Generalized autosort FFT framework

Michał Lenarczyk*

Instytut Podstaw Informatyki PAN

Abstract

In this article, the design of a mixed radix fast Fourier transform (FFT) algorithm is presented. The algorithm is general enough to be able to perform both decimation in time and decimation in frequency decompositions, and has some other advantages. It is argued it best fits signal processors, mainly because of small code footprint and efficient use of modulo addressing modes available on some DSP architectures.

1 Introduction

Among many methods for efficient computation of the discrete Fourier transform, DFT, the algorithm known as the fast Fourier transform is of greatest importance. Given data of composite size, MN, the discrete Fourier transform of the data can be reexpressed in terms DFT's of orders M and N repeated N and M times, respectively, which requires evaluation of $MN(M+N)$ terms as opposed to $(MN)^2$ when using direct formula, and this count can be further reduced by factoring M and/or N into even smaller numbers. The factorization can be done in arbitrary way, but two cases are especially useful : these are referred to as "decimation in time" and "decimation in frequency". Thus, the FFT computes the transform of composite order by a sequence of smaller transforms whose sizes are its divisors.

This approach became widely used after the famous publication of Cooley and Tukey [1] who obtained the decimation in time method for arbitrary factorizations of the transform order (mixed radix), giving special attention to the case where the order is a power of 2 (*i.e.*, radix 2). Gentleman and Sande [2] found the alternative, decimation in frequency form and also showed how reordering of data can be avoided given auxiliary storage, which is the first reported autosort framework. A particularly efficient algorithm known as "split radix" was proposed by Duhammel and Hollman [3] which is present state-of-the-art for orders containing factors 4. Of particular research interest is the adaptation of the FFT for efficient execution in vector and distributed processing architectures. More details on this subject can be found, *e.g.* in the handbook by Van Loan [9].

FFT efficiency has traditionally been measured in terms of arithmetic operations involved in its execution - the total number of multiplications, of multiplications and additions, or the greater of the two counts (see, for instance, [4]). It is

*The algorithm was developed in the time when the author was with AGH University of Science and Technology, Kraków meaningful when the cost of these operations is dominant, which was the case in early computer architectures where especially multiplications were time consuming, and is still applicable to digital signal processors which are able to perform arithmetic calculations in parallel with other operations such as memory access, looping / execution flow, and pointer updates, which effectively come at no cost. However, in most modern general purpose architectures with slow memory and fast ALU and cache, structural overhead and memory access present a nonnegligible cost, and counting arithmetic operations may not reflect actual performance.

Recently, gains in performance have been achieved mainly by the use of larger DFT modules that can be highly optimized by hand coding. This can be viewed as roughly equivalent to loop unrolling. Decomposition into larger radices that need not be prime, has the advantage of having fewer stages, but the resulting programs tend to be large. For example, the popular software package created by Frigo and Johnson [5] uses "small" DFT modules of orders up to 64 using straightforward code with no loop structure. A similar approach led Van Buskirk [6] to improve the code of split radix and reduce the number of required multiplications, which is so far the lowest known count (more details on the method, which does not require unrolling, can be found in [7]).

While the pursuit of the lowest possible count of additions or multiplications is interesting from theoretical point of view, the approach of using large, optimized modules is neither practical nor well suited for implementation on signal processors (DSP's). Firstly, these processors usually are limited in terms of program memory ; secondly, DSP's have very low structural overhead (typically in the range 3-15% [8]), which questions the sense of loop unrolling. For example, implementation of the single 64-point DFT code with Van Buskirk's optimization requires 912 additions and 240 multiplications, the amount that exceeds the capacity of small DSP's with only a few kilowords of program memory. Even for devices with large internal memory, its amount puts a practical limit on the number of elementary DFT modules of only a few smallest orders. On the other hand, various signal processing architectures provide specific features that allow writing efficient FFT programs that are compactly structured as well.

This article introduces a general, out-of-place, mixed-radix framework for FFT that combines autosort property with the ability to incorporate linear time and frequency shifts, which can be useful e.g. for centering the spectrum or for phase rectification. The generality of the algorithm allows it to be configured both in the decimation in time and decimation in frequency manner. A link is made to practical machine implementation issues with emphasis on DSP architectures ; in particular, a novel normalization scheme is proposed for fixed-point arithmetic. The algorithm is accompanied with a proof of concept implementation in C for verification and preliminary analyses. The code has a very small footprint, as it only contains a single generic version of DFT of arbitrary order, plus an optional routine for setting up the FFT structure at run time. When implemented in native assembly, the algorithm could complement library support of some DSP families that only supply code for FFT of radix 2.

It should be pointed out that DFT with shifted indices was first proposed by Bongiovanni *et al.* [10] who termed it "generalized discrete Fourier transform" ; quite recently, generalization to nonlinear phase DFT was proposed (see Akansu and Agirman-Tosun [11]) but is, however, not as yet permitted in this framework.

The derivation for the algorithm and analysis of its properties is presented in the next section. Section 3 considers practical implementation aspects. Finally, some concluding remarks are stated in section 4.

2 Theory and interpretation

This section focuses on how the FFT is obtained and discusses some of its properties, starting with facts well known from literature. After deriving the basic decomposition and its forms in the first paragraph, data arrangement and ordering is studied in paragraph 2.2 where an intuitive three-dimensional array interpretation is presented. Generalization of the DFT is introduced in paragraph 2.3 that allows to incorporate linear time and frequency shift into the body of the FFT, and in particular, to avoid explicit multiplication by twiddle factors. The last paragraph 2.4 considers scaling issues in the context of fixed-point arithmetic and proposes a novel approach to this question.

