The FFT - an algorithm the whole family can use

Daniel N. Rockmore*
Departments of Mathematics and Computer Science
Dartmouth College
Hanover, NH 03755

October 6, 1999

"A paper by Cooley and Tukey [5] described a recipe for computing Fourier coefficients of a time series that used many fewer machine operations than did the straightforward procedure...What lies over the horizon in digital signal processing is anyone's guess, but I think it will surprise us all."[1]

1 Introduction

These days, it is almost beyond belief (at least for many of my students) that there was a time before digital technology. It seems almost everyone knows that somehow all the data whizzing over the internet, bustling through our modems or crashing into our cell phones is ultimately just a sequence of 0's and 1's – a digital sequence – that magically makes the world the convenient high speed place it is today. So much of this magic is due to a family of algorithms that collectively go by the name "The Fast Fourier Transform", or "FFT" to its friends, among which the version published by Cooley and Tukey [5] is the most famous. Indeed, the FFT is perhaps the most ubiquitous algorithm used today in the analysis and manipulation of digital or discrete data.

My own research experience with various flavors of the FFT is evidence of the wide range of applicability: electroacoustic music and audio signal processing, medical imaging, image processing, pattern recognition, computational chemistry, error correcting codes, spectral methods for PDEs and last but not least, in mathematics, as the starting point of my doctoral dissertation in computational harmonic analysis which investigated group theoretic generalizations of the Cooley-Tukey FFT. Of course many more could be listed, notably those to radar and communications. The book [2] is an excellent place to look, especially pages 2 and 3 which contain a (nonexhaustive) list of seventy-seven applications!

*Supported by NSF Presidential Faculty Fellowship DMS-9553134. Email: rockmore@cs.dartmouth.edu

[1] Bruce P. Bogert, Guest Editorial, *Special issue on fast Fourier transform and its application to digital filtering and spectral analysis*, IEEE Trans. Audio Electronics (1967), AU-15, No. 2, p. 43.

The FFT efficiently computes the discrete Fourier transform (DFT). Recall that the DFT of a complex input vector of length N, $X = (X(0)\ldots, X(N-1))$, denoted $\widehat{X}$, is another vector of length N given by the collection of sums

$$\widehat{X}(k) = \sum_{j=0}^{N-1} X(j) W_N^{jk} \tag{1}$$

where $W_N = \exp(2\pi\sqrt{-1}/N)$. Equivalently, this can be viewed as the matrix-vector product $W_N \cdot X$ where $W_N = ((W_N^{jk}))$ is the so-called **Fourier matrix**. The DFT is an invertible transform with inverse given by

$$X(j) = \frac{1}{N} \sum_{k=0}^{N-1} \widehat{X}(k)_j W_N^{-jk}. \tag{2}$$

Thus, if computed directly, the DFT would require N^2 operations. Instead, the FFT is an algorithm for computing the DFT in $O(N \log N)$ operations!

Note that the inverse can in fact be viewed as the DFT of the function $\frac{1}{N}\widehat{X}(-k)$, so that the FFT can also be used to invert the DFT.

One of the most useful properties of the DFT is that it converts circular or cyclic convolution into pointwise multiplication, i.e.,

$$\widehat{X \star Y}(k) = \widehat{X}(k)\widehat{Y}(k) \tag{3}$$

where

$$X \star Y(j) = \sum_{l=0}^{n} X(l) Y(j-l). \tag{4}$$

Consequently, the FFT gives an $O(N \log N)$ (versus N^2) algorithm for computing convolutions: First compute the DFTs of both X and Y, then compute the inverse DFT of the sequence obtained by multiplying pointwise $\widehat{X}$ and $\widehat{Y}$.

