Novel Polynomial Basis and Its Application to Reed-Solomon Erasure Codes

Yunghsiang S. Han (韓永祥)

National Taiwan University of Science and Technology, Taiwan
（國立台灣科技大學）

Sept. 8, 2014

Joint work with Dr. Sian-Jheng Lin and Dr. Wei-Ho Chung (Academic Sinica, Taiwan)
Accepted for presentation at the 55th Annual Symposium on Foundations of Computer Science (FOCS 2014)

Outline

Monomial Polynomial Basis

- Monomial polynomial basis $\{1, x, x^2, \ldots, x^{n-1}\}$:

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

- Fast Fourier transform (FFT) in monomial basis: polynomial evaluations and polynomial multiplications
- An n-point FFT over complex number domain: $O(n \lg(n))$ additions/multiplications (conjuncture lower bound)
- Primitive nth root of unity is needed for FFT

Monomial Polynomial Basis over Finite Fields

- Monomial polynomial basis over finite fields: polynomial codes (e.g. Reed-Solomon codes)
- Fermat number transform (FNT): over Fermat prime fields $\mathbb{F}_{2^r+1}, r \in \{1, 2, 4, 8, 16\}$–$O(n \lg(n))$ additions/multiplications (Cooley -Tukey algoruthm)
- The major drawback of FNT: more space to store one extra symbol in practical implementation
- FFT over characteristic-2 finite field: $O(n \lg(n))$ finite field operations???

Complexities of FFT Algorithms

Table: Complexities of n-point FFT algorithms over $\mathbb{F}_{2^r}$, where $n = 2^r - 1$

Algorithm	Restrict	Add. complex.	Multi. complex.
[Gao10]	$r = 2^m$	$O(n \lg(n) \lg \lg(n))$	$O(n \lg(n))$
[Cantor89]	$r = 2^m$	$O(n \lg^{\lg(3)}(n))$	$O(n \lg(n))$
[Gao10]		$O(n \lg^2(n))$	$O(n \lg(n))$
[Wang88] [Gathen96]		$O(n \lg^2(n))$	$O(n \lg^2(n))$
[Pollard71]	r is even	$O(n^{3/2})$	$O(n^{3/2})$
[Wu12]		$O(n^2 / \lg^{\lg(8/3)}(n))$	$O(n \lg^{\lg(3/2)}(n))$
[Sarwate78]		$O(n^2)$	$O(n \lg(n))$
Naive approach		$O(n^2)$	$O(n^2)$

Objective of This Work

- A new polynomial basis over characteristic-2 finite fields
- A transform with $O(n \lg(n))$ finite field additions/multiplications, polynomial evaluation
- An application of the new basis: encoding/erasure decoding algorithm for (n, k) Reed-Solomon codes

Finite Field Arithmetic

- $\{\omega_i\}_{i=0}^{2^r-1}$: the elements of $\mathbb{F}_{2^r}$
- V: the r-dimensional vector space spanned by $v_0, v_1, \ldots, v_{r-1} \in \mathbb{F}_{2^r}$
- $\omega_i = i_0 \cdot v_0 + i_1 \cdot v_1 + i_2 \cdot v_2 + \cdots + i_{r-1} \cdot v_{r-1}$,
  $i = i_0 + i_1 \cdot 2 + i_2 \cdot 2^2 + \cdots + i_{r-1} \cdot 2^{r-1}, \forall i_j \in \{0, 1\}, 0 \leq i < 2^r$

Subspace Vanishing Polynomial [Ore33]

