The Tangent FFT[⋆]

Daniel J. Bernstein

Department of Mathematics, Statistics, and Computer Science (M/C 249)
University of Illinois at Chicago, Chicago, IL 60607–7045, USA
djb@cr.yp.to

Abstract. The split-radix FFT computes a size-n complex DFT, when n is a large power of 2, using just $4n \lg n - 6n + 8$ arithmetic operations on real numbers. This operation count was first announced in 1968, stood unchallenged for more than thirty years, and was widely believed to be best possible.

Recently James Van Buskirk posted software demonstrating that the split-radix FFT is *not* optimal. Van Buskirk's software computes a size-n complex DFT using only $(34/9 + o(1))n \lg n$ arithmetic operations on real numbers. There are now three papers attempting to explain the improvement from 4 to 34/9: Johnson and Frigo, *IEEE Transactions on Signal Processing*, 2007; Lundy and Van Buskirk, *Computing*, 2007; and this paper.

This paper presents the "tangent FFT," a straightforward in-place cache-friendly DFT algorithm having exactly the same operation counts as Van Buskirk's algorithm. This paper expresses the tangent FFT as a sequence of standard polynomial operations, and pinpoints how the tangent FFT saves time compared to the split-radix FFT. This description is helpful not only for understanding and analyzing Van Buskirk's improvement but also for minimizing the memory-access costs of the FFT.

Keywords: Tangent FFT, split-radix FFT, modified split-radix FFT, scaled odd tail, DFT; convolution,polynomial multiplication, algebraic complexity, communication complexity.

1 Introduction

Consider the problem of computing the size-n complex **DFT** ("discrete Fourier transform"), where n is a power of 2; i.e., evaluating an n-coefficient univariate complex polynomial f at all of the nth roots of 1. The input is a sequence of n complex numbers $f_0, f_1, \ldots, f_{n-1}$ representing the polynomial $f = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1}$. The output is the sequence $f(1), f(\zeta_n), f(\zeta_n^2), \ldots, f(\zeta_n^{n-1})$ where $\zeta_n = \exp(2\pi i/n)$.

The size-n **FFT** ("fast Fourier transform") is a well-known algorithm to compute the size-n DFT using $(5+o(1))n \lg n$ arithmetic operations on real numbers. One can remember the coefficient 5 as half the total cost of a complex addition

[⋆] Permanent ID of this document: a9a77cef9a7b77f9b8b305e276d5fe25. Date of this document: 2007.09.19.

(2 real operations), a complex subtraction (2 real operations), and a complex multiplication (6 real operations).

The FFT was used for astronomical calculations by Gauss in 1805; see, e.g., [6, pages 308–310], published in 1866. It was reinvented and republished on several subsequent occasions and was finally popularized in 1965 by Cooley and Tukey in [2]. The advent of high-speed computers meant that users in the 1960s were trying to handle large values of n in a wide variety of applications and could see large benefits from the FFT.

The Cooley-Tukey paper spawned a torrent of FFT papers—showing, among other things, that Gauss had missed a trick. The original FFT is not the optimal way to compute the DFT. In 1968, Yavne stated that one could compute the DFT using only $(4+o(1))n \lg n$ arithmetic operations, specifically $4n \lg n - 6n + 8$ arithmetic operations (if $n \geq 2$), specifically $n \lg n - 3n + 4$ multiplications and $3n \lg n - 3n + 4$ additions; see [13, page 117]. Nobody, to my knowledge, has ever deciphered Yavne's description of his algorithm, but a comprehensible algorithm achieving exactly the same operation counts was introduced by Duhamel and Hollmann in [3], by Martens in [9], by Vetterli and Nussbaumer in [12], and by Stasinski (according to [4, page 263]). This algorithm is now called the **split-radix FFT**.

The operation count $4n \lg n - 6n + 8$ stood unchallenged for more than thirty years[1] and was frequently conjectured to be optimal. For example, [11, page 152] said that split-radix FFT algorithms did not have minimal multiplication counts but "have what seem to be the best compromise operation count." Here "compromise" refers to counting both additions and multiplications rather than merely counting multiplications.

