Fast Encoding Algorithms for Reed–Solomon Codes with between Four and Seven Parity Symbols

Leilei Yu, Zhichang Lin, Sian-Jheng Lin, *Member, IEEE*, Yunghsiang S. Han, *Fellow, IEEE*, and Nenghai Yu

Abstract—This paper describes a fast Reed–Solomon encoding algorithm with between four and seven parity symbols. First, we show that the syndrome of Reed–Solomon codes can be computed via the Reed–Muller transform. Based on this result, the fast encoding algorithm is then derived. Analysis shows that the proposed approach asymptotically requires 3 XORs per data bit, representing an improvement over previous algorithms. The simulation demonstrates that the performance of the proposed approach improves with the increase of code length and is superior to other methods. In particular, when the parity number is 5, the proposed approach is about 2 times faster than other cutting-edge methods.

Index Terms—Reed–Solomon code, Reed–Muller transform, coding algorithm.

1 Introduction

Data protection is an essential issue in storage systems. To prevent data loss in the event of failures, the conventional approach is to store multiple replicas on different machines. With the dramatic increase in data volumes over recent years, this method of data replication has become very inefficient and expensive. Another approach for solving the problem of data loss is to use erasure codes, which are derived from the field of coding theory [1]. Precisely, an erasure code is a technique that converts the message into a codeword with a longer length, such that the message can be recovered from a subset of the codeword symbols. Because of their obvious advantages in data protection, erasure codes are now widely used in storage systems [2]. A (k, t) systematic erasure code converts a k-symbol data array into a $(k + t)$-symbol array (codeword) by appending t parity symbols to the data array. These parity symbols can be used to correct erasures on the codeword.

Maximum distance separable (MDS) codes have the important property that data can be reconstructed from any k of the $k + t$ codeword symbols. Reed–Solomon (RS) codes are a well-known type of MDS code. Nowadays, RS codes (and related technologies) are applied in several storage systems, such as redundant arrays of independent disks (RAID) [3], Swift [4], and Ceph [5]. Further, a number of products for large-scale storage systems already support erasure codes. For example, the Google File System (GFS) uses RS codes [6], Azure Cloud Storage System adopts locally repairable codes (LRCs) [7], which can be regarded as a variant of RS codes, and Facebook Hadoop Distributed File System (HDFS) is equipped with Mirrored RAID-5 and RS coding to build the Blob Storage Cluster [8]. This demonstrates that the complexity of erasure codes is of widespread interest in (distributed) storage systems.

As RS codes are constructed over finite fields, finite field arithmetic plays an important role in the performance of the codes. The complexity of finite field arithmetic means that native implementations of RS codes often cannot meet the performance requirements of systems with real-time applications. Thus, many fast algorithms have been proposed in recent decades. For example, [9] used the parallel computing capabilities of hardware to accelerate the arithmetic operation, [10] employed a variant called Cauchy Reed–Solomon coding to improve the performance of RS coding, and Trifonov [3] proposed a low-complexity implementation of fast Fourier transform (FFT)-based RS codes. Furthermore, the density of generator matrices for MDS codes has been explored [11], leading to the result that the encoding requires at least $t - t/k$ XORs per data bit if each parity symbol is computed independently without sharing any intermediate results. However, it is possible to further reduce the computational complexity by sharing the intermediate results. Such algorithms are known as scheduled algorithms [12]. The optimal scheduled algorithm for a (k, t) RS code remains an open problem. In [13], [14], Lin et al. proposed RS encoding algorithms that attain the optimal complexity bound of 2 XORs per data bit for $t = 2, 3$. However, [13], [14] do not consider algorithms for $t \geq 4$, limiting the applicability of the algorithms. For example, Facebook's HDFS uses $(10, 4)$ RS codes [15], the file system

- *L. Yu, Z. Lin, S.-J. Lin, and N. Yu are with the School of Information Science and Technology, University of Science & Technology of China, Heifei 230026, China.*
E-mail: yuleilei@mail.ustc.edu.cn, zc592782@mail.ustc.edu.cn, sjlin@ustc.edu.cn, ynh@ustc.edu.cn.
- *Y. S. Han is with the School of Electrical Engineering & Intelligentization, Dongguan University of Technology, Dongguan, China.*
E-mail: yunghsiangh@gmail.com.
- *Corresponding author: Sian-Jheng Lin.*
- *This work was supported in part by the Hundred Talents Program of Chinese Academy of Sciences, Natural Science Foundation of Anhui Province (No. BJ2100330001), the National Natural Science Foundation of China (No. 61671007), and Start Fund of Dongguan University of Technology (KCYXM2017025).*