- The subspace vanishing polynomial:

$$W_j(x) = \prod_{i=0}^{2^j-1}(x+\omega_i),\ \ deg(W_j(x)) = 2^j$$

Lemma

$W_j(x)$ is an $\mathbb{F}_2$-linearlized polynomial for which

$$W_j(x) = \sum_{i=0}^{j} a_{j,i}x^{2^i},$$

where each $a_{j,i} \in \mathbb{F}_{2^r}$ is a constant. Furthermore,

$$W_j(x+y) = W_j(x) + W_j(y), \forall x, y \in \mathbb{F}_{2^r}.$$

Subspace Vanishing Polynomial – An Example

$j = 2$,

$$W_2(x) = (x+\omega_0)(x+\omega_1)(x+\omega_2)(x+\omega_3)$$

$$\begin{aligned}
W_2(\omega_0) = W_2(\omega_1) = W_2(\omega_2) = W_2(\omega_3) &= \omega_0 \cdot \omega_1 \cdot \omega_2 \cdot \omega_3 = 0;\\
W_2(\omega_4) = W_2(\omega_5) = W_2(\omega_6) = W_2(\omega_7) &= \omega_4 \cdot \omega_5 \cdot \omega_6 \cdot \omega_7;\\
W_2(\omega_8) = W_2(\omega_9) = W_2(\omega_{10}) = W_2(\omega_{11}) &= \omega_8 \cdot \omega_9 \cdot \omega_{10} \cdot \omega_{11}
\end{aligned}$$

Proposed Polynomial Basis

- Proposed polynomial basis over $\mathbb{F}_{2^r}[x]/(x^{2^r} - x)$:
  $\mathbb{X}(x) = (X_0(x), X_1(x), \dots, X_{2^r-1}(x))$
- $X_i(x) = \prod_{j=0}^{r-1} \left(\frac{W_j(x)}{W_j(\omega_{2^j})} \right)^{i_j}$,
  $i = i_0 + i_1 \cdot 2 + \cdots + i_{r-1} \cdot 2^{r-1}, \forall i_j \in \{0, 1\}$
- $\left(\frac{W_j(x)}{W_j(\omega_{2^j})} \right)^{i_j} = 1$ for $i_j = 0$ and $deg(X_i(x)) = i$

Polynomial Expression

Definition

A form of polynomial expression over $\mathbb{F}_{2^r}$ is defined as

$$[D_h](x) = \sum_{i=0}^{h-1} d_i X_i(x),$$

where

$$D_h = (d_0, d_1, \ldots, d_{h-1})$$

is an h-element vector denoting the polynomial coefficients. Clearly, $deg([D_h](x)) \leq h-1$.

Definition of Transform $\Psi_h^l[\bullet]$

- Given D_h, the transform:

$$\hat{D}_h^l = \Psi_h^l[D_h],$$

where

$$\hat{D}_h^l = ([D_h](\omega_0 + \omega_l), [D_h](\omega_1 + \omega_l), \ldots, [D_h](\omega_{h-1} + \omega_l)),$$

and l denotes the amount of shift in the transform

- The inversion:

$$(\Psi_h^l)^{-1}[\hat{D}_h^l] = D_h$$

Recursive Structure

- $[D_h](x)$: a recursive function $[D_h](x) = \Delta_0^0(x)$ with

$$\Delta_i^m(x) = \Delta_{i+1}^m(x) + \frac{W_i(x)}{W_i(\omega_{2^i})}\Delta_{i+1}^{m+2^i}(x), \text{ for } 0 \le i \le \lg(h) - 1;$$