In 2004, James Van Buskirk posted software that computed a size-64 DFT using fewer operations than the size-64 split-radix FFT. Van Buskirk then posted similar software handling arbitrary power-of-2 sizes using only $(34/9+o(1))n \lg n$ arithmetic operations. Of course, 34/9 is still in the same ballpark as 4 (and 5), but it is astonishing to see *any* improvement in such a widely studied, widely used algorithm, especially after 36 years of no improvements at all!

Contents of this paper. This paper gives a concise presentation of the **tangent FFT**, a straightforward in-place cache-friendly DFT algorithm having exactly the same operation counts as Van Buskirk's algorithm. This paper expresses the tangent FFT as a sequence of standard polynomial operations, and pinpoints how the tangent FFT saves time compared to the split-radix FFT. This description is helpful not only for understanding and analyzing Van Buskirk's improvement but also for minimizing the memory-access costs of the FFT.

[1] The 1998 paper [14] claimed that its "new fast Discrete Fourier Transform" was much faster than the split-radix FFT. For example, the paper claimed that its algorithm computed a size-16 real DFT with 22 additions and 10 multiplications by various sines and cosines. I spent half an hour with the paper, finding several blatant errors and no new ideas; in particular, Figure 1 of the paper had many more additions than the paper claimed. I pointed out the errors to the authors and have not received a satisfactory response.

There have been two journal papers this year—[8] by Lundy and Van Buskirk, and [7] by Johnson and Frigo—presenting more complicated algorithms with the same operation counts. Both algorithms can be transformed into in-place algorithms but incur heavier memory-access costs than the algorithm presented in this paper.

I chose the name "tangent FFT" in light of the essential role played by tangents as constants in the algorithm. The same name could be applied to all of the algorithms in this class. Lundy and Van Buskirk in [8] use the name "scaled odd tail," which I find less descriptive. Johnson and Frigo in [7] use the name "our new FFT ... our new algorithm ... our algorithm ... our modified algorithm" etc., which strikes me as suboptimal terminology; I have already seen three reports miscrediting Van Buskirk's 34/9 to Johnson and Frigo. All of the credit for these algorithms should be assigned to Van Buskirk, except in contexts where extra features such as simplicity and cache-friendliness play a role.

2 Review of the Original FFT

The remainder $f \bmod x^8 - 1$, where f is a univariate polynomial, determines the remainders $f \bmod x^4 - 1$ and $f \bmod x^4 + 1$. Specifically, if

$$f \bmod x^8 - 1 = f_0 + f_1x + f_2x^2 + f_3x^3 + f_4x^4 + f_5x^5 + f_6x^6 + f_7x^7,$$

then $f \bmod x^4 - 1 = (f_0 + f_4) + (f_1 + f_5)x + (f_2 + f_6)x^2 + (f_3 + f_7)x^3$ and $f \bmod x^4 + 1 = (f_0 - f_4) + (f_1 - f_5)x + (f_2 - f_6)x^2 + (f_3 - f_7)x^3$. Computing the coefficients $f_0+f_4, f_1+f_5, f_2+f_6, f_3+f_7, f_0-f_4, f_1-f_5, f_2-f_6, f_3-f_7$, given the coefficients $f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7$, involves 4 complex additions and 4 complex subtractions. Note that this computation is naturally carried out in place with one sequential sweep through the input. Note also that this computation is easy to invert: for example, the sum of $f_0 + f_4$ and $f_0 - f_4$ is $2f_0$, and the difference is $2f_4$.

More generally, let r be a nonzero complex number, and let n be a power of 2. The remainder $f \bmod x^{2n} - r^2$ determines the remainders $f \bmod x^n - r$ and $f \bmod x^n + r$, since $x^n - r$ and $x^n + r$ divide $x^{2n} - r^2$. Specifically, if

$$f \bmod x^{2n} - r^2 = f_0 + f_1x + \cdots + f_{2n-1}x^{2n-1},$$

then $f \bmod x^n - r = (f_0 + rf_n) + (f_1 + rf_{n+1})x + \cdots + (f_{n-1} + rf_{2n-1})x^{n-1}$ and $f \bmod x^n + r = (f_0 - rf_n) + (f_1 - rf_{n+1})x + \cdots + (f_{n-1} - rf_{2n-1})x^{n-1}$. This computation involves n complex multiplications by r; n complex additions; and n complex subtractions; totalling $10n$ real operations. The following diagram summarizes the structure and cost of the computation:

$$\begin{tikzcd}
 & x^{2n} - r^2 \arrow[dl] \arrow[dr] \arrow[d, phantom, "\boxed{10n}"] & \\
x^n - r & & x^n + r
\end{tikzcd}$$

