$

$$\bar{X}^2_k(z) = \sum_{i=0}^{N-1} X_i(z) W_N^{ik} \bmod (z^{N/2}+1), \quad (75a)$$

$$X_{k,2r+1} = \bar{X}^2_k(z) \bmod (z - W_N^{2r+1}). \quad (75b)$$

Equation (74) is still of the same type as the initial one, hence the same procedure as the one being derived will apply. Let us now concentrate on (75) which is now recognized to be the key aspect of the problem.

Since $(2r+1, N)=1$, the permutation $(2r+1)\cdot k(\bmod N)$ maps all values of k, and replacing k with $(2r+1)\cdot k$ in (74a) will merely result in a reordering of the outputs:

$$\bar{X}^2_{k(2r+1)}(z) = \sum_{i=0}^{N-1} X_i(z) W_N^{(2r+1)ik} \bmod (z^{N/2}+1), \quad (76a)$$

$$X_{k(2r+1),2r+1} = \bar{X}^2_{k(2r+1)}(z) \bmod (z - W_N^{2r+1}), \quad (76b)$$

and, since $z = W_N^{2r+1}$ in (76b), we can replace W_N^{2r+1}, by z in (76a):

$$\bar{X}^2_{k(2r+1)}(z) = \sum_{i=0}^{N-1} X_i(z) z^{ik} \bmod (Z^{N/2}+1), \quad (77)$$

which is exactly a polynomial transform, as defined in [74]. This polynomial transform can be computed using an FFT-type algorithm, without multiplications, and with only $N^2/2 \log_2 N$ additions.

$X_{k,2r+1}$ will now be obtained by application of (76b). $\bar{X}^2(z)$ being computed $\bmod(z^{N/2}+1)$ is of degree $N/2-1$. For each k, (76b) will then correspond to the reduction of one polynomial modulo the odd powers of W_N. From (5), this is seen to be the computation of the odd outputs of a length-N DFT, which is sometimes called an odd DFT.

The terms $X_{k,2r+1}$ are seen to be obtained by one reduction $\bmod(z^{N/2}+1)$ (75), one polynomial transform of N terms $\bmod z^{N/2}+1$ (77) and N odd DFTs. This procedure is then iterated on the terms $X_{2k+1,2r}$, by using exactly the same algorithm, the role of k and r being interchanged. $X_{2k,2r}$ is exactly a length $N/2 \times N/2$ DFT, on which the same algorithm is recursively applied.

In the first version of the polynomial transform computation of the 2-D FFT, the odd DFT was computed by a real-factor algorithm, resulting in an excess in the number of additions required.

As seen in Tables 4 and 5, where the number of multiplications and additions for the various 2-D FFT algorithms are given, the polynomial transform approach results in the algorithm requiring the lowest arithmetic complexity, when counting multiplications and additions altogether. The addition counts given in Table 5 are updates of the previous ones, assuming that the odd DFTs are computed by a split-radix algorithm.

Table 5

Number of real additions per output point for various 2-D FFTs on real data

$N\times N$ (WFTA)	$N\times N$ (Others)	R.C.	VR2	VR4	VSR	WFTA	P.T.
	2×2	2	2		2		2
	4×4	3.25	3.25	3.25	3.25		3.25
	8×8	5.56	5.43		5.43		5.43
	16×16	8.26	8.14	7.86	7.86		7.86
30×30	32×32	11.13	11.06		10.43	12.98	10.34
	64×64	14.06	14.09	13.11	13.02		12.83
120×120	128×128	17.03	17.17		15.65	17.48	15.33
240×240	256×256	20.01	20.27	18.48	17.67	22.79	17.83
504×504	512×512	23.00	23.38		20.92	34.42	20.33
1008×1008	1024×1024	26.00	26.5	23.88	23.56	45.30	22.83

Note that the same kind of performance was obtained by Auslander et al. [82, 83] with a similar approach which, while more sophisticated, gave a better insight on the mathematical structure of this problem. Polynomial transform were also applied to the computation of 2-D DCT [79, 52].