2.1 Derivation of the FFT

Given a signal of composite size MN, the discrete Fourier transform sum

$$\hat{x}_k = \sum_{n=0}^{MN-1} x_n \omega^{nk} \tag{1}$$

where $\omega^{nk} = e^{-i\frac{2\pi}{MN}nj}$ can, by substitution of $Nm+n$ and $Mk+j$ for original n and k, respectively, be expressed as a double summation :

$$\begin{aligned} \hat{x}_{Mk+j} &= \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} x_{Nm+n}\omega^{(Nm+n)(Mk+j)} \\ &= \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} x_{Nm+n}\omega^{Nmj}\omega^{nj}\omega^{Mnk} \end{aligned} \tag{2}$$

The innermost sum of the terms $x_{Nm+n}\omega^{Nmj}$ can be recognized as an M-point DFT over the decimated signal with argument m and offset parameter n. The outer summation is an N-point DFT over the outcome of the former transform multiplied by trigonometric phase shift coefficients ω^{nj} which are customarily referred to as *twiddle factors*, a term introduced by Gentleman and Sande [2]. Each DFT sequence spanning every possible value of the parameter which is not the index of summation will be referred to as DFT pass. The count of terms included in the double summation can be estimated as M^2N for the inner and MN^2 for the outer pass, which gives the overall formula $MN(M+N)$ for the total number of terms in both DFT passes. Considering the number of multiplications, the one-time scaling by twiddle factors requires additionally MN complex multiplications, which results in a total count of $MN(M+N+1)$ complex multiplications. The increase can be avoided if the twiddle factors are absorbed in the computation of either of the two DFT's. This idea is elaborated in paragraph 2.3 of this section.

If either of the numbers M or N is composite, further decomposition is possible and highly composite orders can be decomposed into prime factors in many ways. However, if at every step, the decomposition is made such that one selected dimension is always prime (or a number accepted as radix, *e.g.* 4), then an especially regular structure of the algorithm is obtained. If, at every step, M is prime (or non-decomposable) then the *decimation in time* results (the name reflecting the fact that decimated time index m and contiguous frequency index j are split into pairs of new indices in next pass) ; on the other hand, if the same is assured for N, then the *decimation in frequency* is obtained (decimated frequency index k and contiguous time index n are split further) - following the naming convention introduced in [12]. The resulting forms of the equation are equivalent ; the distinction concerns the association of the twiddle factors with either of the DFT passes. In case of decimation in time, multiplication by twiddle factors affects input before running the next DFT pass, whereas in decimation in frequency, the output of the DFT pass is multiplied. Apart from having some impact on implementation, the two approaches are identical.

In the following we consider the problem of composite order $N_0 = \prod_{l=1}^{L} M_l$ with radix M_l in pass l and set $N_k = \prod_{l=k+1}^{L} M_l$ so that the factorization at l-th level of decomposition is $N_{l-1} = M_l N_l$. It can be shown that in this case the overall complexity reduces to $N_0 \times \sum_l M_l = (\prod_l M_l) \times (\sum_l M_l)$ [1].

Following this convention, the first step of decomposition is $N_0 = M_1 N_1$ and for decimation in time the following index split is performed

$$\begin{aligned} n_0 &= M_1 n_1 + m_1, \quad n_1 = 0, \ldots, N_1 - 1, m_1 = 0, \ldots, M_1 - 1 \\ k_0 &= N_1 j_1 + k_1, \qquad k_1 = 0, \ldots, N_1 - 1, j_1 = 0, \ldots, M_1 - 1 \end{aligned} \tag{3}$$

or, for arbitrary level of decomposition l

$$\begin{aligned} n_{l-1} &= M_l n_l + m_l, \quad n_l = 0, \ldots, N_l - 1, m_l = 0, \ldots, M_l - 1 \\ k_{l-1} &= N_l j_l + k_l, \qquad k_l = 0, \ldots, N_l - 1, j_l = 0, \ldots, M_l - 1 \end{aligned} \tag{4}$$

which yields the formula for the first stage of FFT as follows :

$$\hat{x}_{N_1 j_1 + k_1} = \sum_{m_1=0}^{M_1-1} \sum_{n_1=0}^{N_1-1} x_{M_1 n_1 + m_1} \omega^{M_1 n_1 k_1} \omega^{m_1 k_1} \omega^{N_1 m_1 j_1} \tag{5}$$

Note the subscripts of indices m, n, j and k in (3) and (4) refer to the level of decomposition. For decimation in frequency, the following index split is applied :

$$\begin{aligned} n_0 &= N_1 m_1 + n_1, \quad n_1 = 0, \ldots, N_1 - 1, m_1 = 0, \ldots, M_1 - 1 \\ k_0 &= M_1 k_1 + j_1, \qquad k_1 = 0, \ldots, N_1 - 1, j_1 = 0, \ldots, M_1 - 1 \end{aligned} \tag{6}$$

and, generally,

$$\begin{aligned} n_{l-1} &= N_l m_l + n_l, \quad n_l = 0, \ldots, N_l - 1, m_l = 0, \ldots, M_l - 1 \\ k_{l-1} &= M_l k_l + j_l, \qquad k_l = 0, \ldots, N_l - 1, j_l = 0, \ldots, M_l - 1 \end{aligned} \tag{7}$$

leading to the following form of the first stage of FFT :

$$\hat{x}_{M_1 k_1 + j_1} = \sum_{n_1=0}^{N_1-1} \sum_{m_1=0}^{M_1-1} x_{N_1 m_1 + n_1} \omega^{N_1 m_1 j_1} \omega^{n_1 j_1} \omega^{M_1 n_1 k_1} \tag{8}$$