Note that some operations disappear when multiplications by r are easy: this computation involves only $8n$ real operations if $r \in \left\{\sqrt{i}, -\sqrt{i}, \sqrt{-i}, -\sqrt{-i}\right\}$, and only $4n$ real operations if $r \in \{1, -1, i, -i\}$.

The same idea can be applied recursively:

$$\begin{tikzcd}
& & & & & & & x^8-1 \arrow[ddllll] \arrow[ddrrrr] \\
& & & & & & & \boxed{16} \\
& & & x^4-1 \arrow[ddll] \arrow[ddrr] & & & & & & & & x^4+1 \arrow[ddll] \arrow[ddrr] \\
& & & \boxed{8} & & & & & & & & \boxed{8} \\
& x^2-1 \arrow[ddl] \arrow[ddr] & & & x^2+1 \arrow[ddl] \arrow[ddr] & & & & x^2-i \arrow[ddl] \arrow[ddr] & & & & x^2+i \arrow[ddl] \arrow[ddr] \\
& \boxed{4} & & & \boxed{4} & & & & \boxed{8} & & & & \boxed{8} \\
x-1 & & x+1 & & x-i & & x+i & & x-\sqrt{i} & & x+\sqrt{i} & & x-\sqrt{-i} & & x+\sqrt{-i}
\end{tikzcd}$$

The final outputs $f \bmod x-1, f \bmod x+1, f \bmod x-i, \ldots$ are exactly the (permuted) DFT outputs $f(1), f(-1), f(i), \ldots$, and this computation is exactly Gauss's original FFT. Note that the entire computation is naturally carried out in place, with contiguous inputs to each recursive step. One can further reduce the number of cache misses by merging (e.g.) the top two levels of recursion.

This view of the FFT, identifying each FFT step as a simple polynomial operation, was introduced by Fiduccia in [5]. Most papers (and books) suppress the polynomial structure, viewing each intermediate FFT result as merely a linear function of the input; but "$f \bmod x^n - r$" is much more concise than a matrix expressing the same function!

One might object that the concisely expressed polynomial operations in this section and in subsequent sections are less general than arbitrary linear functions. Is this restriction compatible with the best FFT algorithms? For example, does it allow Van Buskirk's improved operation count? This paper shows that the answer is yes. Perhaps some future variant of the FFT will force Fiduccia's philosophy to be reconsidered, but for the moment one can safely recommend that FFT algorithms be expressed in polynomial form.

3 Review of the Twisted FFT

The remainder $f \bmod x^n + 1$ determines the remainder $f(\zeta_{2n}x) \bmod x^n - 1$. Specifically, if $f \bmod x^n + 1 = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1}$, then

$$f(\zeta_{2n}x) \bmod x^n - 1 = f_0 + \zeta_{2n} f_1 x + \cdots + \zeta_{2n}^{n-1} f_{n-1} x^{n-1}.$$

Computing the **twisted** coefficients $f_0, \zeta_{2n}f_1, \ldots, \zeta_{2n}^{n-1}f_{n-1}$ from the coefficients $f_0, f_1, \ldots, f_{n-1}$ involves one multiplication by ζ_{2n}, one multiplication by ζ_{2n}^2, and so on through ζ_{2n}^{n-1}. These $n-1$ multiplications cost $6(n-1)$ real operations, except that a few multiplications are easier: 6 operations are saved for $\zeta_{2n}^{n/2}$ when $n \geq 2$, and another 4 operations are saved for $\zeta_{2n}^{n/4}, \zeta_{2n}^{3n/4}$ when $n \geq 4$.