9.5. Discussion

A number of conclusions can be stated by considering Tables 4 and 5, keeping the principles of the various methods in mind.

VR2 is more complicated to implement than row-column algorithms, and requires more operations for lengths $\geqslant 32$. Therefore, it should not be considered. Note that this result holds only because efficient and compact 1-D FFTs, such as SRFFT, have been developed.

The row-column algorithm is the one allowing the easiest implementation, while having a reasonable arithmetic complexity. Furthermore, it is easily parallelized, and simplifications can be found for the reorderings (bit reversal, and matrix transposition [66]), allowing one of them to be free in nearly any kind of implementation. WFTA has a huge number of additions (twice the number required for the other algorithms for $N = 1024$), requires huge memory, has a difficult implementation, but requires the least multiplications. Nevertheless, we think that, in today's implementations, this advantage will in general not outweigh its drawbacks.

VSR is difficult to implement, and will certainly seldom defeat VR4, except in very special cases (huge memory available and N very large).

VR4 is a good compromise between structural and arithmetic complexity. When row-column algorithms are not fast enough, we think it is the next choice to be considered.

Polynomial transforms have the greatest possibilities: lowest arithmetic complexity, possibility of in-place computation, but very little work was done on the best way of implementing them. It was even reported to be slower than VR2 [103]. Nevertheless, it is our belief that looking for efficient implementations of polynomial transform based FFTs is worth the trouble. The precise understanding of the link between VR algorithms and polynomial transforms may be a useful guide for this work.

10. Implementation issues

It is by now well recognized that there is a strong interaction between the algorithm and its implementation. For example, regularity, as discussed before, will only pay off if it is closely matched by the target architecture. This is the reason why we will discuss in the sequel different types of implementations. Note that very often, the difference in computational complexity between algorithms is not large enough so as to differentiate between the efficiency of the algorithm and the quality of the implementation ....

10.1. General purpose computers

FFT algorithms are built by repetitive use of basic building blocks. Hence, any improvement (even small) in these building blocks will pay in the overall performance. In the Cooley-Tukey or the split-radix case, the building blocks are small and thus easily optimizable, and the effect of improvements will be relatively more important than in the PFA/WFTA case where the blocks are larger.

When monitoring the amount of time spent in various elementary floating point operations, it is interesting to note that more time is spent in load/store operations than in actual arithmetic computations [30, 107, 109] (this is due to the fact that memory access times are comparable to ALU cycle times on current machines). Therefore, the locality of the algorithm is of paramount importance. This is why the PFA and WFTA do not meet the performance expected from their computational complexity only.

On another side, this drawback of PFA is compensated by the fact that only a few coefficients have to be stored. On the contrary, classical FFTs must store a large table of sine and cosine values, calculate them as needed, or update them with resulting roundoff errors.

Note that special automatic code generation techniques have been developed in order to produce efficient code for often used programs like the FFT. They are based on a 'de-looping' technique that produces loop free code from a given piece of code [107]. While this can produce unreasonably large code for large transforms, it can be applied successfully to sub-transforms as well.

10.2. Digital signal processors

Digital signal processors (DSPs) strongly favor multiply/accumulate based algorithms. Unfortunately, this is not matched by any of the fast FFT algorithms (where sums of products have been changed to fewer but less regular computations). Nevertheless, DSPs now take into account some of the FFT requirements, like modulo counters and bit-reversed addressing. If the modulo counter is general, it will help the implementation of all FFT algorithms, but it is often restricted to the Cooley-Tukey/SRFFT case only (modulo a power of 2) for which efficient timings are provided on nearly all available machines by manufacturers, at least for small to medium lengths.

10.3. Vector and multi processors

Implementations of Fourier transforms on vectorized computers must deal with two interconnected problems [93]. First, the vector (the size of data that can be processed at the maximal rate) has to be full as often as possible. Then, the loading of the vector should be made from data available inside the cache memory (like in general purpose computers) in order to save time. The usual hardware design parameters will in general favor length-2^m FFT implementations. For example, a radix-4 FFT was reported to be efficiently realized on a commercial vector processor [93].