2.2 Arrangement and indexing of data

The index split introduced in the previous section can be interpreted in terms of arrangement of the input array in two dimensions. Let the prime symbol $'$ denote the operation of forming an array with two index variables replacing the single index of the original sequence such that, for input array x :

$$x'_{(n_1,m_1)} = x_{M_1 n_1 + m_1} \tag{9}$$

Then the double summation formula in equation 5 (DIT form) becomes

$$\sum_{m_1=0}^{M_1-1} \sum_{n_1=0}^{N_1-1} x'_{(n_1,m_1)} \omega^{M_1 n_1 k_1} \omega^{m_1 k_1} \omega^{N_1 m_1 j_1} \tag{10}$$

which is equivalent to a two-dimensional discrete Fourier transform over an $N \times M$ array, except for the intervening twiddle factors. Recall that in decimation in time, n is the decimated (by N) and m the contiguous index. From (5) it can be observed that the corresponding output indices are used in reverse order : k is contiguous and j decimated (by M). Therefore, the output can be reinterpreted as a $M \times N$ array $\hat{x}'$ with the following indexing :

$$\hat{x}'_{(j_1,k_1)} = \hat{x}_{N_1 j_1 + k_1} \tag{11}$$

Consequently, one step of DFT factorization transposes data interpreted as a two dimensional array.

Consider now the second level of decomposition according to decimation in time approach. By posing $n_1 = M_2 n_2 + m_2$ and $k_1 = N_2 j_2 + k_2$ it is possible to define x'', a three-dimensional array indexed $x''_{(n_2,m_2,m_1)} = x_{M_1 M_2 n_2 + M_1 m_2 + m_1}$ and $\hat{x}''$, indexed $\hat{x}''_{(j_1,j_2,k_2)} = \hat{x}_{N_1 j_1 + N_2 j_2 + k_2}$. It is no longer possible to recover the output by a simple transposition : the result of two level decomposition is a superposition of two transpositions applied to different array formats.

A full decomposition into L factors would yield L-dimensional input array $x^{(L)}$ indexed with an addressing sequence

$$(m_L, m_{L-1}, \ldots, m_2, m_1) \tag{12}$$

corresponding to the unidimensional data index $M_1 M_2 \cdots M_{L-1} m_L + \ldots + M_1 m_2 + m_1$, and $\hat{x}^{(L)}$, addressed

$$(j_1, j_2, \ldots, j_{L-1}, j_L) \tag{13}$$

corresponding to $N_1 j_1 + N_2 j_2 + \ldots + N_{L-1} j_{L-1} + j_L$. The addressing sequence of the input and output are reversed, which can be accomplished in two ways.

Cooley's in-place algorithm [1] did not change index order before and after each pass, and required reordering of data samples, either before or after execution of the algorithm. The special case he considered where the radix is 2 can efficiently be done on binary machines that allow reversal of bits (some processors even feature reverse-carry addressing mode which obviate bit reversal).

Sande [2] realized the possibility of absorbing the transpositions in the course of subsequent DFT passes, to undo the effect of decompositions, at the cost of auxiliary storage. The algorithm presented in this paper follows the same *autosort* approach, since, being inherently out-of-place, it immediately satisfies the requirement for additional storage.

Consider the first level of decimation in time decomposition, and define the intermediate array y', such that :

$$\hat{x}'_{(j_1,k_1)} = \sum_{m_1=0}^{M_1-1} y'_{(k_1,m_1)} \omega^{m_1 k_1} \omega^{N_1 m_1 j_1} \tag{14}$$

and

$$y'_{(k_1,m_1)} = \sum_{n_1=0}^{N_1-1} x'_{(n_1,m_1)} \omega^{M_1 n_1 k_1} \tag{15}$$

In the above decomposition, the output y' of the inner DFT sum has the same addressing as input, whereas the outer DFT performs transposition according to

$$(k_1, m_1) \to (j_1, k_1)$$

In the second level of factorization, the intermediate array y' from (15) is replaced by a new 3-dimensional array y'' as follows :

$$y''_{(j_2,k_2,m_1)} = \sum_{m_2=0}^{M_2-1} z''_{(k_2,m_2,m_1)} \omega^{M_1 m_2 k_2} \omega^{M_1 N_2 m_2 j_2} \tag{16}$$

$$z''_{(k_2,m_2,m_1)} = \sum_{n_2=0}^{N_2-1} x''_{(n_2,m_2,m_1)} \omega^{M_1 M_2 n_2 k_2} \tag{17}$$

where a new intermediate array z'' is introduced and again the inner DFT does not change the addressing and the outer DFT transposes the output according to :

$$(k_2, m_2, m_1) \to (j_2, k_2, m_1)$$

By following the above procedure, a full decomposition defined by the input addressing (12) and output addressing (13) would be obtained. It is not necessary, however, to expand the data into L dimensions. By considering the addressing of intermediate results of consecutive DFT passes from (12) to (13),

$$\begin{array}{cccccccc}
(& m_L, & m_{L-1}, & m_{L-2}, & \dots, & m_2, & m_1 &) \\
(& j_L, & m_{L-1}, & m_{L-2}, & \dots, & m_2, & m_1 &) \\
(& j_{L-1}, & j_L, & m_{L-2}, & \dots, & m_2, & m_1 &) \\
(& j_{L-2}, & j_{L-1}, & j_L, & \dots, & m_2, & m_1 &) \\
 & \vdots & \vdots & \vdots & & \vdots & \vdots & \\
(& j_1, & j_2, & j_3, & \dots, & j_{L-1}, & j_L &)
\end{array}$$

it can be observed that, at every pass, the address sequence consists of indices of three kinds : the already transformed indices j, the as yet untransformed indices m and the single index m_l that is subject to transformation in the present pass l.

