# Rediscovering the FFT

Matteo Frigo*
athena@lcs.mit.edu

May 4, 1995

## 1 Introduction

Suppose you've never heard of the three-letter acronym FFT (usually attributed to [1], but the ideas can be traced back to Gauss) and still wanted to compute circular convolutions (since they are an useful thing in almost every branch of computer science and engineering). What could you do? I'll present in this paper a novel (as far as I know) algorithm to compute the circular convolution of two vectors of length $n$ (if $n$ is a power of two, but the idea is more general) in time $O(n \log n)$. The algorithm does not require knowledge of the FFT, nor of the convolution theorem. From this approach the Discrete Fourier Transform, the convolution theorem and the FFT algorithm emerge in a natural way. I propose the algorithm as an alternative and more natural introduction to the Fourier Transform.

## 2 Circular convolutions

Given two input vectors $x, y \in \mathbb{C}^n$, you want to compute the *circular convolution* of $x$ and $y$, which is a vector of $\mathbb{C}^n$ whose $j$-th element is defined to be

$$z_j = (x * y)_j \triangleq \sum_{i=0}^{n-1} x_i y_{j-i}$$

(the indices are understood to be modulo the length of the vector for the remainder of this paper).

I present here a recursive algorithm to compute the convolution; the algorithm splits a problem of size $n$ into two subproblems of size $n/2$, requiring $O(n)$ operations to do so. The total running time is therefore $T(n) = O(n \log n)$.

There are no alternatives for the base case of the recursion: when $n = 1$ the algorithm must return $x_0 y_0$, which is the circular convolution of $x$ and $y$. Suppose now a problem of size $2n$ is given: our goal is reduce the problem to two circular convolutions of size $n$, and combine the two results to get the desired convolution of the input vectors.

The first step is to form the *periodic* and *antiperiodic part* of $x$ and $y$, respectively $x^+$ and $x^-$, as follows:

$$\begin{aligned} x_i^+ &= x_i + x_{i+n} \\ x_i^- &= x_i - x_{i+n} \\ y_i^+ &= y_i + y_{i+n} \\ y_i^- &= y_i - y_{i+n} \end{aligned}$$

It's immediate to conclude that the periodic part satisfies the relation $x_i^+ = x_{i+n}^+$ (whence the terminology), and the antiperiodic part obeys $x_i^- = -x_{i+n}^-$. In the same way it's trivial to show that $x = (x^+ + x^-)/2$ and $y = (y^+ + y^-)/2$; therefore

$$\begin{aligned} z &= x * y \\ &= \frac{x^+ * y^+ + x^+ * y^- + x^- * y^+ + x^- * y^-}{4} \end{aligned}$$

Now the second and third terms are easily seen to be zero; for example the second term expands to $(x^+ * y^-)_j = \sum_{i=0}^{2n-1} (x_i + x_{i+n})(y_{j-i} - y_{j-i+n}) = \sum_{i=0}^{2n-1} (x_i y_{j-i} + x_{i+n} y_{j-i} - x_i y_{j-i+n} - x_{i+n} y_{j-i+n})$ and each term of the latter sum cancels out. We are therefore left to compute $z = (x^+ * y^+ + x^- * y^-)/4$.

*Author's address: MIT NE43-203. This research was supported in part by the Ministero per l'Università e la Ricerca Scientifica of Italy, and by ARPA contracts N00014-94-1-0985 and N00014-92-J-1310. The author was supervised by Prof. C. Leiserson of MIT and Prof. G. Bilardi of Università di Padova.

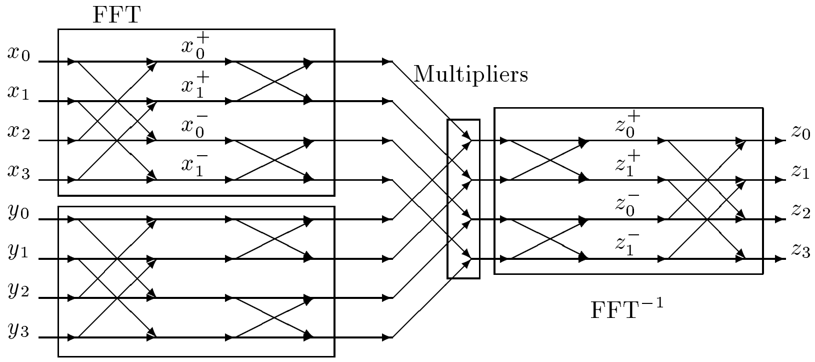


Figure 1: Computation DAG of the algorithm for a problem of size 4; as it can be noticed, the algorithm performs two FFTs and a reverse FFT. Multiplications by constant factors are not shown for lack of space.

The convolution of the periodic parts can be (recursively) computed with a single convolution of size $n$; in fact we can exploit the periodicity, and $\sum_{i=0}^{2n-1} x_i^+ y_{j-i}^+ = 2\sum_{i=0}^{n-1} x_i^+ y_{j-i}^+$. The idea for computing the convolution of the antiperiodic parts is to also make $x^-$ and $y^-$ periodic with period $n$. This is accomplished as follows: let $\omega$ be any complex number such that $\omega^n = -1$; now form two new vectors $\tilde{x}_i = \omega^i x_i^-$ and $\tilde{y}_i = \omega^i y_i^-$. $\tilde{x}$ and $\tilde{y}$ are easily shown to have period $n$; for example, $\tilde{x}_{i+n} = \omega^{i+n} x_{i+n}^- = -\omega^i(-x_i^-) = \tilde{x}_i$, and we can compute $\tilde{x} * \tilde{y}$ as we did before. Can we now reconstruct $x^- * y^-$ knowing $\tilde{x} * \tilde{y}$? Of course we can! And it's not hard to see how: in fact

$$\begin{aligned}(\tilde{x} * \tilde{y})_j &= \sum_{i=0}^{2n-1} \tilde{x}_i \tilde{y}_{j-i} = \sum_{i=0}^{2n-1} \omega^i x_i^- \omega^{j-i} y_{j-i}^- \\ &= \omega^j (x^- * y^-)_j\end{aligned}$$

Hence we succeeded in dividing a single convolution of size $2n$ into two subproblems of size $n$, performing the promised work $O(n)$.

# 3 Additional comments

Compared with the traditional approach (that is, introduce the DFT, prove the convolution theorem and "discover" the FFT), the algorithm I propose is much simpler to explain, to program and to understand, and shows that the convolution doesn't necessarily require the idea of DFT. Moreover the FFT algorithm has some oddities, like the bit-reversal and the twiddle factors, which in my experience always cause bugs in the programs, and require careful coding. In contrast, the flow of the operations and the coding are straightforward in my algorithm. It's also easy to do all computations in place (which is again a tricky part in any FFT code).

Given all these "didactic" advantages of the algorithm, it is natural to wonder whether it is any faster than the approach based on the convolution theorem, i.e. compute the FFT of the input vectors, multiply the transformed vectors and perform the inverse FFT. Not surprisingly, the answer is "no"; in fact the proposed algorithm does *exactly* the same computations as the traditional approach. This is apparent from the figure (which exemplifies the computation DAG for a problem of size 4): the input vectors don't interact until the middle stage, which multiplies the transformed vectors elementwise. The input vectors undergo the same transformations as the ordinary radix-2 FFT, and the operation performed by the recombination stage of the algorithm is nothing else than an inverse FFT. We have discovered a way to define the DFT in terms of the intermediate values computed by the algorithm!

# References

[1] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 19(90):297–301, April 1965.