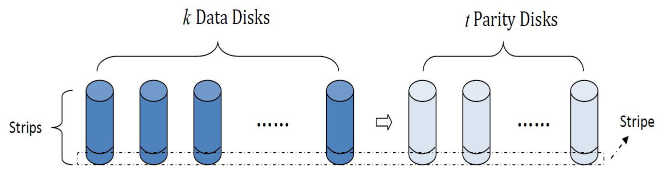

Fig. 1: A typical storage system with erasure coding

Btrfs supports up to six parity devices in RAID [16], and GFS II encodes cold data using $(9,6)$ RS codes [6].

In this paper, we focus on the time complexity of RS codes. To date, codes that tolerate at least four erasures have been proposed, such as generally circulant Cauchy codes [17], Rabin-like codes [18], Trifonov's algorithm [3], and an FFT algorithm for finite fields [19]. We present a novel class of RS encoding algorithms based on Reed–Muller transforms. In particular, we show that a Vandermonde matrix can be decomposed into the product of a sparse matrix and a Reed–Muller matrix. Based on this result, the syndrome computations for RS codes can be performed efficiently via the Reed–Muller transform. In this formulation, the proposed encoding algorithm requires 3 XORs per data bit for $t = 4, 5, 6, 7$ parity symbols.

Currently, the RS code libraries used in storage systems are Intel's ISA-L [20] and Jerasure [21], and these operate on CPUs supporting the single instruction–multiple data (SIMD) instruction set. We implement the proposed approach over $\mathbb{F}_{2^8}$ using Intel's Streaming SIMD Extensions (SSE). The data distribution of the storage system is shown in Fig. 1. We borrow the terminology from [22]. Each disk is split into multiple fixed-size strips, and a total of $k + t$ strips from each disk are called a stripe. We investigate the proposed algorithm for $t = 4, 5, 6$, and 7 with a stripe unit size of $\lambda = 4096$ bytes. Further, the impact of coding efficiency on performance in real applications is briefly discussed. The main contributions of this paper can be summarized as follows.

1) An improved syndrome computation for $(k, t \leq 7)$ RS codes is proposed, where k is the number of data symbols and t is the number of parity symbols.
2) Fast encoding/decoding algorithms for (k, t) RS codes are presented for $t = 4, 5, 6, 7$.
3) The SIMD implementation of the proposed (k, t) RS coding algorithm is given.

The remainder of this paper is organized as follows. In Section 2, the construction of RS codes briefly introduced, and the binary extended fields and the Reed-Muller transform are defined. Section 3 presents the syndrome calculation algorithm for $t \leq 7$ and the RS encoding/decoding algorithms for $t = 4, 5, 6$, and 7. The complexity of the proposed algorithms is analyzed in Section 4. In Section 5, simulations and evaluations are presented. Section 6 concludes this paper.

2 Preliminaries

RS codes are a group of erasure codes with MDS property, where the k original data symbols are encoded into $n > k$ symbols. The MDS property guarantees that all original data can be reconstructed from any k symbols of n symbols. Let

$$\mathrm{Vand}_i(\omega_0, \omega_1, \ldots, \omega_{j-1}) := \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \omega_0 & \omega_1 & \cdots & \omega_{j-1} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_0^{i-1} & \omega_1^{i-1} & \cdots & \omega_{j-1}^{i-1} \end{bmatrix}$$

denote an $i \times j$ Vandermonde matrix. The codeword $\mathbf{c}$ of the (k, t) RS code over $\mathbb{F}_q$ meets

$$\mathbf{0} = \mathbf{H}\mathbf{c}, \tag{1}$$

where $\mathbf{H} = \mathrm{Vand}_t(\omega_0, \omega_1, \ldots, \omega_{n-1})$ is the parity-check matrix, and $n = k + t < q$. The coding rate is given by k/n. The code distance is $t + 1$, as any t columns of $\mathbf{H}$ forms a non-singular matrix. This leads that the RS code can correct up to t erasures. Note that the codeword $\mathbf{c}$ can also be generated by a generator matrix $\mathbf{G}$ of the code as $\mathbf{c} = \mathbf{G}^{\mathrm{T}}\mathbf{m}$, where $\mathbf{m}$ is the column vector of the k data symbols and $\mathbf{G} = \mathrm{Vand}_k(\omega'_0, \omega'_1, \ldots, \omega'_{n-1})$.

