Additive Fast Fourier Transforms over Finite Fields

Shuhong Gao and Todd Mateer

***Abstract*—We present new additive Fast Fourier Transform (FFT) algorithms based on Taylor expansion over finite fields of characteristic two. Our algorithms improve previous approaches by Wang and Zhu (1988), Cantor (1989), and von zur Gathen and Gerhard (1996).**

***Index Terms*—Fast Fourier Transform, FFT, Taylor expansion, multiplication, convolution, Reed-Solomon Codes**

I. Introduction

DISCRETE Fourier transforms of length n correspond to evaluation of polynomials at n distinct points. Fast Fourier transforms (FFT) were discovered by Cooley and Tukey in 1965 [1] and they have since become an important tool in science and engineering. These traditional FFTs are possible when the n points are n-th roots of unity (and so form a cyclic multiplicative group of order n) where n is a power of 2 or a product of small primes (see [2]). They are based on the factorization of the polynomial $x^n - 1$, corresponding to the subgroup structure of the multiplicative group of order n. We will refer to these transforms as multiplicative FFTs throughout this paper.

Unfortunately, multiplicative FFTs do not usually support problems of size n when n is not a product of small primes, or when the underlying fields do not have the desired n-th roots of unity. This is the case for Fourier transforms over finite fields, related to encoding and decoding of Reed-Solomon codes [3], [4], [5]. When n is a power of two, a finite field $\mathbb{F}_{2^k}$ does not support multiplicative FFTs of length n since there are no primitive n-th roots of unity in any field of characteristic two. When n is a power of 3, $\mathbb{F}_{2^k}$ must have a value for k as big as $2n/3$ to contain an nth root of unity, so the field size has to be exponential in n.

Additive FFTs over finite fields were invented in the late 1980s. These algorithms are based on the factorization of $x^n - x$ and can be used to evaluate a polynomial at each of the powers of a primitive $(n-1)$-st root of unity as well as the zero element. Wang and Zhu [6] first showed in 1988 how to compute the FFT based on this factorization for the case where $n = 16$. In 1989, Cantor [7] independently proposed the same algorithm but showed how to get a special basis that allows the FFT to be efficiently computed for any $n = p^m$ where p is the characteristic of the underlying finite field and m is a power of p. In particular, Cantor gives an algorithm for computing FFT of length $n = 2^m$ over $\mathbb{F}_{2^m}$ when m is a power of 2, using at most $\frac{1}{2}n\log_2(n)$ multiplications and $\frac{1}{2}n\log_2^{1.585}(n) + n\log_2(n)$ additions. In 1996, von zur Gathen and Gerhard [9] generalized Cantor's approach and presented an additive FFT of length $n = 2^m$ for any m, using $O(n\log^2(n))$ multiplications and $O(n\log^2(n))$ additions.

In this paper, we present two new algorithms for computing additive FFTs over finite fields of characteristic two. The concept of the Taylor expansion learned in a typical calculus course can be generalized as discussed in [8]. We first present an algorithm for computing one such Taylor expansion that plays a central role in our FFT algorithms. For $n = 2^m$ where m is arbitrary, we provide an algorithm that requires fewer operations than the von zur Gathen-Gerhard algorithm (specifically, at most $\frac{1}{2}n\log^2(n)$ additions but only $2n\log(n)$ multiplications). For $n = 2^m$ and m is a power of 2, we then present an algorithm that requires fewer operations than the Wang-Zhu-Cantor algorithm (specifically, at most $\frac{1}{2}n\log(n)$ multiplications but only $\frac{1}{2}n\log(n)\log\log(n) + n\log_2(n)$ additions).

We mention some related work before proceeding. One application of FFT algorithms is the efficient multiplication of two polynomials with coefficients over a field $\mathbb{F}$. If $f(x), g(x) \in \mathbb{F}[x]$ and the sum of the degrees of $f(x)$ and $g(x)$ is less than n, then one can use an FFT algorithm to efficiently evaluate $f(x)$ and $g(x)$ at each of the points of a subspace of $\mathbb{F}$ of size n, pointwise multiply the evaluations, and then use an inverse FFT algorithm to interpolate the evaluations into the product polynomial. In the case of polynomials with complex number coefficients, the multiplicative FFT is used for this operation while for finite field polynomials, the additive FFT would be used instead. In either case, the polynomial multiplication requires roughly three times the number of operations of the FFT algorithm. In [9], [10], a technique is described that allows polynomials over $\mathbb{F}_2$ to be multiplied using this technique by mapping them to $\mathbb{F}_{2^m}$ and back. If the new additive FFT algorithm is used, then the product of two binary polynomials can often be computed in fewer operations than Schönhage's algorithm. Further analysis comparing the two multiplication algorithms can be found in [11].

The additive FFT algorithms can also be used in a simple Reed-Solomon decoding algorithm proposed independently by Shiozaki [5] and Gao [4]. These decoding algorithms are for nonsystematic Reed-Solomon codes. Algorithms for the systematic case are described in [12] where it is shown that the simple algorithm is equivalent to traditional decoding methods.

Also, Chen and Yan [13] recently proposed a cyclotomic FFT over fields of characteristic two and applied them to the decoding of Reed-Solomon codes [14]. It would interest to compare the efficiency of our additive FFT to their cyclotomic FFT for a wide range of problem sizes.

The first author was partially supported by National Science Foundation under grants DMS-0302549 and CCF-0830481 and National Security Agency under grant H98230-08-1-0030.

Shuhong Gao is with the Department of Mathematical Sciences, Clemson University, Clemson, SC, 29634-0975 USA e-mail: sgao@clemson.edu.

Todd Mateer is with the Mathematics Division, Howard Community College, 10901 Little Patuxent Parkway, Columbia, MD 21044 USA email: tmateer@howardcc.edu.