$$\Delta_{\lg(h)}^m(x) = d_m, \text{ for } 0 \le m \le h - 1$$

- $deg(\Delta_i^m(x)) \le h/2^i - 1$

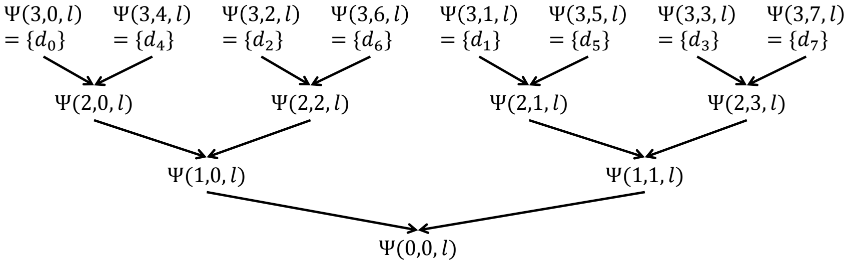
Recursive Structure - An Example

- If $h = 8$, we have

$$
\begin{aligned}
[D_8](x) =& \sum_{i=0}^{7} d_i X_i(x) \\
=& d_0 + d_1 \frac{W_0(x)}{W_0(\omega_1)} + d_2 \frac{W_1(x)}{W_1(\omega_2)} + d_3 \frac{W_0(x)}{W_0(\omega_1)} \frac{W_1(x)}{W_1(\omega_2)} + d_4 \frac{W_2(x)}{W_2(\omega_4)} \\
&+ d_5 \frac{W_0(x)}{W_0(\omega_1)} \frac{W_2(x)}{W_2(\omega_4)} + d_6 \frac{W_1(x)}{W_1(\omega_2)} \frac{W_2(x)}{W_2(\omega_4)} + d_7 \frac{W_0(x)}{W_0(\omega_1)} \frac{W_1(x)}{W_1(\omega_2)} \frac{W_2(x)}{W_2(\omega_4)} \\
=& \left(d_0 + d_4 \frac{W_2(x)}{W_2(\omega_4)} + \frac{W_1(x)}{W_1(\omega_2)} \left(d_2 + d_6 \frac{W_2(x)}{W_2(\omega_4)} \right) \right) \\
&+ \frac{W_0(x)}{W_0(\omega_1)} \left(d_1 + d_5 \frac{W_2(x)}{W_2(\omega_4)} + \frac{W_1(x)}{W_1(\omega_2)} \left(d_3 + d_7 \frac{W_2(x)}{W_2(\omega_4)} \right) \right) \\
=& \left(\Delta_2^0(x) + \frac{W_1(x)}{W_1(\omega_2)} \Delta_2^2(x) \right) + \frac{W_0(x)}{W_0(\omega_1)} \left(\Delta_2^1(x) + \frac{W_1(x)}{W_1(\omega_2)} \Delta_2^3(x) \right) \\
=& \Delta_1^0(x) + \frac{W_0(x)}{W_0(\omega_1)} \Delta_1^1(x) \\
=& \Delta_0^0(x).
\end{aligned}
$$

Proposed Algorithm

- $$\Psi(i, m, l) = \{\Delta_i^m(\omega_c + \omega_l) | c \in \{b \cdot 2^i\}_{b=0}^{h/2^i - 1}\}, \text{ for } 0 \le i \le \lg(h) - 1;$$
  $$\Psi(\lg(h), m, l) = \{d_m\}.$$
  $\Psi(0, 0, l)$: the transform output
- $\Psi(i, m, l) = \Psi(i+1, m, l) \uplus C\Psi(i+1, m+2^i, l)$
- $h = 8$:



Proposed Algorithm - Computational Complexity

- $\Psi(i,m,l) = \Psi_0(i,m,l) + \Psi_1(i,m,l)$

$$\Psi_0(i,m,l) = \{\Delta_i^m(\omega_c + \omega_l) | c \in \{b \cdot 2^{i+1}\}_{b=0}^{h/2^{i+1}-1}\}$$

$$\Psi_1(i,m,l) = \{\Delta_i^m(\omega_c + \omega_l + \omega_{2^i}) | c \in \{b \cdot 2^{i+1}\}_{b=0}^{h/2^{i+1}-1}\}$$

- $\Psi_0(i,m,l)$: $h/2^{i+1}$ elements, 1 addition and 1 multiplication
- $\Psi_1(i,m,l)$: $h/2^{i+1}$ elements, 1 addition
- $A(h)$ and $M(h)$: the number of additions and multiplications
- The recursive formula:

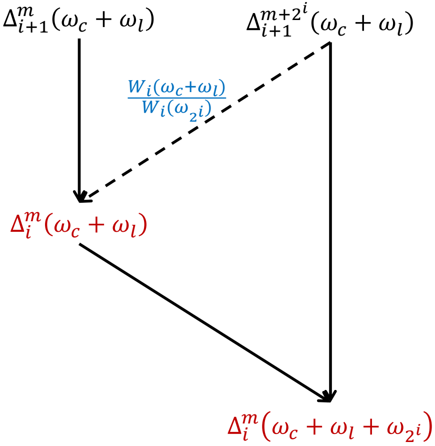
$$A(h) = 2A(h/2) + h; A(1) = 0;$$
$$M(h) = 2M(h/2) + h/2; M(1) = 0.$$