2.1 Systematic Reed-Solomon codes

The codeword is denoted as $\mathbf{c}^{\mathrm{T}} = [\mathbf{p}^{\mathrm{T}} \ \mathbf{m}^{\mathrm{T}}]$, where $\mathbf{p}$ is the column vector of t parity symbols and $\mathbf{m}$ is the column vector of k data symbols. From (1), we have

$$\mathbf{0} = \underbrace{\left[\begin{array}{c|c} \mathbf{H}_{en} & \widehat{\mathbf{H}} \end{array}\right]}_{\mathbf{H}} \begin{bmatrix} \mathbf{p} \\ \mathbf{m} \end{bmatrix}, \tag{2}$$

where the square matrix $\mathbf{H}_{en}$ consists of the first t column of $\mathbf{H}$ and the remaining columns of $\mathbf{H}$ constitute $\widehat{\mathbf{H}}$. As a result, the parity symbols of the systematic RS code can be obtained as

$$\mathbf{p} = \mathbf{H}_{en}^{-1} \cdot \widehat{\mathbf{H}}\mathbf{m}. \tag{3}$$

Upon receiving a codeword with t erasures, the decoder first calculates the syndrome $\mathbf{s}$. We write

$$\mathbf{s} = \mathbf{H} \cdot \mathbf{c}', \tag{4}$$

where $\mathbf{c}'$ denotes the received codeword with t erasures, and the erased symbols have values of 0. Let $\mathbf{e}$ denote the vector of t erased symbols; $\mathbf{H}_{de}$ denote the $t \times t$ matrix consisting of the columns corresponding to the positions of the erasures in $\mathbf{H}$. From (1) and (4), then

$$\mathbf{e} = \mathbf{H}_{de}^{-1} \cdot \mathbf{s}. \tag{5}$$

For a block code, the encoding process can be viewed as a special case of the decoding process. Let $\mathbf{c}'$ be the codeword whose erasures are on the parity part. By applying (4) and (5) to it, one can calculate the erased symbols which are the parity symbols. Note that, an efficient syndrome computation can improve both the encoding and decoding throughput.

2.2 Finite fields

The finite field with size $q = 2^m$ is defined as

$$\mathbb{F}_q := \mathbb{F}_2[x]/\pi(x),$$

with the primitive polynomial

$$\pi(x) = \sum_{i=0}^{m} \pi_i x^i$$

for each $\pi_i \in \mathbb{F}_2$. Let α denote a primitive element of $\mathbb{F}_q$ such that $\pi(\alpha) = 0$. Let $\{v_i \in \mathbb{F}_q\}_{i=0}^{m-1}$ denote a basis of $\mathbb{F}_q$. The standard basis is defined as

$$v_i = \alpha^i, \qquad i = 0, \dots, m-1. \tag{6}$$

The 2^m elements of $\mathbb{F}_q$ are denoted by $\omega_0, \omega_1, \dots, \omega_{2^m-1}$, where

$$\omega_i = i_0 \cdot v_0 + i_1 \cdot v_1 + \cdots + i_{m-1} \cdot v_{m-1}, \forall i_j \in \mathbb{F}_2, \tag{7}$$

with

$$i = i_{m-1} \cdot 2^{m-1} + \cdots + i_1 \cdot 2 + i_0. \tag{8}$$

Conventionally, we denote the multiplication inverse of ω_i as $\frac{1}{\omega_i}$ or ω_i^{-1}. From (7), when N is a power of two,

$$\omega_{i+N/2} = \omega_i + v_{r-1} \tag{9}$$

for $i < N/2 = 2^{r-1}$.

2.3 Reed-Muller transform

Given the input data $\mathbf{x} = [x_0\ x_1\ \cdots\ x_{N-1}]^{\mathrm{T}}$, the output $\mathbf{y} = [y_0\ y_1\ \cdots\ y_{N-1}]^{\mathrm{T}}$ of the Reed–Muller transform is denoted by

$$\mathbf{y} = \mathbf{B}_N \cdot \mathbf{x}, \tag{10}$$

where $N = 2^r$ a power of two. Furthermore, $\mathbf{B}_{2^r}$, $r \geq 1$, is defined as

$$\mathbf{B}_{2^{r+1}} = \begin{bmatrix} \mathbf{B}_{2^r} & \mathbf{B}_{2^r} \\ 0 & \mathbf{B}_{2^r} \end{bmatrix}, \quad \mathbf{B}_1 = [1]. \tag{11}$$

Let $\mathbf{I}_N$ denote the $N \times N$ identity matrix. The following is a property of $\mathbf{B}_N$.