In retrospect, the idea underlying the Cooley-Tukey FFT is quite simple. If $N = N_1 N_2$, then we can turn the one-dimensional equation (1) into a two-dimensional equation via the change of variables

$$\begin{array}{lll} j = j(a,b) = aN_1 + b & 0 \le a < N_2 & 0 \le b < N_2 \\ k = k(c,d) = cN_2 + d & 0 \le c < N_2 & 0 \le d < N_1. \end{array} \tag{5}$$

Using the fact $W^{m+n} = W^m W^n$, it follows quickly from (5) that (1) can be rewritten as

$$\widehat{X}(c,d) = \sum_{b=0}^{N_2-1} W_N^{b(cN_2+d)} \sum_{a=0}^{N_1-1} X(a,b) W_{N_2}^{ad}. \tag{6}$$

The computation is now performed in two steps: first compute for each b the inner sums (for all d)

$$\tilde{X}(b,d) = \sum_{a=0}^{N_1-1} X(a,b) W_{N_2}^{ad}$$

which is now interpreted as a subsampled DFT of length N_2. Even if computed directly, at most $N_2 N_1^2$ arithmetic operations are required to compute all of the $\tilde{X}(b,d)$. Next, N_1 transforms of length N_2 are computed,

$$\sum_{b=0}^{N_2-1} W^{b(cN_2+d)} \tilde{X}(b,d).$$

Thus, instead of $(N_1N_2)^2$ operations, the above uses $(N_1N_2)(N_1+N_2)$ operations. If we had more factors in the expression (6), then this approach would work even better, giving Cooley and Tukey's result. The main idea is that we have converted a one-dimensional algorithm, in terms of indexing, into a two-dimensional algorithm. Furthermore, this algorithm has the advantage of an in-place implementation, and when accomplished this way, concludes with data reorganized according to the well-known bit-reversal shuffle.

This "decimation in time" approach is but one of a variety of FFT techniques. Also notable is the dual approach of "decimation in frequency" developed simultaneously by Sande, whose paper with Gentleman also contains a very interesting discussion of memory considerations as it relates to implementational issues [9]. The book [21] discusses some of the other variations and contains an extensive bibliography. Many of these algorithms rely on the ability to factor N. When N is prime, a different idea, due to Rader, is used in which the DFT is effectively reduced instead to a cyclic convolution [13].

2 History

The first appearance of the FFT, like so much of mathematics, can be traced back to Gauss [10]. His interests were in certain astronomical calculations (a recurrent area of application of the FFT) having to do with the interpolation of asteroidal orbits from a finite set of equally-spaced observations. Surely the prospect of a huge laborious hand calculation was good motivation for the development of a fast algorithm. Fewer calculations also implies less opportunity for error, and hence is also more numerically stable! Gauss observes that a Fourier series of bandwidth $N = N_1N_2$ can be broken up into a computation of the N_2 subsampled DFTs of length N_1 which are combined as N_1 DFTs of length N_2, precisely as Cooley and Tukey explain. Gauss's algorithm was never published outside of his collected works.

A less general, but still important version of the FFT, used for the efficient computation of the Hadamard and Walsh transforms was first published in 1932 by the statistician Yates [20]. Yates's "interaction algorithm" is a fast technique for computing the analysis of variance for a 2^n-factorial design, and is described in almost any text on statistical design and analysis of experiments.

Another important predecessor is the work of Danielson and Lanczos [7], performed in the service of x-ray crystallography, another frequent user of FFT technology. Their "doubling trick" showed how to reduce a DFT on $2N$ points to two DFTs on N points using only N extra operations. Today it brings a smile to our faces as we note their problem sizes and timings (see [7], part I, page 366): "Adopting these improvements the approximate times for Fourier analysis are 10 minutes for 8 coefficients, 25 minutes for 16 coefficients, 60 minutes for 32 coefficients, and 140 minutes for 64 coefficients." This indicates a running time of about $.37N \log N$ minutes for an N point DFT!

Despite these early discoveries of an FFT, it wasn't until Cooley and Tukey's article that the algorithm gained any notice. The story of their collaboration is an interesting one. Tukey arrived at the basic reduction while in a meeting of President Kennedy's Science Advisory Committee where among the topics of discussions were techniques for off-shore detection of nuclear tests in the Soviet Union. Ratification of a proposed United States/Soviet Union nuclear test ban depended upon the development of a method for detecting the tests without actually visiting the Soviet nuclear facilities. One idea was to analyze seismological time series obtained from off-shore seismometers, the length and number of which would require fast algorithms for computing the DFT. Other possible applications to national security included the long-range acoustic detection of nuclear submarines.