Manuscript received XXXX; revised XXXXX.

```
Algorithm 1 : Taylor expansion at $x^t - x$
Input:    $(f, n, t)$ where $n \geq 1$ and $t > 1$, and $f(x) \in \mathbb{F}[x]$ of degree $< n$.
Output:   $\mathrm{T}(f, n, t)$, the Taylor expansion of $f(x)$ at $x^t - x$.
Step 0.   If $n \leq t$ then return $f(x)$.
Step 1.   Find $k$ such that $t2^k < n \leq 2 \cdot t2^k$.
Step 2.   Split $f(x)$ into three blocks as
          $f(x) = f_0(x) + x^{t2^k}(f_1(x) + x^{(t-1)2^k} f_2(x))$
          where $\deg f_0 < t2^k$,   $\deg f_1 < t2^k - 2^k$,   $\deg f_2 < 2^k$.
          Compute
              $h := f_1 + f_2$,   $g_0 := f_0 + x^{2^k} h$,   $g_1 = h + x^{(t-1)2^k} f_2$.
Step 3.   Apply the algorithm recursively to compute
              $V_1 = \mathrm{T}(g_0, t2^k, t)$ and $V_2 = \mathrm{T}(g_1, n - t2^k, t))$.
Step 4.   Return $(V_1, V_2)$.
```

II. Taylor expansion

To present our additive FFT algorithms, we need to be able to compute a generalized Taylor expansion of polynomials; see [8] for more general expansion. Let $\mathbb{F}$ be any field of characteristic two, $t > 1$ any integer, and $f(x) \in \mathbb{F}[x]$ of degree $< n$. We want to find polynomials $h_0(x), h_1(x), \ldots, h_{m-1}(x) \in \mathbb{F}[x]$ such that

$$f(x) = h_0(x) + h_1(x) \cdot (x^t - x) + \cdots + h_{m-1}(x) \cdot (x^t - x)^{m-1}$$

where $m = \lceil n/t \rceil$ and $\deg h_i < t$ for $0 \leq i \leq m-1$. We will call this expression the Taylor expansion of $f(x)$ at $x^t - x$, and denote it by

$$\mathrm{T}(f, n, t) = (h_0, h_1, \ldots, h_{m-1}).$$

To see how to compute such a Taylor expansion, let k be such that

$$2^k < \frac{n}{t} \leq 2^{k+1}.$$

Write $f(x)$ as $f(x) = f_0(x) + x^{t2^k}(f_1(x) + x^{(t-1)2^k} f_2(x))$ where

$$\deg f_0 < t2^k,$$
$$\deg f_1 < \min(n - t2^k, (t-1)2^k) \leq (t-1)2^k,$$
$$\deg f_2 < 2^k.$$

Note that $f_2(x)$ is zero when $n - t2^k < (t-1)2^k$. Since $\mathbb{F}$ has characteristic two, we have

$$x^{t2^k} = (x^t - x)^{2^k} + x^{2^k},$$

thus

$$\begin{aligned} f(x) &= f_0(x) + x^{2^k}(f_1(x) + f_2(x)) \\ &\quad + (x^t - x)^{2^k}(f_1(x) + f_2(x) + x^{(t-1)2^k} f_2(x)). \end{aligned}$$

Set $h(x) = f_1(x) + f_2(x)$, and

$$g_0(x) = f_0(x) + x^{2^k} h(x), \quad g_1(x) = h(x) + x^{(t-1)2^k} f_2(x).$$

Then

$$f(x) = g_0(x) + g_1(x)(x^t - x)^{2^k}.$$

Since $\deg f_1 < (t-1)2^k$ and $\deg f_2 < 2^k$, we have $\deg h < (t-1)2^k$, so

$$\deg g_0 < t2^k, \quad \deg g_1 < n - t2^k.$$

Therefore,

$$\mathrm{T}(f, n, t) = (\mathrm{T}(g_0, t2^k, t), \mathrm{T}(g_1, n - t2^k, t)).$$

Note that if $n = t2^{k+1}$, then both $g_0(x)$ and $g_1(x)$ have size $n/2$. Thus, a problem of size n is reduced to two problems of size $n/2$. If n is not of the form $t2^{k+1}$, then $g_0(x)$ has size $t2^k$ and $g_1(x)$ has size $n - t2^k \leq n/2$. We can apply this procedure recursively to $g_0(x)$ and $g_1(x)$, separately, and in at most $k+1$ steps all the polynomials will have degrees $< t$ and we obtain the Taylor expansion of $f(x)$. We summarize this as Algorithm 1.

To see the time complexity of Algorithm 1, we may pad 0's if necessary and assume that $f(x)$ has exactly $t2^{k+1}$ coefficients. Then $f_0(x)$ has $t2^k$ coefficients, $f_1(x)$ has $(t-1)2^k$ coefficients, $f_2(x)$ has 2^k coefficients, and thus $h(x)$ has $(t-1)2^k$ coefficients. Then 2^k additions in $\mathbb{F}$ are needed to compute $h(x)$, as well as $(t-1)2^k$ additions for $g_0(x)$ and 0 for $g_1(x)$. Hence the total number of additions is $2^k + (t-1)2^k = t2^k$. Now both $g_0(x)$ and $g_1(x)$ have length $t2^k$, so the reduction for each of $g_0(x)$ and $g_1(x)$ will need $t2^{k-1}$ additions, so the reduction for both needs $2 \cdot t2^{k-1} = t2^k$ additions. Inductively, we see that, for i from k down to 0, there are 2^{k-i} reductions from size $t2^{i+1}$ to size $t2^i$, each of which costs $t2^i$ additions. For each i, the number of additions is $2^{k-i} \cdot t2^i = t2^k$. Therefore, the total number of additions used by Algorithm 1 is at most $t2^k(k+1)$, which is

$$\begin{cases} \leq n\lceil \log_2(n/t) \rceil, & \text{for any } n, \\ = \frac{1}{2} n \lceil \log_2(n/t) \rceil, & \text{when } n/t \text{ is a power of two.} \end{cases} \tag{1}$$

III. FFT over $\mathbb{F}_{2^m}$: arbitrary m

Let $\mathbb{F}$ be any field of characteristic two. Assume that $\beta_1, \ldots, \beta_m \in \mathbb{F}$ are linearly independent over $\mathbb{F}_2$. Let B be the subspace spanned by β_i's over $\mathbb{F}_2$, namely,

$$B = \langle \beta_1, \ldots, \beta_m \rangle = \{a_1\beta_1 + \cdots + a_m\beta_m : a_1, \ldots, a_m \in \mathbb{F}_2\}.$$