In the multi-processor case, the performance will be dependent on the number and power of the processing nodes but also strongly on the available interconnection network. Because the FFT algorithms are deterministic, the resource allocation problem can be solved off-line. Typical configurations include arithmetic units specialized for butterfly operations [98], arrays with attached shuffle networks and pipelines of arithmetic units with intermediate storage and reordering [17]. Obviously, these schemes will often favor classical Cooley-Tukey algorithms, because of their high regularity. However, SRFFT or PFA implementations have not been reported yet, but could be promising in high speed applications.

10.4. VLSI

The discussion of partially dedicated multiprocessors leads naturally to fully dedicated hardware structures like the ones that can be realized in very large scale integration (VLSI) [9, 11]. As a measure of efficiency both chip area (A) and time (T) between two successive DFT computations (set-up times are neglected since only throughput is of interest) are of importance. Asymptotic lower bounds for the product $A \cdot T^2$ have been reported for the FFT [116] and lead to

$$\Omega_{AT^2}(\mathrm{DFT}(N)) = N^2 \log^2(N), \tag{78}$$

that is, no circuit will achieve a better behavior than (78) for large N. Interestingly, this lower bound is achieved by several algorithms, notably the algorithms based on shuffle-exchange networks and the ones based on square grids [96, 114]. The trouble with these optimal schemes is that they outperform more traditional ones, like the cascade connection with variable delay [98] (which is asymptotically sub-optimal), only for extremely large Ns and are therefore not relevant in practice [96].

Dedicated chips for the FFT computation are therefore often based on some traditional algorithm which is then efficiently mapped into a layout. Examples include chips for image processing with small size DCTs [115] as well as wafer scale integration for larger transforms. Note that the cost is dominated both by the number of multiplications (which outweigh additions in VLSI) and the cost of communication. While the former figure is available from traditional complexity theory, the latter one is not yet well studied and depends strongly on the structure of the algorithm as discussed in Section 7. Also, dedicated arithmetic units suited for the FFT problem have been devised, like the butterfly unit [98] or the CORDIC unit [94, 97] and contribute substantially to the quality of the overall design. But, similarly to the software case, the realization of an efficient VLSI implementation is still more an art than a mere technique.

11. Conclusion

The purpose of this paper has been threefold: a tutorial presentation of classic and recent results, a review of the state of the art, and a statement of open problems and directions.

After a brief history of the FFT development, we have shown by simple arguments, that the fundamental technique used in all fast Fourier transforms algorithms, namely the divide and conquer approach, will always improve the computational efficiency.

Then, a tutorial presentation of all known FFT algorithms has been made. A simple notation, showing how various algorithms perform various divisions of the input into periodic subsets, was used as the basis for a unified presentation of Cooley-Tukey, split-radix, prime factor, and Winograd fast Fourier transforms algorithms. From this presentation, it is clear that Cooley-Tukey and split-radix algorithms are instances of one family of FFT algorithms, namely FFTs with twiddle factors.

The other family is based on a divide and conquer scheme (Good's mapping) which is costless (computationally speaking). The necessary tools for computing the short-length FFTs which then appear were derived constructively and lead to the presentation of the PFA and of the WFTA.

These practical algorithms were then compared to the best possible ones, leading to an evaluation of their suboptimality. Structural considerations and special cases were addressed next. In particular, it was shown that recently proposed alternative transforms like the Hartley transform do not show any advantage when compared to real valued FFTs.

Special attention was then paid to multidimensional transforms, where several open problems remain. Finally, implementation issues were outlined, indicating that most computational structures implicitly favor classical algorithms. Therefore, there is room for improvements if one is able to develop architectures that match more recent and powerful algorithms.

Acknowledgments

The authors would like to thank Prof. M. Kunt for inviting them to write this paper, as well as for his patience. Prof. C. S. Burrus, Dr. J. Cooley, Dr. M. T. Heideman and Prof. H. J. Nussbaumer are also thanked for fruitful interactions on the subject of this paper. We are indebted to J. S. White, J. C. Bic and P. Gole for their careful reading of the manuscript.