Richard Garwin of IBM was another of the participants at this meeting and when Tukey showed him this idea he immediately saw a wide range of potential applicability and quickly set to getting this algorithm implemented. He was directed to Cooley, and, needing to hide the national security issues, instead told Cooley that he wanted the code for another problem of interest: the determination of the periodicities of the spin orientations in a 3-D crystal of He^3. Cooley had other projects going on, and only after quite a lot of prodding did he sit down to program the "Cooley-Tukey" FFT. In short order, Cooley and Tukey prepared a paper which, for a mathematics/computer science paper, was published almost instantaneously (in six months!) [5]. This publication, as well as Garwin's fervent prosletizing, did a lot to help publicize the existence of this (apparently) new fast algorithm. (See also [4] and the introductory papers of [17] for more historical details.)

The timing of the announcement was such that now usage spread quickly. The roughly simultaneous development of analog to digital converters capable of producing digitized samples of a time-varying voltage at rates of $300,000$ samples/second had already initiated something of a digital revolution, and was also providing scientists with heretofore unimagined quantities of digital data to analyze and manipulate (just as is the case today!). Even the "standard" applications of Fourier analysis as an analysis tool for waveforms or solving PDEs, meant that a priori there would be a tremendous interest in the algorithm. But even more, the ability to do this analysis quickly allowed scientists from new areas to try the DFT without having to invest too much time and energy in the exercise. I can do no better than to quote the introduction to the FFT from Numerical Recipes, "If you speed up any nontrivial algorithm by a factor of a million or so the world will beat a path towards finding useful applications for it [3]."

3 Its effect

It truly does seem difficult to overstate the importance of the FFT. Much of its central place in digital signal and image processing is due to the fact that it made working in the frequency domain equally computationally feasible as working in the temporal or spatial domain. By providing a fast algorithm for convolution, the FFT enables fast large integer and polynomial multiplication, as well as efficient matrix-vector multiplication for Toeplitz, circulant and other kinds of structured matrices, and more generally, plays a key role in most efficient sorts of filtering algorithms. Modifications of the FFT are one approach to fast algorithms for discrete cosine or sine transforms, and also chebyshev transforms. In particular, the discrete cosine transform is at the heart of mp3 encoding which gives life to real time audio streaming. thereby realizing Cooley's dream that "someday radio tuners will operate with digital processing units. I have heard this suggested with tongue in cheek, but one can speculate." ([17], (1969), p. 66.) Last but not least, it also one of the few algorithms to make it in the movies – I can still recall the scene in "No Way Out" where the image processing guru declares that he will need to "Fourier transform the image" to help Kevin Costner see the detail in a photograph!

Even beyond these direct technological applications, the FFT influenced the direction of academic research too. The FFT was one of the first instances of a less than straightforward algorithm with a high payoff in efficiency to compute something important. Furthermore, it raised the natural question of "Could an even faster algorithm be found for the DFT?", (the answer is no, see [19]) thereby raising awareness of and heightening interest in the subject of lower bounds and the analysis and development of efficient algorithms in general. With respect to Winograd's lower bound analysis, Cooley writes in the discussion of the 1968 Arden House Workshop on FFT [17] "These are the beginnings, I believe, of a branch of computer science which will probably uncover and evaluate other algorithms for high speed computers."

Ironically, the prominence of the FFT may have also contributed to slow other areas of research. The FFT provided scientists with a big analytic hammer, and for many, the world suddenly looked as though it was full of nails – even if this wasn't always so. Problems which may have benefitted from other more appropriate techniques were sometimes massaged into a DFT framework, simply because the FFT was so efficient. One example which comes to mind is some of the early spectral methods work used in solving PDEs in spherical geometry. In this case the spherical harmonics are a natural set of basis functions. Discretization for numerical solutions implies the computation of discrete Legendre transforms (as well as FFTs). Many of the early computational approaches tried instead to approximate these expansions completely in terms of Fourier series, rather than address the development of an efficient Legendre transform.