We order the elements of B as follows. For any $0 \leq i < 2^m$, suppose the binary representation of i is

$$i = a_1 + a_2 \cdot 2 + \cdots + a_m \cdot 2^{m-1} = (a_1, a_2, \cdots, a_m)_2,$$

where each $a_j = 0$ or 1. Then the ith element of B is

$$B[i] = a_1\beta_1 + a_2\beta_2 + \cdots + a_m\beta_m. \tag{2}$$

```
Algorithm 2 : Additive FFT of length $n = 2^m$ (arbitrary $m$)
Input:    $(f, m, B)$ where $m \geq 1$, $f(x) \in \mathbb{F}[x]$ of degree $< n = 2^m$, and
          $B = \langle \beta_1, \ldots, \beta_m \rangle$, where $\beta_i$'s are linearly independent over $\mathbb{F}_2$.
Output:   FFT$(f, m, B) = (f(B[0]), f(B[1]), \ldots, f(B[n-1]))$.
Step 1.   If $m = 1$ then return $(f(0), f(\beta_1))$.
Step 2.   Compute $g(x) = f(\beta_m x)$.
Step 3.   Compute the Taylor expansion of $g(x)$ as in (3) and
          let $g_0(x)$ and $g_1(x)$ be as in (4).
Step 4.   Compute $\gamma_i = \beta_i \cdot \beta_m^{-1}$ and $\delta_i = \gamma_i^2 - \gamma_i$ for $1 \leq i \leq m-1$.
          Let $G = \langle \gamma_1, \ldots, \gamma_{m-1} \rangle$, and $D = \langle \delta_1, \ldots, \delta_{m-1} \rangle$.
Step 5    Let $k = 2^{m-1}$. Compute
              FFT$(g_0, m-1, D) = (u_0, u_1, \ldots, u_{k-1})$, and
              FFT$(g_1, m-1, D) = (v_0, v_1, \ldots, v_{k-1})$.
Step 6.   For $0 \leq i < 2^{m-1}$, set $w_i = u_i + G[i] \cdot v_i$ and $w_{k+i} = w_i + v_i$.
Step 7.   Return $(w_0, w_1, \ldots, w_{n-1})$.
```

Suppose that we are given a polynomial $f(x)$ of degree less than $n = 2^m$. We wish to evaluate $f(x)$ at each of the points in B, called the FFT of f over B, denoted as

$$\text{FFT}(f, m, B) = (f(B[0]), f(B[1]), \ldots, f(B[n-1])).$$

We show how to reduce such a problem of size $n > 1$ to two problems of size $k = n/2 = 2^{m-1}$. Define

$$\gamma_i = \beta_i \cdot \beta_m^{-1}, \quad 1 \leq i \leq m-1,$$

and

$$\begin{aligned} G &= \langle \gamma_1, \cdots, \gamma_{m-1} \rangle \\ &= \{a_1\gamma_1 + \cdots + a_{m-1}\gamma_{m-1} : a_1, \ldots, a_{m-1} \in \mathbb{F}_2\}. \end{aligned}$$

Let $g(x) = f(\beta_m x)$, the "twisted" or weighted polynomial of $f(x)$. Then evaluating $f(x)$ over B is the same as evaluating $g(x)$ over $B \cdot \beta_m^{-1} = G \cup (G+1)$, that is,

$$\text{FFT}(f, m, B) = (\text{FFT}(g, m-1, G), \text{FFT}(g, m-1, G+1)),$$

where $G + 1 = \{\alpha + 1 : \alpha \in G\}$.

We need to show how to compute FFT(g, G) and FFT$(g, G+1)$. Define

$$\delta_i = \gamma_i^2 - \gamma_i, \quad 1 \leq i \leq m-1,$$

and

$$\begin{aligned} D &= \langle \delta_1, \ldots, \delta_{m-1} \rangle \\ &= \{a_1\delta_1 + \cdots + a_{m-1}\delta_{m-1} : a_1, \ldots, a_{m-1} \in \mathbb{F}_2\}. \end{aligned}$$

Since $\gamma_1, \ldots, \gamma_{m-1}$ and 1 are linearly independent over $\mathbb{F}_2$, the new elements $\delta_1, \ldots, \delta_{m-1}$ are linearly independent over $\mathbb{F}_2$, so D is a subspace of $\mathbb{F}$ of size $k = 2^{m-1} = n/2$. For each $\alpha = a_1\gamma_1 + \cdots + a_{m-1}\gamma_{m-1} \in G$, let

$$\alpha^* = \alpha^2 - \alpha = a_1\delta_1 + \cdots + a_{m-1}\delta_{m-1} \in D.$$

Then we have

$$G[i]^* = D[i], \quad 0 \leq i < k,$$

where $G[i]$ and $D[i]$ are the ith elements of G and D, respectively, which are ordered according to the binary representation of i in a similar fashion as that described above for B.

Suppose the Taylor expansion of $g(x)$ at $x^2 - x$ is

$$g(x) = \sum_{i=0}^{k-1} (g_{i0} + g_{i1}x) \cdot (x^2 - x)^i \tag{3}$$

where $g_{ij} \in \mathbb{F}$. Let

$$g_0(x) = \sum_{i=0}^{k-1} g_{i0} \cdot x^i, \quad \text{and} \quad g_1(x) = \sum_{i=0}^{k-1} g_{i1} \cdot x^i. \tag{4}$$

For any $\alpha \in G$ and $b \in \mathbb{F}_2$, since $(\alpha + b)^2 - (\alpha + b) = \alpha^*$, we have,

$$g(\alpha + b) = \big(g_0(\alpha^*) + \alpha \cdot g_1(\alpha^*)\big) + bg_1(\alpha^*). \tag{5}$$

Hence the FFT of $g(x)$ can be obtained from those of $g_0(x)$ and $g_1(x)$ as follows. Let the FFT of $g_0(x)$ and $g_1(x)$ over D be

$$\text{FFT}(g_0, m-1, D) = (u_0, u_1, \ldots, u_{k-1}),$$

$$\text{FFT}(g_1, m-1, D) = (v_0, v_1, \ldots, v_{k-1}),$$

where $u_i = g_0(D[i])$ and $v_i = g_1(D[i])$, $0 \leq i < k$. Then the equation (5) implies that

$$\text{FFT}(g, m-1, G) = (w_0, w_1, \ldots, w_{k-1}),$$

where $w_i = u_i + G[i] \cdot v_i$ for $0 \leq i < k$, and

$$\text{FFT}(g, m-1, G+1) = \text{FFT}(g, m-1, G) + \text{FFT}(g_1, m-1, D).$$

Applying this reduction step repeatedly, we obtain a fast algorithm for computing additive FFT. This is summarized in Algorithm 2.