References

Books

[1] N. Ahmed and K.R. Rao, *Orthogonal Transforms for Digital Signal Processing*, Springer, Berlin, 1975.
[2] R.E. Blahut, *Fast Algorithms for Digital Signal Processing*, Addison-Wesley, Reading, MA, 1986.
[3] E.O. Brigham, *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
[4] C.S. Burrus and T.W. Parks, *DFT/FFT and Convolution Algorithms*, Wiley, New York, 1985.
[5] C.S. Burrus, "Efficient Fourier transform and convolution algorithms", in: J.S. Lim and A.V. Oppenheim, eds., *Advanced Topics in Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
[6] Digital Signal Processing Committee, ed., *Selected Papers in Digital Signal Processing, II*, IEEE Press, New York, 1975.

[7] Digital Signal Processing Committee ed., *Programs for Digital Signal Processing*, IEEE Press, New York, 1979.

[8] M.T. Heideman, *Multiplicative Complexity, Convolution and the DFT*, Springer, Berlin, 1988.

[9] S.Y. Kung, H.J. Whitehouse and T. Kailath, eds., *VLSI and Modern Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[10] J.H. McClellan and C.M. Rader, *Number Theory in Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1979.

[11] C. Mead and L. Conway, *Introduction to VLSI*, Addison-Wesley, Reading, MA, 1980.

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

[13] A.V. Oppenheim ed., *Papers on Digital Signal Processing*, MIT Press, Cambridge, MA, 1969.

[14] A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[15] L.R. Rabiner and C.M. Rader ed., *Digital Signal Processing*, IEEE Press, New York, 1972.

[16] L.R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[17] E.E. Schwartzlander, *VLSI Signal Processing Systems*, Kluwer Academic Publishers, Dordrecht, 1986.

[18] M.A. Soderstrand, W.K. Jenkins, G.A. Jullien and F.J. Taylor, eds., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, IEEE Press, New York, 1986.

[19] S. Winograd, *Arithmetic Complexity of Computations*, SIAM CBMS-NSF Series, No. 33, SIAM, Philadelphia, 1980.

1-D FFT algorithms

[20] R.C. Agarwal and C.S. Burrus, "Fast one-dimensional digital convolution by multi-dimensional techniques", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-22, No. 1, February 1974, pp. 1-10.

[21] G.D. Bergland, "A fast Fourier transform algorithm using base 8 iterations", *Math. Comp.*, Vol. 22, No. 2, April 1968, pp. 275-279 (reprinted in [13]).

[22] G. Bruun, "*z*-transform DFT filters and FFTs", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-26, No. 1, February 1978, pp. 56-63.

[23] C.S. Burrus, "Index mappings for multidimensional formulation of the DFT and convolution", *IEEE Trans. Acoust. Speech Signal Process.*, ASSP Vol. 25, No. 3, June 1977, pp. 239-242.

[24] K.M. Cho and G.C. Temes, "Real-factor FFT algorithms", *Proc. ICASSP 78*, Tulsa, OK, April 1978, pp. 634-637.

[25] J.W. Cooley and J.W. Tukey, "An algorithm for the machine calculation of complex Fourier series", *Math. Comp.*, Vol. 19, April 1965, pp. 297-301.

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

[27] P. Duhamel and H. Hollmann, "Split-radix FFT algorithm", *Electronics Letters*, Vol. 20, No. 1, 5 January 1984, pp. 14-16.

[28] 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, August 1984, pp. 690-692.

[29] P. Duhamel, "Un algorithme de transformation de Fourier rapide à double base", *Annales des Telecommunications*, Vol. 40, Nos. 9-10, September 1985, pp. 481-494.

[30] P. Duhamel, "Implementation of "split-radix" FFT algorithms for complex, real and real-symmetric data", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-34, No. 2, April 1986, pp. 285-295.