The solution:

$$A(h) = h \lg(h); \;\; M(h) = \frac{h}{2} \lg(h).$$

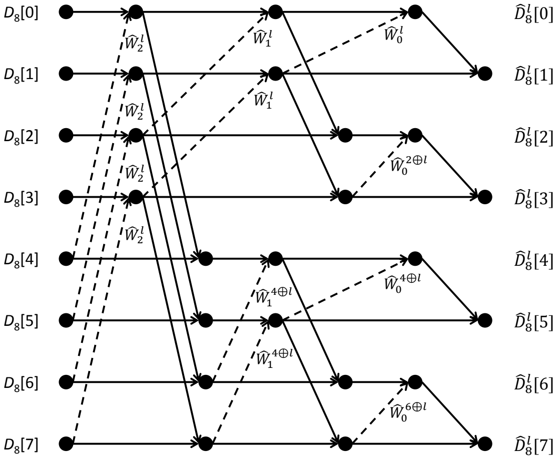
Proposed Algorithm

- Two-point flow chart:



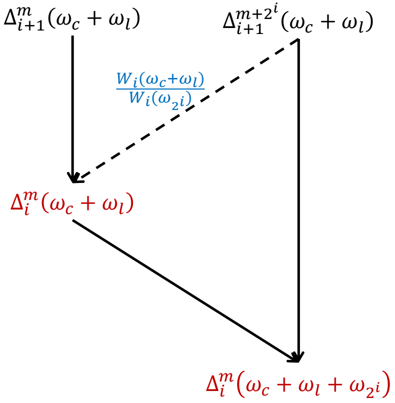
Proposed Algorithm - An Example for Transform at $h = 8$

$$\hat{W}_i^j = \frac{W_i(\omega_j)}{W_i(\omega_{2^i})}$$

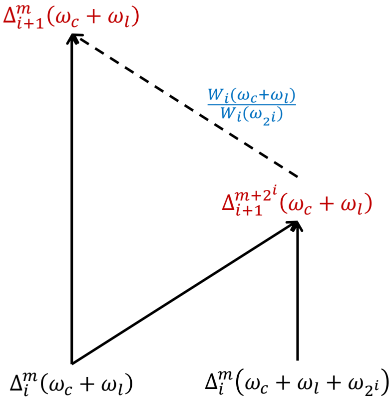


Proposed Algorithm - Inverse Transform

- Flow chart of transform:

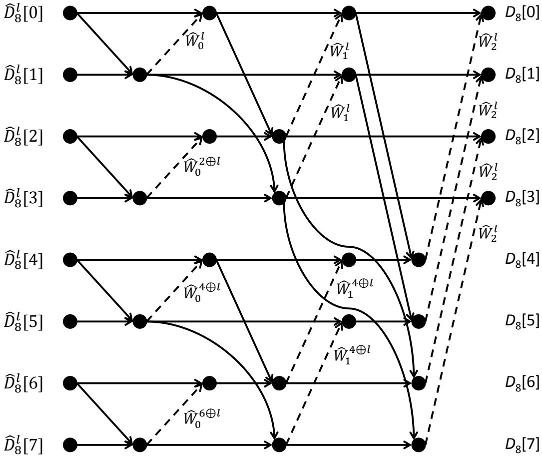


- The inverse algorithm is its backtracking steps:



Proposed Algorithm - An Example for Inverse Transform at $h = 8$

$$\hat{W}_i^j = \frac{W_i(\omega_j)}{W_i(\omega_{2^i})}$$



Formal Derivative

Lemma

Formal derivative of $W_i(x)$ is a constant given by

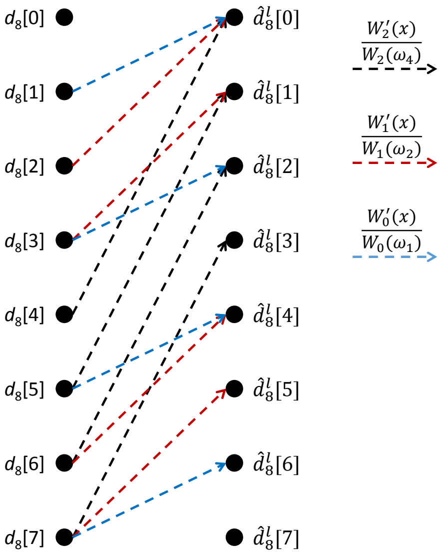
$$W_i'(x) = \prod_{j=1}^{2^i-1} \omega_j.$$