Let us compute the cost of this algorithm. In computing the basis elements in G and D in Step 4, the number of multiplications is $2(m-1) + 2(m-2) + \cdots + 2 \cdot 1 = m(m-1) = O(\log_2^2(n))$, and the number of additions is $(m-1) + (m-2) + \cdots + 1 = m(m-1)/2 = O(\log_2^2(n))$. Also, in Step 2, we need to compute the powers $\beta_m^i, 2 \leq i \leq n-1$, for which the total number of multiplications is at most $(2^m - 2) + (2^{m-1} - 2) + \cdots + (2^2 - 2) < 2 \cdot 2^m = 2n$. This part can be precomputed or computed as needed. In either case, the cost is negligible.

Next, we will consider the number of operations required by the other steps. Step 1 is used to end the recursion and costs 1 multiplication and 1 addition. Step 2 costs $n-1$ multiplications (besides computing powers of β_m). By letting $t = 2$ in (1), Step 3 costs $\frac{1}{2} \cdot n \cdot \log_2(n) - \frac{1}{2} \cdot n$ additions. The cost of Step 5 is the number of operations needed to compute an additive FFT of size $n/2$. The cost of Step 6 is n multiplications and n additions. Step 7 costs no operations. Let $M(n)$ and $A(n)$ denote the numbers of multiplications and additions, respectively, used by the algorithm for input of length n. Then $M(2) = 1$, $A(2) = 1$, and, for any $n = 2^m > 2$, the reduction implies that

$$\begin{aligned} M(n) &= 2 \cdot M\left(\frac{n}{2}\right) + 2n - 1, \\ A(n) &= 2 \cdot A\left(\frac{n}{2}\right) + \frac{1}{2} \cdot n \cdot \log_2(n) + \frac{1}{2} \cdot n. \end{aligned}$$

By induction, we have

$$\begin{aligned} M(n) &= 2 \cdot n \cdot \log_2(n) - 2n + 1, \\ A(n) &= \frac{1}{4} \cdot n \cdot (\log_2(n))^2 + \frac{3}{4} \cdot n \cdot \log_2(n) - \frac{1}{2} \cdot n. \end{aligned}$$

The new algorithm significantly reduces the number of multiplications compared to the $\Theta(n \cdot (\log_2(n))^2)$ multiplications required in [9]. The number of additions is slightly reduced compared to [9].

IV. FFT over $\mathbb{F}_{2^m}$: m a power of two

Suppose $\mathbb{F}$ contains a subfield of 2^m elements where m is a power of 2. Let $n = 2^m$ and $f \in \mathbb{F}[x]$ of degree less than n. In this section, we present a faster algorithm to compute the FFT of f over $\mathbb{F}_{2^m}$. Our strategy is to reduce a problem of size n to $2\sqrt{n}$ problems each of size $\sqrt{n}$, and apply the technique recursively. Our algorithm makes use of a special basis introduced by Cantor [7]. Choose any any element $\beta_m \in \mathbb{F}_{2^m}$ that has trace 1 in $\mathbb{F}_2$. Then define

$$\beta_{i-1} = \beta_i^2 + \beta_i, \quad 1 \le i < m. \tag{6}$$

Particularly, $\beta_1 = 1$. Also, $\beta_1, \ldots, \beta_m$ form a linear basis for $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. See the appendix for proof of these facts and others used below.

Let us enumerate the $n = 2^m$ elements of $\mathbb{F}_{2^m}$ as follows. For $0 \le i < n = 2^m$, write i in binary form, i.e.,

$$i = i_1 + i_2 \cdot 2 + \cdots + i_m \cdot 2^{m-1} = (i_1, i_2, \cdots, i_m)_2,$$

where each $i_j = 0$ or 1. Then the ith element of $\mathbb{F}_{2^m}$ is

$$\varpi_i = i_1 \cdot \beta_1 + i_2 \cdot \beta_2 + \cdots + i_m \cdot \beta_m. \tag{7}$$

It is useful to observe that $\varpi_{i2^k+j} = \varpi_{i2^k} + \varpi_j$ whenever $j < 2^k$.

For each $1 \le k \le m$, let us define the subspace W_k of $\mathbb{F}_{2^m}$ to be all linear combinations of $\beta_1, \beta_2, \ldots, \beta_k$ over $\mathbb{F}_2$, that is,

$$W_k = \langle \beta_1, \beta_2, \ldots, \beta_k \rangle = \{\varpi_i : 0 \le i < 2^k\}.$$

For example, $W_1 = \{0, 1\} = \{\varpi_0, \varpi_1\}$. These subspaces form a strictly ascending chain of subspaces:

$$W_1 \subset W_2 \subset W_3 \subset W_4 \subset \cdots \subset W_m = \mathbb{F}_{2^m}. \tag{8}$$