The remainder $f \bmod x^{2n}-1$ determines the remainders $f \bmod x^n-1$ and $f \bmod x^n+1$, as discussed in the previous section. It therefore determines the remainders $f \bmod x^n-1$ and $f(\zeta_{2n}x) \bmod x^n-1$, as summarized in the following diagram:

$$\begin{tikzcd}
 & x^{2n}-1 \arrow[dl] \arrow[dr] & \\
x^n-1 & \boxed{4n} & x^n+1 \arrow[d, "\zeta_{2n}"] \\
 & \boxed{\max\{6n-16,0\}} & x^n-1
\end{tikzcd}$$

The **twisted FFT** performs this computation and then recursively evaluates both $f \bmod x^n-1$ and $f(\zeta_{2n}x) \bmod x^n-1$ at the nth roots of 1, obtaining the same results as the original FFT. Example, for $n=8$:

$$\begin{tikzcd}
 & & & x^8-1 \arrow[ddll] \arrow[ddrr] & & & & \\
 & & & \boxed{16} & & & & \\
 & x^4-1 \arrow[ddddl] \arrow[ddddr] & & & & x^4+1 \arrow[dd, "\sqrt{i}"] & & \\
 & & & & \boxed{8} & & & \\
 & & & & & x^4-1 \arrow[ddl] \arrow[ddr] & & \\
 & \boxed{8} & & & & \boxed{8} & & \\
x^2-1 \arrow[dddd] \arrow[ddddr] & & x^2+1 \arrow[dd, "i"] & & x^2-1 \arrow[dddd] \arrow[ddddr] & & x^2+1 \arrow[dd, "i"] & \\
 & \boxed{0} & & & & \boxed{0} & & \\
 & & x^2-1 \arrow[dd] \arrow[ddr] & & & & x^2-1 \arrow[dd] \arrow[ddr] & \\
\boxed{4} & & \boxed{4} & & \boxed{4} & & \boxed{4} & \\
x-1 & x+1 \arrow[dd, "-1"] & x-1 & x+1 \arrow[dd, "-1"] & x-1 & x+1 \arrow[dd, "-1"] & x-1 & x+1 \arrow[dd, "-1"] \\
\boxed{0} & & \boxed{0} & & \boxed{0} & & \boxed{0} & \\
 & x-1 & & x-1 & & x-1 & & x-1
\end{tikzcd}$$

Note that the twisted FFT never has to consider moduli other than $x^n \pm 1$. The twisted FFT thus has a simpler recursive structure than the original FFT. The recursive step does not need to distinguish f from $f(\zeta_{2n}x)$: its job is simply to evaluate an input modulo $x^n - 1$ at the nth roots of 1.

One can easily prove that the twisted FFT uses the same number of real operations as the original FFT: the cost of twisting $x^n + 1$ into $x^n - 1$ is exactly balanced by the savings from avoiding $x^{n/4} - \sqrt{i}$ etc. In fact, the algorithms have the same number of multiplications by each root of 1. (One way to explain this coincidence is to observe that the algorithms are "transposes" of each other.) One might speculate at this point that *all* FFT algorithms have the same number of real operations; but this speculation is solidly disproven by the split-radix FFT, as discussed in Section 4.

4 Review of the Split-Radix FFT

The **split-radix FFT** applies the following diagram recursively:

$$\begin{tikzcd}
 & & x^{4n}-1 \arrow[dll] \arrow[drr] & & & & \\
x^{2n}-1 & & \boxed{8n} & & x^{2n}+1 \arrow[dll] \arrow[drr] & & \\
 & & x^n - i \arrow[d, "\zeta_{4n}", "\boxed{\max\{6n-8,0\}}"'] & & \boxed{4n} & & x^n + i \arrow[d, "\zeta_{4n}^{-1}", "\boxed{\max\{6n-8,0\}}"'] \\
 & & x^n - 1 & & & & x^n - 1
\end{tikzcd}$$

The notation here is the same as in previous sections:

- from $f \bmod x^{4n} - 1$ compute $f \bmod x^{2n} - 1$ and $f \bmod x^{2n} + 1$;
- from $f \bmod x^{2n} + 1$ compute $f \bmod x^n - i$ and $f \bmod x^n + i$;
- from $f \bmod x^n - i$ compute $f(\zeta_{4n}x) \bmod x^n - 1$;
- from $f \bmod x^n + i$ compute $f(\zeta_{4n}^{-1}x) \bmod x^n - 1$;
- recursively evaluate $f \bmod x^{2n} - 1$ at the 2nth roots of 1;
- recursively evaluate $f(\zeta_{4n}x) \bmod x^n - 1$ at the nth roots of 1; and
- recursively evaluate $f(\zeta_{4n}^{-1}x) \bmod x^n - 1$ at the nth roots of 1.

If $f \bmod x^n - i = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1}$ then $f(\zeta_{4n}x) \bmod x^n - 1 = f_0 + \zeta_{4n} f_1 x + \cdots + \zeta_{4n}^{n-1} f_{n-1} x^{n-1}$. The $n - 1$ multiplications here cost $6(n-1)$ real operations, except that 2 operations are saved for $\zeta_{4n}^{n/2}$ when $n \geq 2$. Similar comments apply to $x^n + i$.

The split-radix FFT uses only about $8n+4n+6n+6n = 24n$ operations to divide $x^{4n}-1$ into $x^{2n}-1, x^n-1, x^n-1$, and therefore only about $(24/1.5)n \lg n = 16n \lg n$

operations to handle $x^{4n}-1$ recursively. Here $1.5 = (2/4)\lg(4/2)+(1/4)\lg(4/1)+(1/4)\lg(4/1)$ arises as the entropy of $2n/4n, n/4n, n/4n$. An easy induction produces a precise operation count: the split-radix FFT handles $x^n - 1$ using 0 operations for $n = 1$ and $4n \lg n - 6n + 8$ operations for $n \geq 2$.

For the same split of $x^{4n} - 1$ into $x^{2n} - 1, x^n - 1, x^n - 1$, the twisted FFT would use about $30n$ operations: specifically, $20n$ operations to split $x^{4n} - 1$ into $x^{2n} - 1, x^{2n} - 1$, and then $10n$ operations to split $x^{2n} - 1$ into $x^n - 1, x^n - 1$, as discussed in Section 3. The split-radix FFT does better by delaying the expensive twists, carrying out only two size-n twists rather than one size-$2n$ twist and one size-n twist.

Most descriptions of the split-radix FFT replace $\zeta_{4n}, \zeta_{4n}^{-1}$ with $\zeta_{4n}, \zeta_{4n}^{3}$. Both ζ_{4n}^{-1} and ζ_{4n}^{3} are nth roots of $-i$; both variants compute (in different orders) the same DFT outputs. There is, however, an advantage of ζ_{4n}^{-1} over ζ_{4n}^{3} in reducing memory-access costs. The split-radix FFT naturally uses ζ_{4n}^{k} and ζ_{4n}^{-k} as multipliers at the same moment; loading precomputed real numbers $\cos(2\pi k/4n)$ and $\sin(2\pi k/4n)$ produces not only $\zeta_{4n}^{k} = \cos(2\pi k/4n) + i\sin(2\pi k/4n)$ but also $\zeta_{4n}^{-k} = \cos(2\pi k/4n) - i\sin(2\pi k/4n)$. Reciprocal roots also play a critical role in the tangent FFT; see Section 5.

5 The Tangent FFT

The obvious way to multiply $a + bi$ by a constant $\cos\theta + i\sin\theta$ is to compute $a\cos\theta - b\sin\theta$ and $a\sin\theta + b\cos\theta$. A different approach is to factor $\cos\theta + i\sin\theta$ as $(1 + i\tan\theta)\cos\theta$, or as $(\cot\theta + i)\sin\theta$. Multiplying by a real number $\cos\theta$ is relatively easy, taking only 2 real operations. Multiplying by $1 + i\tan\theta$ is also relatively easy, taking only 4 real operations.