- Formal derivative of $\Delta_i^m(x)$:

$$(\Delta_i^m)'(x) = \underbrace{(\Delta_{i+1}^m)'(x)}_{\text{Compute recursively}} + \underbrace{\frac{W_i'(x)}{W_i(\omega_{2^i})}}_{\text{Constant}} \underbrace{\Delta_{i+1}^{m+2^i}(x)}_{\textit{Known}} + \frac{W_i(x)}{W_i(\omega_{2^i})} \underbrace{(\Delta_{i+1}^{m+2^i})'(x)}_{\text{Compute recursively}};$$

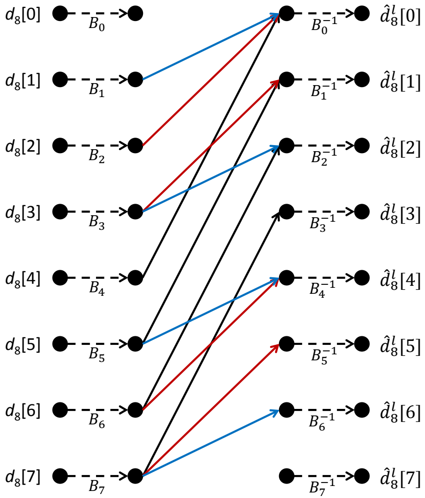
- $T(h) = 2T(h/2) + O(h)$ finite field operations $\rightarrow O(h\lg(h))$

Formal Derivative - An Example at $h = 8$



Formal Derivative - An Example at $h = 8$

Method with lower multiplicative complexity $(O(n \lg(n)) \longrightarrow O(n))$



Reed-Solomon Erasure Codes

- Polynomial evaluation approach
- The vector of message: $M_k = (m_0, m_1, \ldots, m_{k-1})$, $m_i \in \mathbb{F}_{2^r}$
- The codeword: $F_n = (F(\omega_0), F(\omega_1), \ldots, F(\omega_{n-1}))$
- Systematic construction: $F(\omega_i) = m_i$, for $0 \leq i \leq k-1$
- For k a power of two and $n = 2^r$: $k|n$

Encoding Algorithm

Algorithm 1: Encoding algorithm

Data: A k-element message vector M_k

Result: An n-element systematic codeword F_n

1 $\bar{M}_k = (\Psi_k^0)^{-1}[M_k]$;

2 **for** $i = 1$ **to** $(n/k - 1)$ **do**

3 $\quad \bar{F}_i = \Psi_k^{i \cdot k}[\bar{M}_k]$;

4 **end**

5 **return** $F_n = (M_k, \bar{F}_1, \bar{F}_2, \ldots, \bar{F}_{n/k-1})$;

- Line 1: $[\bar{M}_k](x) = F(x)$
- Line 2-4: Encoding in $n/k - 1$ blocks with size k
- A k-element inversion $(\Psi_k^0)^{-1}[\bullet]$ and $(n/k - 1)$ times of k-element transform $\Psi_k^i[\bullet] \longrightarrow O((n/k)k \lg(k)) = O(n \lg(k))$

Erasure Decoding Algorithm

- The transmitted codeword: $F_n = (F(\omega_0), F(\omega_1), \ldots, F(\omega_{n-1}))$
- Received codeword $\bar{F}_n$ with $n-k$ erasures at evaluation points $E = \{\omega_{e_i}\}_{i=0}^{n-k-1}$
- Error locator polynomial:

$$\Pi(x) = \prod_{y \in E} (x + y)$$

- **The goal of decoding:** find $F(j)$ for $\forall j \in E$
- Let $\hat{F}(x) = F(x)\Pi(x)$
- $F(j) = \frac{\hat{F}'(j)}{\Pi'(j)}, \forall j \in E$, where $\hat{F}'(x)$, and $\Pi'(x)$ are the formal derivatives of $\hat{F}(x)$ and $\Pi(x)$
- The decoding procedure: i) compute the coefficients of $\hat{F}(x)$; ii) compute $\hat{F}'(x)$; and iii) compute $F(j)$