Denote the vanishing polynomial of W_k by $s_k(x)$, that is,

$$s_k(x) = \prod_{a \in W_k} (x - a).$$

In particular, $s_1(x) = x(x-1) = x^2 + x$. By Lemma 1 in the appendix, $s_k(x)$ is a 2-linearized polynomial, and

$$s_{k+1}(x) = s_k^2(x) - s_k(x), \quad k \ge 1.$$

Furthermore, $s_k(x) = x^{2^k} + x$ whenever k is a power of 2. Also, if k is a power of 2 and $i2^k < n = 2^m$ then

$$(\varpi_{i2^k})^{2^k} + \varpi_{i2^k} = \varpi_i. \tag{9}$$

Now we are ready to describe the reduction step of our algorithm. Let k be any power of two with $1 < 2k \le m$. Let $t = 2^k$ and $T = t^2 = 2^{2k} \le n$. Our reduction is based on the following factorization

$$x^{t^2} - x - c = \prod_{b \in W_k} (x^t - x - a - b) \tag{10}$$

where $a, c \in \mathbb{F}$ such that $a^t + a = c$. This follows from the identities

$$x^t + x = \prod_{b \in W_k} (x - b), \quad \text{and}$$

$$x^{t^2} - x - c = (x^t - x - a)^t + (x^t - x - a).$$

Let $f(x)$ be any polynomial in $\mathbb{F}[x]$ of degree $< T$. We want to compute the values of $f(x)$ at the roots of $x^T - x - c$. We first compute the Taylor expansion of $f(x)$ at $x^t - x$ to get

$$f(x) = g_0(x) + g_1(x)(x^t - x) + \cdots + g_{t-1}(x)(x^t - x)^{t-1} \tag{11}$$

where $g_i(x) \in \mathbb{F}[x]$ has degree $< t$. Then, for any $\omega \in \mathbb{F}$, we have

$$f(x) \equiv g_0(x) + g_1(x)\omega + \cdots + g_{t-1}(x)\omega^{t-1} \pmod{x^t - x - \omega}.$$

Suppose

$$g_i(x) = \sum_{j=0}^{t-1} g_{ij} x^j, \quad 0 \le i \le t-1,$$

where $g_{ij} \in \mathbb{F}$. Define

$$h_j(x) = \sum_{i=0}^{t-1} g_{ij} x^i, \quad 0 \le j \le t-1.$$

Then

$$\begin{aligned} & g_0(x) + g_1(x) \cdot \omega + \cdots + g_{t-1}(x) \cdot \omega^{t-1} \\ = \; & h_0(\omega) + h_1(\omega) \cdot x + \cdots + h_{t-1}(\omega) \cdot x^{t-1}. \end{aligned}$$

Hence $f(x) \bmod x^{2^k} - x - \omega$, $\omega \in a + W_k$, can be obtained by computing the additive FFT of the t polynomials $h_j(x)$ over $a + W_k$. Since k is a power of 2, we have $s_k(x) = x^{2^k} + x$ and the vanishing polynomial of $a + W_k$ is

$$s_k(x-a) = s_k(x) - s_k(a) = x^{2^k} - x - (a^{2^k} - a) = x^{2^k} - x - c.$$

Hence evaluating $h_j(x)$ over $a + W_k$ is equivalent to $h_j(x) \bmod x^{2^k} - x - c$.

```
Algorithm 3 : FFT of length $n = 2^m$ ($m$ a power of 2)
Input:   $(f, n, s)$ where $s = 0$ initially and $f(x) \in \mathbb{F}[x]$ of degree $< n$,
Output:  $\mathrm{FFT}(f, n, s) = (f(\varpi_{sn}), f(\varpi_{sn+1}), \ldots, f(\varpi_{sn+n-1}))$,
         the FFT of $f(x)$ over $\varpi_s + W_m$
Step 1.  If $n = 2$ then return $(f(\varpi_{2s}), f(\varpi_{2s+1}))$.
Step 2.  Let $t$ be such that $t^2 = n$. Compute the Taylor expansion of $f(x)$
         at $x^t - x$ to get a matrix $G$ as in (12).
Step 3.  Column FFT of $G$: for $1 \le j \le t$, let $h_j$ be the $j$-column of $G$,
         compute $\mathrm{FFT}(h_j, t, st)$, a column vector denoted by $C_j$,
         update the $j$-th column of $G$ by $C_j$.
Step 4.  Row FFT of $G$: for $1 \le i \le t$, let $g_i$ be the $i$-th row of $G$,
         compute $\mathrm{FFT}(g_i, t, sn + (i-1)t)$, a row vector denoted by $R_i$.
Step 5.  Return $(R_1, R_2, \ldots, R_t)$.
```

The Taylor expansion (11) of f can be represented by the following matrix

$$G = \begin{pmatrix} g_{0\,0} & g_{0\,1} & \cdots & g_{0\,t-1} \\ g_{1\,0} & g_{1\,1} & \cdots & g_{1\,t-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{t-1\,0} & g_{t-1\,1} & \cdots & g_{t-1\,t-1} \end{pmatrix}. \tag{12}$$

Then g_i corresponds to the i-th row and $h_j(x)$ corresponds to the j-th column of G. Hence our reduction amounts to the following: for each column, perform an FFT over $x^{2^k} - x - c$, and then, for $0 \le j \le t-1$, compute the FFT of the i-th row over $x^t - x - a - \varpi_i$. This means that our reduction uses one Taylor expansion and reduces a problem of size t^2 to $2t$ problems of size t.

To implement the above reduction, we need to see how the indices of elements change in term of the representation in (7). Let $c = \varpi_s$ in (10) where $s < 2^{m-2k} = n/T$. By (9), we can take $a = \varpi_{st}$. Thus,

$$\begin{aligned} x^{t^2} - x - \varpi_s &= \prod_{b \in W_k} (x^t - x - \varpi_{st} - b) \\ &= \prod_{i=0}^{t-1} (x^t - x - \varpi_{st+i}). \end{aligned}$$

Note that the set of roots of $x^{t^2} - x - \varpi_s$ is $\varpi_{st^2} + W_{2k}$, i.e.,

$$\{\varpi_{st^2+it+j} : 0 \le i, j < t\},$$

that of $x^t - x - \varpi_{st+i}$ is $\varpi_{(st+i)t} + W_k = \varpi_{st^2+it} + W_k$, i.e.,

$$\{\varpi_{st^2+it+j} : 0 \le j < t\},$$

and that of $x^t - x - \varpi_s$ is $\varpi_{st} + W_k$, i.e.,

$$\{\varpi_{st+j} : 0 \le j < t\}.$$

Hence our reduction step reduces an FFT over $\varpi_{st^2} + W_{2k}$ to t FFTs over $\varpi_{st} + W_k$, which corresponds to FFT on the columns of G, and an FFT over $\varpi_{st^2+it} + W_k$, FFT of the i-th row of G, for $i = 0, 1, \ldots, t-1$. Note that in the matrix G above, after computed the row and column FFTs, one should read off the entries of the final matrix G by rows starting at the first row. Pseudocode for the new additive FFT is given in Algorithm 3. In the algorithm, we identify a polynomial of degree $< n$ with a vector of length n, the FFT of a row vector is a row vector, and the FFT of a column vector is a column vector.