[31] P. Duhamel, "Algorithmes de transformées discrètes rapides pour convolution cyclique et de convolution cyclique pour transformées rapides", Thèse de doctorat d'état, Université Paris XI, September 1986.

[32] I.J. Good, "The interaction algorithm and practical Fourier analysis", *J. Roy. Statist. Soc. Ser. B*, Vol. B-20, 1958, pp. 361-372, Vol. B-22, 1960, pp. 372-375.

[33] M.T. Heideman and C.S. Burrus, "A bibliography of fast transform and convolution algorithms II", Technical Report No. 8402, Rice University, 24 February 1984.

[34] M.T. Heideman, D.H. Johnson and C.S. Burrus, "Gauss and the history of the FFT", *IEEE Acoust. Speech Signal Process. Magazine*, Vol. 1, No. 4, October 1984, pp. 14-21.

[35] 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 Process.*, Vol. ASSP-34, No. 1, February 1986, pp. 91-95.

[36] M.T. Heideman, "Application of multiplicative complexity theory to convolution and the discrete Fourier transform" PhD Thesis, Dept. of Elec. and Comp. Eng., Rice Univ., April 1986.

[37] H.W. Johnson and C.S. Burrus, "Large DFT modules: 11, 13, 17, 19, and 25", Tech. Report 8105, Dept of Elec. Eng., Rice Univ., Houston, TX, December 1981.

[38] H.W. Johnson and C.S. Burrus, "The design of optimal DFT algorithms using dynamic programming", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-31, No. 2, April 1983, pp. 378-387.

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

[40] J.B. Martens, "Recursive cyclotomic factorization—A new algorithm for calculating the discrete Fourier transform", *IEEE Trans. Accoust. Speech Signal Process.*, Vol. ASSP-32, No. 4, August 1984, pp. 750-761.

[41] H.J. Nussbaumer, "Efficient algorithms for signal processing", *Second European Signal Processing Conference*, EUSIPCO-83, Erlangen, September 1983.

[42] R.D. Preuss, "Very fast computation of the radix-2 discrete Fourier transform", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-30, August 1982, pp. 595-607.

[43] C.M. Rader, "Discrete Fourier transforms when the number of data samples is prime", *Proc. IEEE*, Vol. 56, 1968, pp. 1107-1008.

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

[45] R. Singleton, "An algorithm for computing the mixed radix fast Fourier transform", *IEEE Trans. Audio Electroacoust.*, Vol. AU-17, June 1969, pp. 93-103 (reprinted in [13]).

[46] R. Stasinski, "Asymmetric fast Fourier transform for real and complex data", *IEEE Trans. Acoust. Speech Signal Process.*, submitted.

[47] R. Stasinski, "Easy generation of small-N discrete Fourier transform algorithms", *IEE Proc.*, Vol. 133, Pt. G, No. 3, June 1986, pp. 133-139.

[48] R. Stasinski, "FFT pruning. A new approach", *Proceedings Eusipco 86*, 1986, pp. 267-270.

[49] Y. Suzuki, T. Sone and K. Kido, "A new FFT algorithm of radix 3, 6, and 12", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-34, No. 2, April 1986, pp. 380-383.

[50] C. Temperton, "Self-sorting mixed-radix fast Fourier transforms", *J. Comput. Phys.*, Vol. 52, No. 1, October 1983, pp. 1-23.

[51] M. Vetterli and H.J. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations", *Signal Process.*, Vol. 6, No. 4, August 1984, pp. 267-278.

[52] M. Vetterli and H.J. Nussbaumer, "Algorithmes de transformée de Fourier et cosinus mono et bi-dimensionnels", *Annales des Télécommunications*, Tome 40, Nos. 9-10, September-October 1985, pp. 466-476.

[53] M. Vetterli and P. Duhamel, "Split-radix algorithms for length-p^m DFTs", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-37, No. 1, January 1989, pp. 57-64.

[54] S. Winograd, "On computing the discrete Fourier transform", Proc. Nat. Acad. Sci. USA, Vol. 73, April 1976, pp. 1005-1006.