Erasure Decoding Algorithm - Computing the Coefficients of $\hat{F}(x)$

- We have
$$\hat{F}(j) = F(j)\Pi(j) = \begin{cases} 0 & \forall j \in E \\ F(j)\Pi(j) & \text{otherwise} \end{cases}$$
- Compute $\bar{\Pi} = \{\Pi(j) | j \in \mathbb{F}_{2^r} \backslash E\}$ by fast Walsh-Hadamard transform [Didier09]
- Obtain $\Phi = (\hat{F}(\omega_0), \hat{F}(\omega_1), \ldots, \hat{F}(\omega_{n-1}))$ with n multiplications
- Compute $\bar{\Phi}_n = (\Psi_n^0)^{-1}[\Phi]$ by the proposed inverse transform and $\hat{F}(x) = [\bar{\Phi}_n](x)$

Erasure Decoding Algorithm - Finding $\hat{F}'(x)$

- $[\bar{\Phi}_n](x)$ is under the proposed polynomial basis
- Compute $[\bar{\Phi}_n]'(x)$ by the proposed formal derivative method and

$$[\bar{\Phi}_n]'(x) = [\bar{\Phi}_n^{\mathrm{d}}](x) = \sum_{i=0}^{n-1} \bar{\phi}_i^{\mathrm{d}} X_i(x)$$

Erasure Decoding Algorithm - Computing $F(j)$

- Compute $\Phi_n^{\mathrm{d}} = \Psi_n^0[\bar{\Phi}_n^{\mathrm{d}}]$ via proposed transform, where Φ_n^{d} consists of $\{\hat{F}'(j) | j \in \mathbb{F}_{2^r}\}$
- Compute $\Pi' = \{\Pi'(j) | j \in E\}$ via fast Walsh-Hadamard transform [Didier09]
- Compute $F(j) = \frac{\hat{F}'(j)}{\Pi'(j)}, \forall j \in E$
- The overall decoding complexity is $O(n \lg(n))$

Complexities of operations in polynomial basis

Table: Complexities of operations in polynomial basis over characteristic-2 finite fields

Operations	Monomial basis	Proposed basis
Addition	$O(h)$	$O(h)$
Multiplication	$O(h \lg(h) \lg\lg(h))$	$O(h \lg(h))$
Polynomial degree	$O(h)$	$O(h)$
Formal derivative	$O(h)$	$O(h \lg(h))$

- In the proposed polynomial basis, $[A_h](x) \times [B_h](x)$ can be computed via

$$(\Psi_{2h}^{l})^{-1}[\Psi_{2h}^{l}[A_{2h}] \star \Psi_{2h}^{l}[B_{2h}]],$$

where A_{2h} (and B_{2h}) is $2h$-point vector by appending zeros to A_h (and B_h), and $\star$ denotes the pairwise multiplication

Simulations

- Tested on Intel core i7-950 CPU in C language
- $n = 2^{16}$, $k/n = 1/2$: about 1.12 seconds in encoding and 3.06 seconds in erasure decoding on average
- Didier's approach: about 52.91 seconds in both encoding and erasure decoding
- 17 times faster than Didier's

Concluding Remarks

- We propose a new polynomial basis over characteristic-2 finite fields
- Two polynomial operations, termed as addition and multiplication, can be completed in $O(h \lg(h))$ finite field operations
- The $O(n \lg(k))$ encoding algorithm and the $O(n \lg(n))$ erasure decoding algorithm of (n, k) Reed-Solomon codes over characteristic-2 finite field are provided based on the proposed transform
- The draft of this work can be found at http://arxiv.org/abs/1404.3458

Thanks

Q&A

科技部 Ministry of Science and Technology