Now we estimate the numbers of $\mathbb{F}$-operations used by our algorithm. Suppose $n = 2^m$ with $m = 2^\ell$. For $0 \le i \le \ell$, let $A(i)$ denote the number of additions in $\mathbb{F}$ used by the algorithm for input length 2^{2^i}, and similarly $M(i)$ for the number of multiplications used. Note that $i = 0$ corresponds to Step 1 where $f(x)$ is of the form $ax+b \in \mathbb{F}[x]$. As $\varpi_{2s+1} = \varpi_{2s} + 1$, we have $f(\varpi_{2s+1}) = f(\varpi_{2s}) + a$. Hence $f(\varpi_{2s})$ and $f(\varpi_{2s+1})$ can be computed using one multiplication and two additions in $\mathbb{F}$, that is, $A(0) = 2$ and $M(0) = 1$. For each i from ℓ down to 1, our reduction step involves a Taylor expansion with input length 2^{2^i} at $x^{2^{2^{i-1}}} - x$. By our argument in Section II (with $n = 2^{2^i}$ and $t = 2^{2^{i-1}}$), the number of $\mathbb{F}$-additions used is $2^{2^i} \cdot 2^{i-1}$. Hence we have, for $1 \le i \le \ell$,

$$M[i] = 2 \cdot 2^{2^{i-1}} \cdot M[i-1], \quad A[i] = 2 \cdot 2^{2^{i-1}} \cdot A[i-1] + 2^{2^i} \cdot 2^{i-1}.$$

By induction, we see that

$$\begin{aligned} M(\ell) &= 2^{2^\ell - 1} \cdot 2^\ell = \frac{1}{2} \cdot n \cdot \log_2(n), \\ A(\ell) &= n \cdot \log_2(n) + \frac{1}{2} \cdot n \cdot \log_2(n) \cdot \log_2 \log_2(n). \end{aligned}$$

So the new algorithm requires the same number of multiplications as the Wang-Zhu-Cantor algorithm, but the number of additions has been reduced. The new algorithm is said to be $\Theta(n \cdot \log_2(n) \cdot \log_2 \log_2(n))$.

V. Concluding remarks

We presented two new additive FFT algorithms over finite fields of characteristic two. The first FFT can evaluate polynomials over any additive subspace of size $n = 2^m$ where m is arbitrary. This algorithm improves upon von zur Gathen and Gerhard's approach in that the number of multiplications required is reduced by a log factor while preserving the number of additions.

When $n = 2^m$ and m is a power of 2, we present a more efficient algorithm for FFT over $\mathbb{F}_{2^m}$. The new additive FFT requires the same number of multiplications and fewer additions than the Wang-Zhu-Cantor FFT discussed at the beginning of this paper. The actual running time of an algorithm on a particular computer depends on other factors besides the number of arithmetical operations. Therefore, one should carefully implement both techniques to determine which one is actually faster on a particular architecture.

The inverse FFT efficiently interpolates a collection of evaluations of a polynomial back into the polynomial. An inverse additive FFT can be constructed by performing the inverse of each operation of an additive FFT algorithm in reverse order. Each inverse FFT algorithm can be shown to require the same number of operations as the companion FFT algorithm.

Finally, all the algorithms in this paper can be implemented as parallel algorithms (no recursive calls). For FFT of length n, they need to store only n field elements (for the coefficients), except that Algorithm 2 may need to store extra elements for the δ's if they are precomputed (so no need to compute them during the execution of the algorithm). This means that our algorithms are well suitable for hardware implementation.

Appendix

In this appendix, we prove a few properties needed by our algorithms. Some of these properties can be found in Cantor's paper [7], but we present them in a simplified fashion that may be more accessible to the reader. We shall work over a field $\mathbb{F}$ of characteristic two where the identity $(x+y)^2 = x^2+y^2$ is valid. Let $\phi(x) = x^2 + x$ and

$$\phi^{i+1}(x) = \phi(\phi^i(x)), \quad i \ge 1.$$

Define

$$s_i(x) = \phi^i(x) \in \mathbb{F}[x], \quad i \ge 1. \tag{13}$$

Then $s_1(x) = x^2 + x$, and

$$s_{i+1}(x) = s_i^2(x) + s_i(x), \quad i \ge 1.$$

By induction, we have

$$s_i(x) = \sum_{k=0}^{i} \binom{i}{k} x^{2^k}. \tag{14}$$

Here we used the properties that

$$\binom{i}{k} + \binom{i}{k-1} = \binom{i+1}{k}, \quad \binom{i}{k}^2 \equiv \binom{i}{k} \pmod 2.$$

To see the number of nonzero terms in $s_i(x)$, we recall Lucas' Lemma. Represent i and k in binary form:

$$i = i_1 + i_2 \cdot 2 + \cdots + i_m \cdot 2^{m-1}, \quad k = k_1 + k_2 \cdot 2 + \cdots + k_m \cdot 2^{m-1},$$

where $i_j, k_j \in \{0,1\}$. Then

$$\binom{i}{k} \equiv \binom{i_1}{k_1} \cdot \binom{i_2}{k_2} \cdots \binom{i_m}{k_m} \pmod 2.$$