[55] S. Winograd, "Some bilinear forms whose multiplicative complexity depends on the field of constants", *Math. Systems Theory*, Vol. 10, No. 2, 1977, pp. 169-180 (reprinted in [10]).

[56] S. Winograd, "On computing the DFT", *Math. Comp.*, Vol. 32, No. 1, January 1978, pp. 175-199 (reprinted in [10]).

[57] S. Winograd, "On the multiplicative complexity of the discrete Fourier transform", *Adv. in Math.*, Vol. 32, No. 2, May 1979, pp. 83-117.

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

Related algorithms

[59] N. Ahmed, T. Natarajan and K.R. Rao, "Discrete cosine transform", *IEEE Trans. Comput.*, Vol. C-23, January 1974, pp. 88-93.

[60] G.D. Bergland, "A radix-eight fast Fourier transform subroutine for real-valued series", *IEEE Trans. Audio Electroacoust.*, Vol. 17, No. 1, June 1969, pp. 138-144.

[61] R.N. Bracewell, "Discrete Hartley transform", *J. Opt. Soc. Amer.*, Vol. 73, No. 12, December 1983, pp. 1832-1835.

[62] R.N. Bracewell, "The fast Hartley transform", *Proc. IEEE*, Vol. 22, No. 8, August 84, pp. 1010-1018.

[63] C.S. Burrus, "Unscrambling for fast DFT algorithms", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-36, No. 7, pp. 1086-1087.

[64] W.-H. Chen, C.H. Smith and S.C. Fralick, "A fast computational algorithm for the discrete cosine transform", *IEEE Trans. Comm.*, Vol. COM-25, September 1977, pp. 1004-1009.

[65] P. Duhamel and M. Vetterli, "Improved Fourier and Hartley transform algorithms. Application to Cyclic Convolution of Real Data", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-35, No. 6, June 1987, pp. 818-824.

[66] P. Duhamel and J. Prado, "A connection between bit-reverse and matrix transpose. Hardware and software consequences", *Proc. IEEE Acoust. Speech Signal Process.*, pp. 1403-1406.

[67] D.M. Evans, "An improved digit reversal permutation algorithm for the fast Fourier and Hartley transforms" *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-35, No. 8, August 87, pp. 1120-1125.

[68] G. Goertzel, "An algorithm for the evaluation of finite Fourier series", *Amer. Math. Monthly, Vol.* 65, No. 1, January 1958, pp. 34-35.

[69] M.T. Heideman, "Computation of an odd-length DCT from a real-valued DFT of the same length", *IEEE Trans. Acoust. Speech Signal Process.*, submitted.

[70] H.S. Hou, "A fast recursive algorithm for computing the discrete Fourier transform", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-35, No. 10, October 1987, pp. 1455-1461.

[71] A.K. Jain, "A sinusoidal family of unitary transforms", *IEEE Trans. PAMI*, Vol. 1, No. 4, October 1979, pp. 356-365.

[72] B.G. Lee, "A new algorithm to compute the discrete cosine transform", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-32, December 1984, pp. 1243-1245.

[73] Z.J. Mou and P. Duhamel, "Fast FIR filtering: algorithms and implementations", *Signal Process.*, Vol. 13, No. 4, December 1987, pp. 377-384.

[74] H.J. Nussbaumer, "Digital filtering using polynomial transforms", *Electronics Letters*, Vol. 13, No. 13, June 1977, pp. 386-387.

[75] R.J. Polge, B.K. Bhaganan and J.M. Carswell, "Fast computational algorithms for bit-reversal", *IEEE Trans. Comput.*, Vol. 23, No. 1, January 1974, pp. 1-9.

[76] P. Duhamel, "Algorithms meeting the lower bounds on the multiplicative complexity of length-2^n DFTs and their connection with practical algorithms", *IEEE Trans. Acoust. Speech Signal Process.*, September 1990.

[77] H.V. Sorensen, D.L. Jones, M.T. Heideman and C.S. Burrus, "Real-valued fast Fourier transform algorithms", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-35, No. 6, June 1987, pp. 849-863.