This change does not make any immediate difference in operation count: either strategy takes 6 real operations, when appropriate constants such as $\tan\theta$ have been precomputed. But the change allows some extra flexibility: the real multiplication can be moved elsewhere in the computation. Van Buskirk's clever observation is that these real multiplications can sometimes be combined!

Specifically, let's change the basis $1, x, x^2, \ldots, x^{n-1}$ that we've been using to represent polynomials modulo $x^n - 1$. Let's instead use a vector $(f_0, f_1, \ldots, f_{n-1})$ to represent the polynomial $f_0/s_{n,0} + f_1 x/s_{n,1} + \cdots + f_{n-1}x^{n-1}/s_{n,n-1}$ where

$$s_{n,k} = \prod_{\ell \geq 0} \max\left\{ \left|\cos\frac{4^\ell 2\pi k}{n}\right|, \left|\sin\frac{4^\ell 2\pi k}{n}\right| \right\}.$$

This might appear at first glance to be an infinite product, but $4^\ell 2\pi k/n$ is a multiple of 2π once ℓ is large enough, so almost all of the terms in the product are 1.

This wavelet $s_{n,k}$ is designed to have two important features. The first is **periodicity**: $s_{4n,k} = s_{4n,k+n}$. The second is **cost-4 twisting**: $\zeta_{4n}^{k}(s_{n,k}/s_{4n,k})$ is $\pm(1 + i\tan\cdots)$ or $\pm(\cot\cdots + i)$.

The **tangent FFT** applies the following diagram recursively:

$$\begin{tikzcd}
 & & & \begin{array}{c}x^{8n}-1\\ x^k/s_{8n,k}\end{array} \arrow[ddll] \arrow[ddrr] & & & \\
 & & & \boxed{16n} & & & \\
 & \begin{array}{c}x^{4n}-1\\ x^k/s_{8n,k}\end{array} \arrow[ddl] \arrow[ddr] & & & & \begin{array}{c}x^{4n}+1\\ x^k/s_{8n,k}\end{array} \arrow[ddl] \arrow[ddr] & \\
 & \boxed{8n} & & & & \boxed{8n} & \\
\begin{array}{c}x^{2n}-1\\ x^k/s_{8n,k}\end{array} \arrow[d, "\boxed{4n-2}"] & & \begin{array}{c}x^{2n}+1\\ x^k/s_{8n,k}\end{array} \arrow[d, "\boxed{4n-2}"] & & \begin{array}{c}x^{2n}-i\\ x^k/s_{8n,k}\end{array} \arrow[dddd, "\boxed{8n-6}"', "\zeta_{8n}"] & & \begin{array}{c}x^{2n}+i\\ x^k/s_{8n,k}\end{array} \arrow[dddd, "\boxed{8n-6}"', "\zeta_{8n}^{-1}"] \\
\begin{array}{c}x^{2n}-1\\ x^k/s_{2n,k}\end{array} & & \begin{array}{c}x^{2n}+1\\ x^k/s_{4n,k}\end{array} \arrow[ddl] \arrow[ddr] & & & & \\
 & & \boxed{4n} & & & & \\
\boxed{\begin{array}{c}\max\{\\ 4n-6\\ ,0\}\end{array}} & \begin{array}{c}x^{n}-i\\ x^k/s_{4n,k}\end{array} \arrow[dl, "\zeta_{4n}"'] & \boxed{\begin{array}{c}\max\{\\ 4n-6\\ ,0\}\end{array}} & \begin{array}{c}x^{n}+i\\ x^k/s_{4n,k}\end{array} \arrow[dl, "\zeta_{4n}^{-1}"] & & & \\
\begin{array}{c}x^{n}-1\\ x^k/s_{n,k}\end{array} & & \begin{array}{c}x^{n}-1\\ x^k/s_{n,k}\end{array} & & \begin{array}{c}x^{2n}-1\\ x^k/s_{2n,k}\end{array} & & \begin{array}{c}x^{2n}-1\\ x^k/s_{2n,k}\end{array}
\end{tikzcd}$$