Therefore, it suffices to use three indices for every pass. Denote by n the currently transformed index m_l, and collapse all remaining indices m into a single combined index $\vec{m}$ and all j into $\vec{j}$ (either of which can be empty, namely, $\vec{j}$ is an empty sequence in the first pass and $\vec{m}$ is empty in the last pass). Then, the following transformation of addressing sequence is applied along with DFT computation :

$$(\vec{j}, n, \vec{m}) \to (k, \vec{j}, \vec{m})$$

This leads to an elegant interpretation of the transform in terms of three-dimensional arrays with indices j, n and m. The dimensions must be defined anew in every DFT pass ; the index k from the previous pass is fused into the index $\vec{j}$ and a new n is extracted from the untransformed index sequence $\vec{m}$.

As a final remark, all the above results remain valid for decimation in frequency decomposition, except that the sequence of indices of the input and output (as well as any intermediate output from any DFT pass) is reversed. This case is depicted on the "butterfly" diagram of Fig. 1, and on Figs. 2–5, which illustrate shape transformations carried out in each DFT pass.

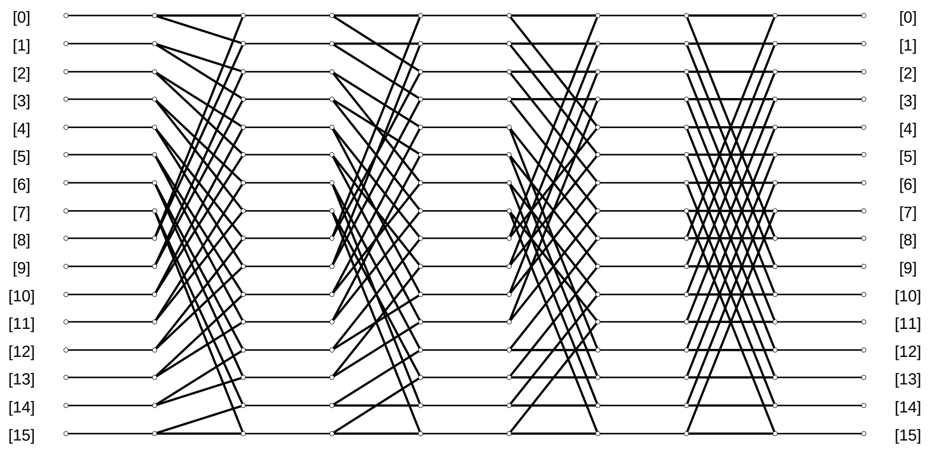


Figure 1: Autosort FFT flow diagram for $N_0 = 16$. Weight labels are omitted for clarity. Note the irregular butterfly structure of the first three passes due to transposition of output. The last pass is computed without transposition and hence its regular shape.

2.3 Generalized transform framework

Because twiddle factors only contribute insignificant amount of computation, they are usually applied as a separate stage between DFT passes, permitting deep optimization of DFT modules which are the core of the algorithm [5]. The algorithm discussed is deliberately not optimized for selected radix values but made as general as possible. As a result, the twiddle coefficients are not applied as a separate

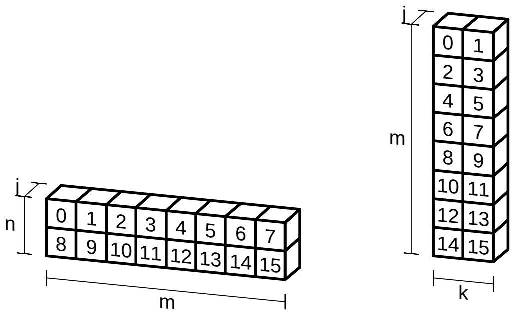


Figure 2: FFT for $N_0 = 16$, pass 1 (left : input, right : transposed output). In all cases, the transform is applied along the decimating vertical index n and results in horizontal output index k.

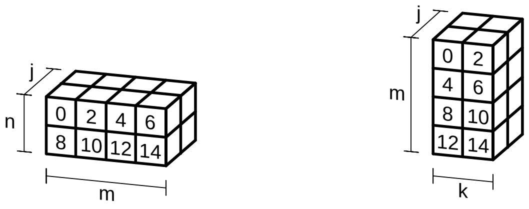


Figure 3: FFT for $N_0 = 16$, pass 2 (left : input, right : transposed output)

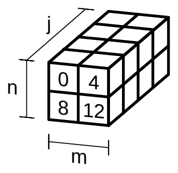
scaling stage but incorporated into the trigonometric coefficients of one of the neighbouring DFT passes. Again, it is possible to merge the coefficients either into the inner or the outer loop. In what follows we focus on the latter case, the former being analogous.