[78] H.V. Sorensen, C.S. Burrus and D.L. Jones, "A new efficient algorithm for computing a few DFT points", *Proc. of the 1988 IEEE Internat. Symp. on CAS*, 1988, pp. 1915-1918.

[79] M. Vetterli, "Fast 2-D discrete cosine transform", *Proc. 1985 IEEE Internat. Conf. Acoust. Speech Signal Process.*, Tampa, March 1985, pp. 1538-1541.

[80] M. Vetterli, "Analysis, synthesis and computational complexity of digital filter banks", PhD Thesis, Ecole Polytechnique Federale de Lausanne, Switzerland, April 1986.

[81] M. Vetterli, "Running FIR and IIR filtering using multirate filter banks", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-36, No. 5, May 1988, pp. 730-738.

Multi-dimensional transforms

[82] L. Auslander, E. Feig and S. Winograd, "New algorithms for the multidimensional Fourier transform", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-31, No. 2, April 1983, pp. 338-403.

[83] L. Auslander, E. Feig and S. Winograd, "Abelian semisimple algebras and algorithms for the discrete Fourier transform", *Adv. in Applied Math.*, Vol. 5, 1984, pp. 31-55.

[84] J.O. Eklundh, "A fast computer method for matrix transposing", *IEEE Trans. Comput.*, Vol. 21, No. 7, July 1972, pp. 801-803 (reprinted in [6]).

[85] R.M. Mersereau and T.C. Speake, "A unified treatment of Cooley-Tukey algorithms for the evaluation of the multidimensional DFT", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. 22, No. 5, October 1981, pp. 320-325.

[86] Z.J. Mou and P. Duhamel, "In-place butterfly-style FFT of 2-D real sequences", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-36, No. 10, October 1988, pp. 1642-1650.

[87] H.J. Nussbaumer and P. Quandalle, "Computation of convolutions and discrete Fourier transforms by polynomial transforms", *IBM J. Res. Develop.*, Vol. 22, 1978, pp. 134-144.

[88] H.J. Nussbaumer and P. Quandalle, "Fast computation of discrete Fourier transforms using polynomial transforms", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-27, 1979, pp. 169-181.

[89] M.C. Pease, "An adaptation of the fast Fourier transform for parallel processing", *J. Assoc. Comput. Mach.*, Vol. 15, No. 2, April 1968, pp. 252-264.

[90] S.C. Pei and J.L. Wu, "Split-vector radix 2-D fast Fourier transform", *IEEE Trans. Circuits Systems*, Vol. 34, No. 1, August 1987, pp. 978-980.

[91] G.E. Rivard, "Algorithm for direct fast Fourier transform of bivariant functions", *1975 Annual Meeting of the Optical Society of America*, Boston, MA, October 1975.

[92] G.E. Rivard, "Direct fast Fourier transform of bivariant functions", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. 25, No. 3, June 1977, pp. 250-252.

Implementations

[93] R.C. Agarwal and J.W. Cooley, "Fourier transform and convolution subroutines for the IBM 3090 Vector Facility", *IBM J. Res. Develop.*, Vol. 30, No. 2, March 1986, pp. 145-162.

[94] H. Ahmed, J.M. Delosme and M. Morf, "Highly concurrent computing structures for matrix arithmetic and signal processing", *IEEE Trans. Comput.*, Vol. 15, No. 1, January 1982, pp. 65-82.

[95] C.S. Burrus and P.W. Eschenbacher, "An in-place, in order prime factor FFT algorithm", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-29, No. 4, August 1981, pp. 806-817.

[96] H.C. Card, "VLSI computations: from physics to algorithms", *Integration*, Vol. 5, 1987, pp. 247-273.

[97] A.M. Despain, "Fourier transform computers using CORDIC iterations", *IEEE Trans. Comput.*, Vol. 23, No. 10, October 1974, pp. 993-1001.

[98] A.M. Despain, "Very fast Fourier transform algorithms hardware for implementation", *IEEE Trans. Comput.*, Vol. 28, No. 5, May 1979, pp. 333-341.