Even now there are still lessons to be learned from the FFT's development. In this day and age in which almost any new technological idea seems fodder for internet venture capitalists and patent lawyers, it is natural to ask, "Why was the FFT not patented by IBM?" As Cooley tells the story, on the one hand, Tukey was not an IBM employee, so IBM had some worry that they might not be able to gain the patent. Consequently, they had great interest in putting the algorithm in the public domain. The effect of this was that no one else would be able to patent the algorithm, and even more, like many computer manufacturers of that time, the thought was that the money was to be made in hardware, not software. In fact, the FFT was designed as a tool for the analysis of huge time series, in theory something only tackled by supercomputers. So by placing in the public domain an algorithm which would make feasible the analysis of large time series, more big companies might have an interest in buying supercomputers (like IBM mainframes) to do their work. Times certainly have changed.

Whether having the FFT in the public domain had the effect IBM hoped for is moot, but it is certain that it did provide many, many scientists with applications to work on and apply the algorithm. The breadth of scientific interests at the Arden workshop (held only two years after publication of the paper) is truly impressive. In fact, the rapid pace of today's technological developments is in many ways a testament to the advantage of this open development. This is a cautionary tale in today's arena of proprietary research, and we can only wonder which of the many recent private technological discoveries might have prospered from a similar announcement.

4 The future FFT - the FFFT

As the torrents of digital data continue to stream in to our computers, whether it be from our bodies, the web or the stars, it would seem that the FFT will continue to play a prominent in role in our analysis and understanding of this river of data. What follows is a brief discussion of some future FFT challenges, as well as a few new directions of related research.

4.1 Even bigger FFTs

Astronomy continues to be a chief consumer of large FFT technology. At the 1968 Arden Workshop ([17], 1969, p. 66) the organizers presented the 512k FFTs demanded by certain interferometry calculations as a new computational goal. Today the needs of projects like MAP (Microwave Anisotropy Project) or LIGO (which is looking to detect gravitational waves) will require FFTs of several (even tens of) gigapoints. FFTs of this size do not fit in the main memory of most machines, and these so-called *out-of-core* FFTs are an active area of research (see eg. [6]).

As computing technology evolves, undoubtedly, versions of the FFT will evolve to keep pace and take advantage of new technologies. Different kinds of memory hierarchies and architectures present new challenges and opportunities. The work of Auslander, Johnson and Johnson [1] shows interesting mathematical relationships between different FFTs and architectures.

4.2 Approximate FFTs and nonuniform FFTs

For a variety of applications (eg. fast MRI) it is necessary to compute DFTs for nonuniformly spaced grid points and/or frequencies. Multipole-based approaches (see the multipole article in this issue) efficiently compute these quantities approximately, in such a way that the running time increases by a factor of $\log(\frac{1}{\epsilon})$ where ϵ denotes the precision of the approximation [8]. Algebraic approaches based on efficient polynomial evaluation are also possible (see eg. [12]).

4.3 Group FFTs

The FFT may also be explained and interpreted using the language of group representation theory and working along these lines raises some interesting avenues for generalization. One approach is to view a 1-D DFT of length N as computing the expansion of a function defined on C_N, the cyclic group of length N (the group of integers mod N) in terms of the basis of **irreducible matrix elements** of C_N, which are precisely the familiar sampled exponentials: $e_k(m) = \exp(2\pi\sqrt{-1}km/N)$. The FFT is a highly efficient algorithm for computing the expansion in this basis. More generally, a function on any compact group (cyclic or not) has an expansion in terms of a basis of irreducible matrix elements (which generalize the exponentials from the point of view of group invariance) and it is natural to ask if there exist efficient algorithms for performing this change of basis. For example, the problem of an efficient spherical harmonic expansion falls into this framework.

The first FFT for a noncommutative finite group seems to have been developed by Willsky in the context of analyzing certain markov processes [18]. To date, fast algorithms exist for many classes of compact groups. Areas of applications of this work include signal processing, data analysis and robotics. See [12, 14] for pointers to the literature.

4.4 Quantum FFTs

One of the first great triumphs of the quantum computing model is Shor's fast algorithm for integer factorization on a quantum computer [15]. At the heart of Shor's algorithm is a subroutine which computes (on a quantum computer) the DFT of a binary vector representing an integer. The implementation of this transform as a sequence of one- and two-bit quantum gates, now called the quantum FFT, is effectively the Cooley-Tukey FFT realized as a particular factorization of the Fourier matrix into a product of matrices composed as certain tensor products of two by two unitary matrices, each of which is a so-called local unitary transform. Similarly, the quantum solution to the "Modified Deutsch-Josza problem" uses the matrix factorization arising from Yates's algorithm (see eg. [16]). Extensions of these ideas to the more general group transforms mentioned above are currently being explored [11].