Let us reexpress formula (14) as follows :

$$\hat{x}'_{(j_1,k_1)} = \sum_{m_1=0}^{M_1-1} y'_{(k_1,m_1)} \omega^{m_1(N_1 j_1 + k_1)} \tag{18}$$

The frequency index k_1 which previously served as a replication parameter now becomes a variable of the DFT pass, affecting the phase of the twiddle coefficients applied in each DFT in a pass.

It is possible to absorb the twiddle factors into the body of inner DFT pass if we consider a more general form of the DFT equation (which was introduced



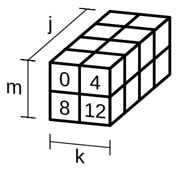
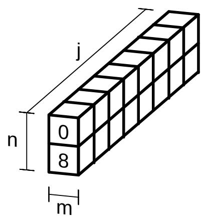


Figure 4: FFT for $N_0 = 16$, pass 3 (left : input, right : transposed output)



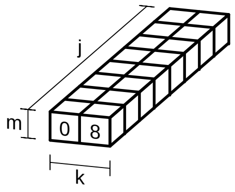


Figure 5: FFT for $N_0 = 16$, last pass (left : input, right : transposed output). Note in this case that the transposition does not affect sample ordering, which is consistent with the shape of the last pass in Fig. 1.

in [10] and termed GFT, for "generalized discrete Fourier transform") :

$$\hat{x}_k = \sum_{n=0}^{N-1} x_n \omega^{(n+P)(k+Q)} = \sum_{n=0}^{N-1} x_n \omega^{nk+Qn+Pk+PQ} \tag{19}$$

By the periodicity property of the signal and its spectrum, the above formula can easily be verified to be equivalent to

$$\hat{x}_{k-Q} = \sum_{n=0}^{N-1} x_{n-P} \omega^{nk} \tag{20}$$

which leads to interpretation of parameters P and Q as the time and the frequency shift, respectively.

Equation 20 has the disadvantage that index shifted by a nonzero value exceeds the usual range of definition $1 \ldots N-1$. Implementation of periodic signals that are indexed in a range exceeding one period requires modulo addressing, to confine the index to the range where storage for samples is defined. Modulo adressing is assisted by hardware in some machines but most often requires software implementation, incurring some cost of execution time.

In what follows we turn to the form of equation 19 where x and $\hat{x}$ are addressed linearly. Considering the powers of $\omega = e^{-i2\pi/N_0}$, we easily note the periodicity $\omega^{N_0+n} = \omega^n$ which permits use of tabulated values of constants $\omega_n = \omega^n$ for $n = 1 \ldots N_0 - 1$ in place of calculating them during execution of the algorithm, which is the most common approach. Access to ω constants must then be performed modulo N_0.

Let us begin with the initial DFT equation involving time shift P and frequency shift Q, focusing only on the form of ω factors :

$$\omega^{(n_0+P)(k_0+Q)} = \omega^{n_0k_0+Qn_0+Pk_0+PQ} \tag{21}$$

Applying first level of decomposition (3) yields

$$\omega^{(M_1n_1+m_1+P)(N_1j_1+k_1+Q)} = \omega^{M_1n_1(k_1+Q)} \times \omega^{(m_1+P)(N_1j_1+k_1+Q)} \tag{22}$$

a formula for trigonometric constants applied : $\omega^{M_1n_1(k_1+Q)}$ for the inner DFT pass and $\omega^{(m_1+P)(N_1j_1+k_1+Q)}$ for the outer DFT pass. It is worthwile to observe that, given arbitray initial shifts P and Q, only frequency shift is propagated to the next (inner) level of decomposition, the time shift being absorbed in the outer DFT which is the last pass of the overall decomposition. Recursively applying the above decomposition, a structure composed only of DFT with time and frequency shift is obtained, and no intervening twiddle stage is required. It should be noted that the form of (22) is not a unique solution of decomposition (5) and many equivalent splits can be considerd which could be a subject of a separate study.

2.4 Scaling

The classical definition of the discrete Fourier transform pair is

$$\hat{x}_k = \sum_{n=0}^{N-1} x_n \omega^{nk} \leftrightarrow x_k = \frac{1}{N} \sum_{n=0}^{N-1} \hat{x}_n \omega^{-nk}$$

For practical implementation in fixed point arithmetic, it is necessary to include the normalizing factor N^{-1} in the forward rather than inverse formula, so that input of samples from a fixed interval, *e.g.* $(-1, +1)$, yields output bounded to the same range. Additionally, this scaling must be performed incrementally during the computation to prevent exceeding the maximum range at every stage.

Some signal processors have built-in scaling modes, allowing data to be scaled up or down during load or store by a factor of 2 through a simple bit shift. This proves particularly helpful for radix 2 fast Fourier algorithms allowing scaling to be effectuated within every pass. Good noise properties are achieved because intermediate results can make use of the full range of values allowed by the size of data words so that impact of digit truncation is minimal.

A similar feature is difficult to implement in the general mixed radix setup where each pass can be a DFT of a different order, and hence the normalizing factor can be arbitrary. Applying the scaling by the current DFT order would cost an additional real-complex multiplication of all values in every pass.