Note that $\binom{i_j}{k_j} = 0$ whenever $k_j > i_j$. Hence, in $\mathbb{F}$, we have $\binom{i}{k} = 1$ if and only if $k_j \le i_j$ for all $1 \le j \le m$. Thus, if the binary representation of i has t 1's, then $s_i(x)$ has 2^t nonzero terms. In particular,

$$s_i(x) = x^{2^i} + x \quad \text{if any only if } i \text{ is a power of 2 .}$$

Next we define a sequence of elements in the algebraic closure of $\mathbb{F}_2$: $\beta_1 = 1$, and

$$\beta_{i+1}^2 + \beta_{i+1} = \beta_i, \quad i = 1, 2, \ldots.$$

Note that, for each i, the equation $x^2 + x = \beta_i$ has two solutions (in the algebraic closure of $\mathbb{F}_2$), so β_{i+1} has two choices, either of which will work in our algorithms. For each $i \ge 0$, let $i = i_1 + i_2 \cdot 2 + \cdots + i_m \cdot 2^{m-1}$ be its binary representation, and define

$$\varpi_i = i_1 \cdot \beta_1 + i_2 \cdot \beta_2 + \cdots + i_m \cdot \beta_m. \tag{15}$$

Note that $\varpi_{i2^k+j} = \varpi_{i2^k} + \varpi_j$ whenever $j < 2^k$. These properties are summarized as follows.

Lemma 1: (a) $s_k(\beta_{i+k}) = \beta_i$ for all $i, k \ge 1$;

(b) $s_i(\beta_{i+1}) = \beta_1 = 1$ for $i \ge 1$;

(c) $s_{k+1}(x) = s_k(x)^2 + s_k(x)$ for all $k \ge 1$, and the roots of $s_k(x)$ are the $\mathbb{F}_2$-linear subspace

$$W_k = \langle \beta_1, \beta_2, \ldots, \beta_k \rangle = \{\varpi_i : 0 \le i < 2^k\};$$

(d) $\varpi_{i2^k}^{2^k} + \varpi_{i2^k} = \varpi_i$ for any k being a power of two and $i \ge 1$.

We outline the proof. By definition, $\beta_i = \phi(\beta_{i+1})$ for all $i \ge 1$, thus, by induction, we have

$$\beta_i = \phi^k(\beta_{i+k}), \quad \text{for all } i, k \ge 1.$$

Then part (a) follows, as $s_k(\beta_{i+k}) = \phi^k(\beta_{i+k})$, and (b) is just a special case of (a). Part (c) follows by induction: it's true for $k = 1$; and if it is true for k then

$$\begin{aligned} \prod_{a \in W_{k+1}} (x - a) &= \prod_{a \in W_k} (x - a) \prod_{a \in W_k} \big(x - (a + \beta_{k+1})\big) \\ &= s_k(x) s_k(x - \beta_{k+1}) \\ &= s_k(x)\big(s_k(x) - s_k(\beta_{k+1})\big) \\ &= s_k(x)\big(s_k(x) - 1\big) = s_{k+1}(x). \end{aligned}$$

For part (d), suppose $\varpi_i = i_1 \cdot \beta_1 + i_2 \cdot \beta_2 + \cdots + i_m \cdot \beta_m$. Then

$$\varpi_{i2^k} = i_1 \cdot \beta_{1+k} + i_2 \cdot \beta_{2+k} + \cdots + i_m \cdot \beta_{m+k}.$$

So, if k is a power of two, then $s_k(x) = x^{2^k} + x$, and

$$\varpi_{i2^k}^{2^k} + \varpi_{i2^k} = s_k(\varpi_{i2^k}) = \sum_{j=1}^{m} i_j s_k(\beta_{j+k}) = \sum_{j=1}^{m} i_j \beta_j = \varpi_i.$$

This completes the proof.

Lemma 2: For each $m \ge 1$, the elements $\beta_1, \beta_2, \ldots, \beta_m$ are linearly independent over $\mathbb{F}_2$.

It follows by induction on m, as it's true for $m = 1$, and any linear relation among $\beta_1, \ldots, \beta_m, \beta_{m+1}$ implies a linear relation among $\beta_1, \ldots, \beta_m$ via the map ϕ.

Next we determine the subfields where the elements β_m's lie. First consider the case $m = 2^k$. By Lemma 1 (c) , β_k is a root of $s_m(x) = x^{2^m} + x$. Hence $\beta_m \in \mathbb{F}_{2^m}$, which implies that $\beta_i \in \mathbb{F}_{2^m}$ for all $i \le m$. Since $m - 1 = 1 + 2 + 2^2 + \cdots + 2^{k-1}$, we have

$$s_{m-1}(x) = \sum_{j=0}^{m-1} x^{2^j}.$$

Hence $s_{m-1}(\beta_m)$ is the trace of $\beta_m \in \mathbb{F}_{2^m}$ into $\mathbb{F}_2$. By Lemma 1 (b), $s_{m-1}(\beta_m) = \beta_1 = 1$, that is, β_m has trace 1 in $\mathbb{F}_2$. Since m is a power of 2, β_m can not be any proper subfield of $\mathbb{F}_{2^m}$ (which would imply that β_m has trace 0). Therefore, β_m has degree m over $\mathbb{F}_2$ whenever m is a power of two.

Now let $i \geq 1$. Suppose $2^{k-1} < i \leq m = 2^k$. Then $\beta_i \in \mathbb{F}_{2^m} = \mathbb{F}_{2^{2^k}}$, but not in any proper subfield. In fact, the biggest subfield of $\mathbb{F}_{2^m}$ is $\mathbb{F}_{2^{2^{k-1}}}$. If $\beta_i \in \mathbb{F}_{2^{2^{k-1}}}$, then $\beta_j \in \mathbb{F}_{2^{2^{k-1}}}$ for all $1 \leq j \leq i$, hence W_i is contained in $\mathbb{F}_{2^{2^{k-1}}}$. But the number of elements in W_i is 2^i, which is bigger than $2^{2^{k-1}}$. A contradiction has been reached and we have just proved the following result:

Lemma 3: If $2^{k-1} < i \leq 2^k$, then $\beta_i \in \mathbb{F}_{2^{2^k}}$ but not in any smaller field.
In particular, we have $\beta_1 = 1 \in \mathbb{F}_2$, $\beta_2 \in \mathbb{F}_{2^2}$, $\beta_3, \beta_4 \in \mathbb{F}_{2^{2^2}}$, $\beta_5, \beta_6, \beta_7, \beta_8 \in \mathbb{F}_{2^{2^3}}$, and so on.