Acknowledgement. Special thanks to J. Cooley, S. Winograd, and M. Taylor for helpful conversations. The Santa Fe Institute provided partial support and a very friendly and stimulating environment while much of this paper was written.

References

[1] L. Auslander, J.R. Johnson, and R.W. Johnson, *Multidimensional Cooley-Tukey algorithms revisited*, Adv. in Appl. Math. **17** (1996), no. 4, 477–519.

[2] E. O. Brigham, *The fast Fourier transform and its applications*, Prentice Hall Signal Processing Series, Englewood Cliffs, NJ 1988.

[3] W. H. Press, B. P. Falnnery, S. A. Teukolsky, and W. T. Vettering, *Numerical recipes in C: the art of scientific computing*, Cambridge University Press, New York, 1988.

[4] J. W. Cooley, *The re-discovery of the fast Fourier transform algorithm*, Mikrochimica Acta, **III** (1987), 33–45.

[5] J. W. Cooley and J. W. Tukey, *An algorithm for machine calculation of complex Fourier series*, Math. Comp., **19** (1965), 297–301.

[6] T. H. Cormen and D. M. Nicol. *Performing out-of-core FFTs on parallel disk systems.* Parallel Computing, **24** (1998), no. 1, 5–20

[7] G. C. Danielson and C. Lanczos, *Some improvements in practical Fourier analysis and their application to X-ray scattering from liquids*, J. Franklin Inst. **233** (1942), no. 4 and 5, 365–380 and 432–452.

[8] A. Dutt and V. Rokhlin, *Fast Fourier transforms for nonequispaced data*, SIAM J. Sci. Comput. **14** (1993), no. 6, 1368–1393 and (part II) Appl. Comput. Harmon. Anal. **2** (1995), no. 1, 85–100

[9] W. M. Gentleman and G. Sande, *Fast Fourier transforms – for fun and profit*, 1966 Fall Joint Computer Conf. AFIPS Proc., vol. 29, Washington D.C., Spartan, (1966), pp. 563–578.

[10] M. T. Heideman, D. H. Johnson and C. S. Burrus, *Gauss and the history of the fast Fourier transform*, Archive for History of Exact Sciences, **34** (1985), no. 3, 265–277.

[11] P. Høyer, *Efficient quantum transforms*, LANL preprint quant-ph/9702028, February 1997.

[12] D. K. Maslen and D. N. Rockmore, *Generalized FFTs—a survey of some recent results*, Groups and computation, II (New Brunswick, NJ, 1995), 183–237, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., 28, Amer. Math. Soc., Providence, RI, 1997

[13] C. M. Rader, *Discrete Fourier transforms when the number of data points is prime*, Proc. of IEEE, vol. 56 (1968), 1107–1108.

[14] D. N. Rockmore, *Some applications of generalized FFTs* (An appendix w/D. Healy), in Proceedings of the DIMACS Workshop on Groups and Computation, June 7-10, 1995, eds. L. Finkelstein and W. Kantor, (1997), pp. 329–369

[15] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Computing **26** (1997), 1484–1509.

[16] D. Simon, *On the power of quantum computation*, in Proc. 35th Annual ACM Symposium on Foundations of Computer Science, 1994, pp. 116–123.

[17] *Special issue on fast Fourier transform and its application to digital filtering and spectral analysis*, IEEE Trans. Audio Electronics (1967), AU-15 and *Special issue on fast Fourier transform*, IEEE Trans. Audio Electronics (1969), AU-17.

[18] A. S. Willsky, *On the algebraic structure of certain partially observable finite-state Markov processes*, Inform. Contr., **38** (1978), 179–212.

[19] S. Winograd, *Arithmetic complexity of computations*, CBMS-NSF Regional Conference Series in Applied Mathematics, Vol. 33. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, Pa., 1980.

[20] F. Yates, *The design and analysis of factorial experiments*, Imp. Bur. Soil Sci. Tech. Comm., **35** (1937).

[21] C. Van Loan, *Computational framework for the fast Fourier transform*, SIAM, Philadelphia, 1992.