The algorithm described in this article has the property that in every DFT pass, all samples are multiplied by trigonometric factors exactly once. This leads to the observation that every output sample is a result of a process composed of L consecutive multiplications by some power of ω, where $|\omega| = 1$. To take advantage of this fact, a new vector of scaled trigonometric constants is defined :

$$\omega_S = N^{-\frac{1}{L}} \times \omega$$

The scaling can be verified to produce, after L passes, an overall factor of

$$|\omega_S|^L = \left|\left(N^{-\frac{1}{L}} \times \omega\right)\right|^L = N^{-1} \times |\omega| = \frac{1}{N}$$

Since $N = \prod_l M_l$, the constant $N^{\frac{1}{L}}$ is the geometric mean of primary orders M_l. It is easy to verify that, by chosing, in consecutive passes, DFT orders M_l from smallest to largest, the property $\left|\sum_n^{M_l} x_n \omega_S^{nk}\right| \leq 1$ for $|x_n| \leq 1$ is assured.

The choice of the the above scaling approach is a compromise between rescaling the input by N^{-1} once before execution (not acceptable due to excessive loss in signal to noise ratio) and scaling in each pass by M_l^{-1} which allows tight match of the range $(-1, +1)$ for output at the cost of additional complexity. It can be seen that when the transform order is a product of factors of very different magnitude, the factor $N^{\frac{1}{L}}$ is significantly smaller than 1 causing dynamic range compaction when small DFT's are executed in early passes and a degradation of signal to noise ratio. This fact must be taken into consideration when selecting the order of the transform. In most cases, the order can be decomposed into many small primes, in which case the algorithm should assure a good noise-performance balance.

3 Description of algorithm

This section provides details important for implementation. The algorithm has the following properties :

1. general mixed radix, out-of-place structure with one scratch array which is used together with the target output array on an alternating basis
2. employs the generalized framework described in section 2.3, allowing computation of FFT with time and frequency shift, and execution of both decimation in time and decimation in frequency structures
3. allows (but does not require) execution of autosort algorithm whereby the intermediate results are stored with transposition, to yield final result in natural order
4. has an optional scaling mode allowing the implementation in fixed range arithmetic

In the first paragraph the structure of the algorithm is described and in the second some preliminary performance metrics invoked.

3.1 Structure

The algorithm has a standard structure of four nested loops in each pass. They are discussed beginning with the innermost loop.

Loop over n **:** This loop calculates the scalar product $\sum_{n=0}^{N-1} x_{a(n)}\omega_{b(n)}$, where addressing $a(n) = K_x n + L_x$ is linear and $b(n) = K_\omega n + L_\omega$ circular (modulo N_0). Parameters $K_x, L_x, K_\omega, L_\omega$ are provided from the containing loop.

Loop over k **:** The outcome of the previous loop is a single Fourier coefficient, and execution over $k = 1 \ldots N-1$ yields the full spectrum. The result is assigned to output $\hat{x}_{c(k)}$ with linear addressing $c(k) = K_{\hat{x}}k + L_{\hat{x}}$. In every pass of the loop, the parameters passed to the inner loop K_x, L_x are supplied from the outer loop and K_ω, L_ω are found according to

$$\begin{cases} K_\omega(k) &= (K_{K,\omega}k + L_{K,\omega}) \mod N_0 \\ L_\omega(k) &= (K_{L,\omega}k + L_{L,\omega}) \mod N_0 \end{cases}$$

Loop over j **:** Here the index j spans the already transformed dimensions of the full decomposition as discussed in paragraph 2.2. The single DFT defined by the loop over k is repeated for every value of j with parameters found according to

$$\begin{cases} K_x(j) &= K_{K,x}j + L_{K,x} \\ L_x(j) &= K_{L,x}j + L_{L,x} \\ K_{\hat{x}}(j) &= K_{K,\hat{x}}j + L_{K,\hat{x}} \\ L_{\hat{x}}(j) &= K_{L,\hat{x}}j + L_{L,\hat{x}} \\ K_{K,\omega}(j) &= (K_{K,K,\omega}j + L_{K,K,\omega}) \mod N_0 \\ K_{L,\omega}(j) &= (K_{K,L,\omega}j + L_{K,L,\omega}) \mod N_0 \\ L_{K,\omega}(j) &= (K_{L,K,\omega}j + L_{L,K,\omega}) \mod N_0 \\ L_{L,\omega}(j) &= (K_{L,L,\omega}j + L_{L,L,\omega}) \mod N_0 \end{cases}$$

Loop over m **:** In this loop, m is the replication index for which the same structure (in terms of trigonometric constants) is applied to successive slices of the data interpreted as array in three dimensions. Therefore, the index only affects the addressing of inputs and outputs as follows :

$$\begin{cases} K_{K,x}(m) &= K_{K,K,x}m + L_{K,K,x} \\ K_{L,x}(m) &= K_{K,L,x}m + L_{K,L,x} \\ L_{K,x}(m) &= K_{L,K,x}m + L_{L,K,x} \\ L_{L,x}(m) &= K_{L,L,x}m + L_{L,L,x} \\ K_{K,\hat{x}}(m) &= K_{K,K,\hat{x}}m + L_{K,K,\hat{x}} \\ K_{L,\hat{x}}(m) &= K_{K,L,\hat{x}}m + L_{K,L,\hat{x}} \\ L_{K,\hat{x}}(m) &= K_{L,K,\hat{x}}m + L_{L,K,\hat{x}} \\ L_{L,\hat{x}}(m) &= K_{L,L,\hat{x}}m + L_{L,L,\hat{x}} \end{cases}$$