Finally, we give an explicit construction for the tower of subfields:

$$\mathbb{F}_2 = \mathbb{F}_{2^{2^0}} \subset \mathbb{F}_{2^{2^1}} \subset \mathbb{F}_{2^{2^2}} \subset \cdots \subset \mathbb{F}_{2^{2^k}} \subset \cdots .$$

Let $\alpha_0 = 1$ and, for each $k \geq 1$, α_k is defined such that

$$\alpha_k + \alpha_k^{-1} = \alpha_{k-1}. \tag{16}$$

Then α_k has degree 2^k over $\mathbb{F}_2$, that is, $\mathbb{F}_2[\beta_k] = \mathbb{F}_{2^{2^k}}$, for all $k \geq 1$. A natural basis for $\mathbb{F}_{2^{2^k}}$ over $\mathbb{F}_2$ is then

$$\{\alpha_1^{e_1}\alpha_2^{e_2}\cdots\alpha_k^{e_k} : e_j = 0 \text{ or } 1, 1 \leq j \leq k\}.$$

Under this basis, addition and multiplication are both easy to implement. Another major advantage of this representation is that the basis structure corresponds to the subfield structure.

We note that the minimal polynomial of α_k can be obtained as follows. First define two sequences of polynomials $a_k(x)$ and $b_k(x)$ recursively as follows:

$$\begin{array}{rcl} a_0(x) = x, & & b_0(x) = 1, \\ a_{k+1}(x) & = & a_k(x) \cdot b_k(x), \\ b_{k+1}(x) & = & a_k^2(x) + b_k^2(x), \end{array}$$

for $k \geq 0$. Then the minimal polynomial of α_k is $a_k(x)+b_k(x)$ for $k \geq 0$. The proof of these results can be found in the book [15]; see particularly Theorem 3.10, Theorem 3.20, Corollary 3.22, and Research Problem 3.1.

The defining equation (16) implies that α_{k-1} is the trace of α_k from $\mathbb{F}_{2^{2^k}}$ to $\mathbb{F}_{2^{2^{k-1}}}$. By induction, we see that α_k has trace 1 in $\mathbb{F}_2$. So, for any $m = 2^k$, the desired sequence $\beta_1, \beta_2, \beta_3, \ldots, \beta_m$ in the above discussion can be obtained as follows: First set $\beta_m = \alpha_k$. Then for i from $m-1$ down to 1, define $\beta_i = \beta_{i+1}^2 + \beta_{i+1}$. This was the method used to obtain (6).

References

[1] Cooley, James W. and John W. Tukey. "An algorithm for the machine calculation of complex Fourier series", *Math. Comp.* 19 (1965), 297–301.

[2] Walker, James, S. *Fast Fourier Transforms*, Studies in Advanced Mathematics, CRC-Press, 2nd edition, 1996.

[3] Reed, I. S. and G. Solomon. "Polynomial codes over certain finite fields," *J. Soc. Indust. Appl. Math.* 8 (1960), 300–304.

[4] Gao, Shuhong. "A new algorithm for decoding Reed-Solomon codes" in "Communications, Information and Network Security", V. Bhargava, H. V. Poor, V. Tarokh, and S. Yoon, Eds. Norwell, MA: Kluwer, 2003, vol. 712, pp. 55-68.

[5] Shiozaki, A. "Decoding of redundant residue polynomial codes using Euclid's algorithm," *IEEE Trans. Inf. Theory*, vol. 34, no. 5, pp. 1351-1354, September 1988.

[6] Wang, Yao and Xuelong Zhu. "A fast algorithm for Fourier transform over finite fields and its VLSI implementation'.' *IEEE Journal on Selected Areas in Communications*, Vol. 6, No. 3, April 1988.

[7] Cantor, David G. "On arithmetical algorithms over finite fields, " *J. Combinatorial Theory, Series A*, 50(2): 285-300, 1989.

[8] Von zur Gathen, Joachim and Jürgen Gerhard. Modern Computer Algebra. Cambridge University Press, 2003. ISBN: 0 521 82646 2.

[9] Von zur Gathen, Joachim and Jürgen Gerhard. "Arithmetic and factorization of polynomials over F_2," In *Proceedings of the 1997 International Symposism on Symbolic and Algebraic Computation* ISSAC '96, Zurich, Switzerland, ed. Lakshman Y.N., ACM Press. Technical report tr-rsfb-96-018, University of Paderborn, Germany, 1996, 43 pages, http://www-math.uni-paderborn.de/ aggathen/Publications/polyfactTR.ps.

[10] Schönhage, Arnold. "Schnelle multiplikation von polynomen über körpern der charakteristik 2," *Acta Informat.* 7 (1976/77), no. 4, 395–398.

[11] Mateer, Todd D. Fast Fourier Transform Algorithms with Applications. PhD Dissertation. Available online at http://cr.yp.to/f2mult.html.

[12] Mateer, Todd D. "New algorithms for decoding systematic Reed-Solomon codes". In Preparation.

[13] Chen, Ning and Zhiyuan Yan, "Cyclotomic FFTs with reduced additive complexities based on a novel common subexpression elimination algorithm," *IEEE Transactions on Signal Processing*, vol.57, no.3, pp.1010-1020, March 2009.

[14] Chen, Ning and Zhiyuan Yan, "Reduced-complexity Reed-Solomon decoders based on cyclotomic FFTs," *IEEE Signal Processing Letters*, Vol. 16, No. 4, pp. 279-282, April 2009.

[15] Menezes, Alfred J. (Editor), Ian F. Blake, Xuhong Gao, Ronald C. Mullin, Scott A. Vanstone and Tomic Yaghoobian. *Applications of Finite Fields*, Kluwer Academic Publishers, Boston/ Dordrecht/Lancaster, 1993.