[99] P. Duhamel, B. Piron and J.M. Etcheto, "On computing the inverse DFT", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-36, No. 2, February 1988, pp. 285-286.

[100] P. Duhamel and H. H'mida, "New 2^n DCT algorithms suitable for VLSI implementation" *Proc. IEEE Internat. Conf. Acoust. Speech Signal Process.*, 1987, pp. 1805-1809.

[101] J. Johnson, R. Johnson, D. Rodriguez and R. Tolimieri, "A methodology for designing, modifying, and implementing Fourier transform algorithms on various architectures" Preliminary draft, September 1988 (to be submitted).

[102] A. Elterich and W. Stammler, "Error analysis and resulting structural improvements for fixed point FFT's", *Proc. IEEE Internat. Conf. Acoust. Speech Signal Process.*, April 1988, pp. 1419-1422.

[103] B. Lhomme, J. Morgenstern and P. Quandalle, "Implantation de transformées de Fourier de dimension 2", *Techniques et Science Informatiques*, Vol. 4, No. 2, 1985, pp. 324-328.

[104] D.C. Manson and B. Liu, "Floating point roundoff error in the prime factor FFT", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. 29, No. 4, August 1981, pp. 877-882.

[105] B. Mescheder, "On the number of active *-operations needed to compute the DFT", *Acta Inform.*, Vol. 13, May 1980, pp. 383-408.

[106] J. Morgenstern, "The linear complexity of computation", *Assoc. Comput. Mach.*, Vol. 22, No. 2, April 1975, pp. 184-194.

[107] L.R. Morris, "Automatic generation of time efficient digital signal processing software", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-25, February 1977, pp. 74-78.

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

[109] H. Nawab and J.H. McClellan, "Bounds on the minimum number of data transfers in WFTA and FFT programs", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-27, August 1979, pp. 394-398.

[110] V.Y. Pan, "The additive and logical complexities of linear and bilinear arithmetic algorithms", *J. Algorithms*, Vol. 4, No. 1, March 1983, pp. 1-34.

[111] J.H. Rothweiler, "Implementation of the in-order prime factor transform for variable sizes" *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-30, No. 1, February 1982, pp. 105-107

[112] H.F. Silverman, "An introduction to programming the Winograd Fourier transform algorithm", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-25, No. 2, April 1977, pp. 152-165, with corrections in: *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-26, No. 3, June 1978, p. 268, and in Vol. ASSP-26, No. 5, October 1978, p. 482.

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

[114] C.D. Thompson, "Fourier transforms in VLSI", *IEEE Trans. Comput.*, Vol. 32, No. 11, November 1983, pp. 1047-1057.

[115] M. Vetterli and A. Ligtenberg, "A discrete Fourier-cosine transform chip", *IEEE J. Selected Areas in Communications*, Special Issue on VLSI in Telecommunications. Vol. SAC-4, No. 1, January 1986, pp. 49-61.

[116] J. Vuillemin, "A combinatorial limit to the computing power of VLSI circuits", *Proc. 21st Symp. Foundations of Comput. Sci.*, IEEE Comp. Soc., October 1980, pp. 294-300.

[117] P.D. Welch, "A fixed-point fast Fourier transform error analysis", *IEEE Trans. Audio Electro.*, Vol. 15, No. 2, June 1969, pp. 70-73 (reprinted in [13] and [15]).

Software

FORTRAN (or DSP) code can be found in the following references.

[7] contains a set of classical FFT algorithms.
[111] contains a prime factor FFT program.
[4] contains a set of classical programs and considerations on program optimization, as well as TMS 32010 code.
[113] contains a compact split-radix Fortran program.
[29] contains a speed-optimized split-radix FFT.
[77] contains a set of real-valued FFTs with twiddle factors.
[65] contains a split-radix real valued FFT, as well as a Hartley transform program.

[112], as well as [7] contains a Winograd Fourier transform Fortran program.

[66], [67] and [75] contain improved bit-reversal algorithms.