Execution of pass l : The master loop executes the sequence of passes and uses tabulated values of the parameters defined above (there are 8 parameters defined per each of the three arrays $x, \hat{x}, \omega$). A separate routine is used to precompute the parameters according to the chosen structure. For example, the following choice of parameters is used to obtain a decimation in frequency, autosort structure for arbitrary factorization $\{M_l, l = 0, \ldots, L\}$:

$$\begin{array}{ccc}
K_{K,K,x} = 0 & K_{K,K,\hat{x}} = 0 & K_{K,K,\omega} = 0 \\
K_{K,L,x} = 0 & K_{K,L,\hat{x}} = 0 & K_{K,L,\omega} = \prod_{r=1}^{l-1} M_r \\
K_{L,K,x} = 0 & K_{L,K,\hat{x}} = 0 & K_{L,K,\omega} = 0 \\
K_{L,L,x} = 1 & K_{L,L,\hat{x}} = 1 & K_{L,L,\omega} = 0 \\
L_{K,K,x} = 0 & L_{K,K,\hat{x}} = 0 & L_{K,K,\omega} = N_0/M_l \\
L_{K,L,x} = \prod_{r=1}^{l-1} M_r & L_{K,L,\hat{x}} = N_0/M_l & L_{K,L,\omega} = 0 \\
L_{L,K,x} = N_0/M_l & L_{L,K,\hat{x}} = \prod_{r=1}^{l-1} M_r & L_{L,K,\omega} = 0 \\
L_{L,L,x} = 0 & L_{L,L,\hat{x}} = 0 & L_{L,L,\omega} = 0
\end{array}$$

for all except L-th pass, where the parameter $K_{K,L,\omega}$ is set to zero and the remaining parameters are the same.

If time and frequency offset are desired, then the parameters for ω should be changed according to the formulas from paragraph 2.3. In general, however, the parameters as found using the formulas in this paper lead to only some of the parameters that are nonzero. Since these parameters are determined uniquely, there exists a potential for further generalization, which can be an interesting subject of study. One seemingly achievable result is Good's prime factor algorithm, but for this, a modification of the implementation is required to allow modulo addressing of input and output arrays.

3.2 Complexity

The algorithm, derived and implemented in its full generality, compares infavorably with highly optimized algorithms like the popular FFTW library [5] when speed is considered. Preliminary implementation tests executed on an Intel x86 machine in a single processor core show that the FFTW is 3 to 6 times faster (see Table 1), the difference being smallest for factorizations into large primes (combinations of 17, 19 and 23 were considered) which it executes using generic routine and largest for orders being powers of 2 which are performed by highly optimized modules. This is not surprising, given the generalized structure in which all multiplications are performed directly regardless of whether they are trivial or not. On general purpose architectures (such as the Intel x86), all operations involved in the algorithm contribute to the exection time.

Table 2 shows the breakdown of the algorithm into elementary operations involved in its execution, per each of the loops from the innermost (over n) to the outermost (over m) that make up one pass.

To analyse the workload it is sufficient to consider the innermost loop, over n, which has a dominant impact. The loop iterates M_l times ; the containing loop over k also iterates M_l times and the two outer loops in combination make N_0/M_l repetitions, which makes a total of $N_0 \times M_l$ iterations of the innermost loop per

Table 1: Comparison of execution times of proposed algorithm and FFTW. Mean times and standard errors measured in microseconds. Observed measurement variability is due to processor speed stepping.

		Proposed		FFTW		
Size	Factorization	$\bar{t}$	SE t	$\bar{t}$	SE t	ratio
512	2^9	1055	5,8	177	2,9	**5,96**
1024	2^{10}	1519	731,8	284	122,8	**5,35**
2048	2^{11}	2150	3,5	385	2,7	**5,58**
323	$17 \cdot 19$	462	169,1	154	60,6	**2,99**
391	$17 \cdot 23$	1001	2,5	267	1,3	**3,75**
437	$19 \cdot 23$	1158	2,3	323	2,4	**3,59**
289	17^2	654	2,0	168	1,3	**3,9**
361	19^2	888	2,0	228	0,7	**3,89**
529	23^2	1552	148,1	412	0,7	**3,76**
7429	$17 \cdot 19 \cdot 23$	11679	24,9	3460	17,1	**3,38**
4913	17^3	6764	8,5	2155	8,1	**3,14**
6859	19^3	10471	13,9	3237	14,1	**3,23**
240	$2^4 \cdot 3 \cdot 5$	190	38,4	46	7,2	**4,11**
300	$2^2 \cdot 3 \cdot 5^2$	458	59,8	106	12,8	**4,31**
320	$2^6 \cdot 5$	585	5,1	112	0,1	**5,23**
350	$2 \cdot 5^2 \cdot 7$	554	1,4	156	0,6	**3,54**
400	$2^4 \cdot 5^2$	653	131,2	137	25,0	**4,77**
450	$2 \cdot 3^2 \cdot 5^2$	359	74,4	125	22,8	**2,88**
500	$2^2 \cdot 5^3$	741	106,0	185	23,5	**4,01**
600	$2^3 \cdot 3 \cdot 5^2$	1053	1,2	265	18,4	**3,98**
700	$2^2 \cdot 5^2 \cdot 7$	637	234,1	204	64,7	**3,12**
800	$2^5 \cdot 5^2$	860	255,0	199	50,1	**4,31**
900	$2^2 \cdot 3^2 \cdot 5^2$	1542	171,1	437	47,4	**3,53**
1000	$2^3 \cdot 5^3$	1171	514,1	320	132,8	**3,66**