This diagram explicitly shows the basis used for each remainder $f \bmod x^{\cdots} - \cdots$. The top node, $x^{8n}-1$ with basis $x^k/s_{8n,k}$, reads an input vector $(f_0, f_1, \ldots, f_{8n-1})$ representing $f \bmod x^{8n}-1 = \sum_{0\le k<8n} f_k x^k/s_{8n,k}$. The next node to the left, $x^{4n}-1$ with basis $x^k/s_{8n,k}$, computes a vector $(g_0, g_1, \ldots, g_{4n-1})$ representing $f \bmod x^{4n}-1 = \sum_{0\le k<4n} g_k x^k/s_{8n,k}$; the equation $s_{8n,k+4n} = s_{8n,k}$ immediately implies that

$$(g_0, g_1, \ldots, g_{4n-1}) = (f_0 + f_{4n}, f_1 + f_{4n+1}, \ldots, f_{4n-1} + f_{8n-1}).$$

The next node to the left, $x^{2n}-1$ with basis $x^k/s_{8n,k}$, similarly computes a vector $(h_0, h_1, \ldots, h_{2n-1})$ representing $f \bmod x^{2n}-1 = \sum_{0\le k<2n} h_k x^k/s_{8n,k}$. The next node after that, $x^{2n}-1$ with basis $x^k/s_{2n,k}$ (suitable for recursion), computes a vector $(h'_0, h'_1, \ldots, h'_{2n-1})$ representing $f \bmod x^{2n}-1 = \sum_{0\le k<2n} h'_k x^k/s_{2n,k}$; evidently $h'_k = h_k(s_{2n,k}/s_{8n,k})$, requiring a total of $2n$ real multiplications by the precomputed real constants $s_{2n,k}/s_{8n,k}$, minus 1 skippable multiplication by $s_{2n,0}/s_{8n,0} = 1$. Similar comments apply throughout the diagram: for example, moving from $x^{2n}-i$ with basis $x^k/s_{8n,k}$ to $x^{2n}-1$ with basis $x^k/s_{2n,k}$ involves cost-4 twisting by $\zeta_{8n}^k s_{2n,k}/s_{8n,k}$.

The total cost of the tangent FFT is about $68n$ real operations to divide $x^{8n}-1$ into $x^{2n}-1, x^{2n}-1, x^{2n}-1, x^n-1, x^n-1$, and therefore about $(68/2.25)n\lg n = (34/9)8n\lg n$ to handle $x^{8n}-1$ recursively. Here 2.25 is the entropy of $2n/8n, 2n/8n, 2n/8n, n/8n, n/8n$. More precisely, the cost $S(n)$ of handling x^n-1 with basis $x^k/s_{n,k}$ satisfies $S(1)=0$, $S(2)=4$, $S(4)=16$, and $S(8n) = 60n-16+\max\{8n-12,0\}+3S(2n)+2S(n)$. The $S(n)$ sequence begins $0, 4, 16, 56, 164, 444, 1120, 2720, 6396, 14724, 33304, \ldots$; an easy induction shows that $S(n) = (34/9)n\lg n - (142/27)n - (2/9)(-1)^{\lg n}\lg n + (7/27)(-1)^{\lg n} + 7$ for $n \ge 2$.

For comparison, the split-radix FFT uses about $72n$ real operations for the same division. The split-radix FFT uses the same $16n$ to divide $x^{8n}-1$ into $x^{4n}-1, x^{4n}+1$, the same $8n$ to divide $x^{4n}-1$ into $x^{2n}-1, x^{2n}+1$, the same $8n$ to divide $x^{4n}+1$ into $x^{2n}-i, x^{2n}+i$, and the same $4n$ to divide $x^{2n}+1$ into x^n-i, x^n+i. It also saves $4n$ changing basis for $x^{2n}-1$ and $4n$ changing basis for $x^{2n}+1$. But the tangent FFT saves $4n$ twisting $x^{2n}-i$, another $4n$ twisting $x^{2n}+i$, another $2n$ twisting x^n-i, and another $2n$ twisting x^n+i. The $12n$ operations saved in twists outweigh the $8n$ operations lost in changing basis.