Table 2: Operation counts per iteration

Operation	n-loop	k-loop	j-loop	m-loop
real multiply-accumulate	4	0	0	0
linear address update	1	1	4	8
circular address update	1	2	4	0
data word read	4	0	0	0
data word write	0	2	0	0

pass. In each iteration, a complex multiply-accumulate operation is performed, which amounts to 4 multiplies and accumulations, assuming real-imaginary representation of complex data stored in a single data word per each part. Consequently, four data reads are required to fetch the real and imaginary part of the input data and the trigonometric coefficients. In most DSP architectures, a data transfer can be performed in parallel with the multiply-accumulate operations ; sometimes, it is even possible to perform two reads or a double/quad word read. It is also common in DSP to be able to offset data pointers in parallel with arithmetic operation and data read. If a given architecture features modulo addressing, the circular address can be updated with no cost, otherwise the range check and possible index wrap will contribute to the execution time. The result is written back after the inner loop completes the calculation of the Fourier coefficient and is done in the body of the k-loop.

From the above it is clear that the execution time is determined by arithmetic instructions. The algorithm's speed could be improved by reducing their number. FFTW library uses a technique involving discrete Hartley transform to halve the number of multiplications required by computing two coefficients at a time. This partly accounts for the difference in execution speed between the library and the implementation of the presented algorithm ; an additional difference comes from the fact that FFTW avoids the problem of addressing trigonometric constants by expanding the table of all nontrivial coefficients at the cost of $(M_l - 1)^2$ memory words per pass l. In the algorithm presented, the generalized configuration in which ω factors can occur in any power prevents direct application of the mentioned technique which requires deeper study. Some known improvements, such as for real, real-odd and real-even transforms could be applied directly ; others, however, are incompatible with the framework's mixed-radix and generalized design.

The major advantage of the proposed algorithm is its small code footprint. Compiled for Intel x86, object binaries have a size of less than 16 KB, whereas FFTW library by far exceeds 500 KB. This aspect may be of key importance in many cases.

4 Concluding remarks

The algorithm presented in this article is a generalized FFT framework which also has autosort capability. In its full generality, it is able to implement many of the structures and decompositions proposed in the past. The derivations presented are implemented in the form of a test program which is available from the author. The advantages of the algorithm are its simplicity (only a single version to execute all permitted structures), which translates into a small code footprint, and the flexibility it offers. However, preliminary tests show that the implementation is inferior in terms of speed when compared with a state of the art industry implementation. Some acceleration techniques could be considered but require further study.

A true test for the algorithm would be an attempt to run it in a DSP environment, where it could take advantage of circular addressing modes it makes extensive use of, and where structural overhead is small. The incremental scaling option which comes at no performance cost makes it directly applicable in the most typical case where fixed point arithmetic is required. It is interesting to see how this approach would perform from the perspective of noise - which was not considered in this paper because of the fact that floating point implementation is known [2] to behave in the opposite way, *i.e.*, direct formula implementation results in *higher* noise than the fast algotithm. The author plans to elaborate this aspect in the future.

Acknowledgement Study was supported by research fellowship within "Information technologies : research and its interdisciplinary applications" agreement number POKL.04.01.01-00-051/10-00.

References

[1] Cooley J. W., Tukey J. W., An algorithm for the machine computation of complex Fourier series, *Math. Computation* vol. 9, pp 297-301, 1965

[2] Gentleman W. M., Sande G., Fast Fourier transforms - for fun and profit, *Proc. AFIPS*, pp 563-578, 1966

[3] Duhamel P., Hollmann H., 'Split radix' FFT algorithm, *Electronics Letters* vol. 20 pp 14-16, 1984

[4] Stasiński R., Prime factor FFT for modern computers, *IWSSIP 2012* pp 346-349, 2012

[5] Frigo M., Johnson S. G., The design and implementation of FFTW3, *Proc. IEEE* vol. 93 no 2, pp 216-231, 2005

[6] Van Buskirk J. posted his results on comp.dsp newsgroup and the programs he wrote are available (as of 2012) at `http://home.comcast.net/~kmbtib/` and `http://www.cuttlefisharts.com/newfft/`

[7] Johnson S. G., Frigo M., A modified split-radix FFT with fewer arithmetic operations, *IEEE Trans. on Signal Processing*, vol. 55 no 1, pp 111-119, 2007

[8] Sohie G. R. L., Chen W., *Implementation of Fast Fourier Transforms on Motorola's Digital Signal Processors*, Motorola application note APR4/D, 1993

[9] Van Loan C., *Computational Frameworks for the Fast Fourier Transforms*, SIAM, 1992

[10] Bongiovanni G., Corsini P., Frosini G., One-dimensional and two-dimensional generalized discrete Fourier transform, *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. 24 no 1, pp 97-99, 1976

[11] Akansu A. N., Agirman-Tosun H., Generalized discrete Fourier transform with nonlinear phase, *IEEE Trans. on Signal Processing*, vol. 58 no 9, pp 4547-4556, 2010

[12] Cochran W. T., Cooley J. W., Favin D. L., Helms H. D., Kaenel R. A., Lang W. W., Maling G. C., Nelson D. E. , Rader C. M., Welch P. D., What is the fast Fourier transform ?, *Proc. IEEE*, vol. 55 no 10, pp 1664-1674, 1967