What if the input is in the traditional basis $1, x, x^2, \ldots, x^{n-1}$? One could scale the input immediately to the new basis, but it is faster to wait until the first twist:

$$\begin{tikzcd}
 & & \substack{x^{4n}-1 \\ x^k} \arrow[dll] \arrow[drr] & & & \\
\substack{x^{2n}-1 \\ x^k} & & \boxed{8n} & & \substack{x^{2n}+1 \\ x^k} \arrow[dl] \arrow[dr] & \\
 & & & \substack{x^n-i \\ x^k} \arrow[d, "\zeta_{4n}"] & \boxed{4n} & \substack{x^n+i \\ x^k} \arrow[d, "\zeta_{4n}^{-1}"] \\
 & & \boxed{\max\{6n-8,0\}} & \substack{x^n-1 \\ x^k/s_{n,k}} & \boxed{\max\{6n-8,0\}} & \substack{x^n-1 \\ x^k/s_{n,k}}
\end{tikzcd}$$

The coefficient of x^k in $f \bmod x^n - i$ is now twisted by $\zeta_{4n}^k s_{n,k}$, costing 6 real operations except for the easy cases $\zeta_{4n}^0 s_{n,0} = 1$ and $\zeta_{4n}^{n/2} s_{n,n/2} = \sqrt{i}$.

The cost $T(n)$ of handling x^n-1 with basis x^k satisfies $T(1)=0$, $T(2)=4$, and $T(4n) = 12n + \max\{12n-16,0\} + T(2n) + 2S(n)$. The $T(n)$ sequence begins $0, 4, 16, 56, 168, 456, 1152, 2792, 6552, 15048, 33968, \ldots$; an easy induction shows that

$$T(n) = \frac{34}{9}n\lg n - \frac{124}{27}n - 2\lg n - \frac{2}{9}(-1)^{\lg n}\lg n + \frac{16}{27}(-1)^{\lg n} + 8$$

for $n \ge 2$, exactly matching the operation count in [7, Equation (1)].

References

1. 1968 Fall Joint Computer Conference. In: AFIPS conference proceedings, vol. 33, part one. See [13] (1968)
2. Cooley, J.W., Tukey, J.W.: An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics of Computation 19, 297–301 (1965)
3. Duhamel, P., Hollmann, H.: Split-Radix FFT algorithm. Electronics Letters 20, 14–16 (1984)
4. Duhamel, P., Vetterli, M.: Fast Fourier Transforms: a Tutorial Review and a State of the Art. Signal Processing 19, 259–299 (1990)
5. Fiduccia, C.M.: Polynomial Evaluation Via the Division Algorithm: the Fast Fourier Transform Revisited. In: [10], pp. 88–93 (1972)
6. Gauss, C.F.: Werke, Band 3 Königlichen Gesellschaft der Wissenschaften. Göttingen (1866)
7. Johnson, S.G., Frigo, M.: A Modified Split-Radix FFT with Fewer Arithmetic Operations. IEEE Trans. on Signal Processing 55, 111–119 (2007)
8. Lundy, T.J., Van Buskirk, J.: A New Matrix Approach to Real FFTs and Convolutions of Length 2^k. Computing 80, 23–45 (2007)
9. Martens, J.B.: Recursive Cyclotomic Factorization—A New Algorithm for Calculating the Discrete Fourier Transform. IEEE Trans. Acoustics, Speech, and Signal Processing 32, 750–761 (1984)
10. Rosenberg, A.L.: Fourth Annual ACM Symposium on Theory Of Computing. Association for Computing Machinery, New York (1972)
11. Sorensen, H.V., Heideman, M.T., Burrus, C.S.: On Computing the Split-Radix FFT. IEEE Trans. Acoustics, Speech, and Signal Processing 34, 152–156 (1986)
12. Vetterli, M., Nussbaumer, H.J.: Simple FFT and DCT Algorithms with Reduced Number of Operations. Signal Processing 6, 262–278 (1984)
13. Yavne, R.: An Economical Method for Calculating the Discrete Fourier Transform. In: [1], pp. 115–125 (1968)
14. Zhou, F., Kornerup, P.: A New Fast Discrete Fourier Transform. J. VLSI Signal Processing 20, 219–232 (1998)