Theorem 1. $\mathbf{B}_N^2 = \mathbf{I}_N$ *over* $\mathbb{F}_2$ *for any* $N \in \{2^r | r \in \mathbb{N}\}$.

Proof. The proof uses mathematical induction. For the basis case $\ell = 0$, we have $\mathbf{B}_1 = [1] = \mathbf{I}_1$, and so $\mathbf{B}_1^2 = \mathbf{I}_1^2 = \mathbf{I}_1$. Assume that the theorem holds for $\mathbf{B}_{2^r}$. Then,

$$\mathbf{B}_{2^{r+1}}^2 = \begin{bmatrix} \mathbf{B}_{2^r} & \mathbf{B}_{2^r} \\ 0 & \mathbf{B}_{2^r} \end{bmatrix}^2 = \begin{bmatrix} \mathbf{B}_{2^r}^2 & 0 \\ 0 & \mathbf{B}_{2^r}^2 \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{2^r} & 0 \\ 0 & \mathbf{I}_{2^r} \end{bmatrix}.$$

This completes the proof. □

The Reed–Muller transform is a recursive algorithm for calculating (10). Fig. 2 shows an example for $N = 8$. Let

$$\mathbf{x}^{(i)} := \left[x_{0+iN/2}\ \cdots\ x_{L/2-1+iN/2}\right]^{\mathrm{T}},$$

$$\mathbf{y}^{(i)} := \left[y_{0+iN/2}\ \cdots\ y_{L/2-1+iN/2}\right]^{\mathrm{T}}$$

for $i = 0, 1$. The following equalities are satisfied.

Lemma 1. $y_0 = \sum_{i=0}^{N-1} x_i$.

Proof. From (11), the first row of the matrix $\mathbf{B}_N$ is filled with ones. Hence, y_0 is the summation of $\{x_i\}_{i=0,1,\cdots,N-1}$. □

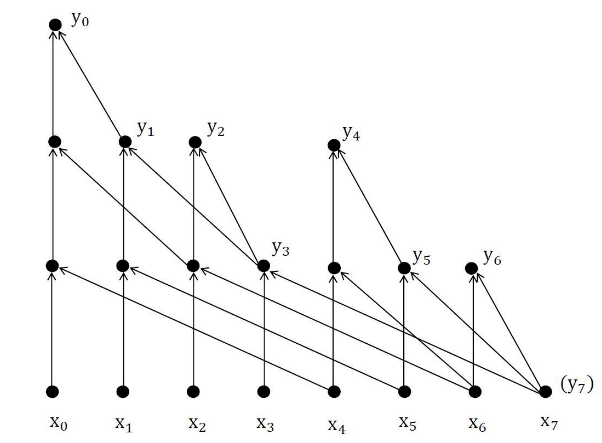

Fig. 2: Transform graph of the input array with $N = 8$

TABLE 1: Definitions of notations in the proposed algorithm

Symbol	Definition
	Common
k	Number of data symbols in a codeword
t	Number of parity symbols in a codeword
n	$n = k + t$ is the codeword length
$\mathbf{H}$	Parity-check matrix (13)
$\mathbf{c}$	$\mathbf{c} = [c_0\ \cdots\ c_{n-1}]$ denotes a codeword of size n
$\mathbf{c}'$	$\mathbf{c}' = [c'_0\ \cdots\ c'_{n-1}]$ denotes a codeword with t erasures
$\mathbf{s}$	$\mathbf{s} = [s_0\ \cdots\ s_{t-1}]$ denotes the syndrome used in coding
	Syndrome computation
N	N is a power of two and $N/2 < n-1 \leq N$
r	r is defined as $r = \lg N$
$\mathbf{x}$	$\mathbf{x} = [x_0\ \cdots x_{N-1}]$ constructed for fast computation
$\mathbf{y}$	$\mathbf{y} = [y_0\ \cdots y_{N-1}]$ is the value of $\mathbf{x}$ after RM transform
$\mathbf{B}_N$	A matrix consists of 0 and 1, defined as (11)
$P_N^i(\mathbf{x})$	Output of fast computation (15)
	Encoding/Decoding
$\mathbf{m}$	$\mathbf{m} = [m_0\ \cdots\ m_{k-1}]$ denotes a data vector of size k
$\mathbf{p}$	$\mathbf{p} = [p_0\ \cdots\ p_{t-1}]$ denotes a parity vector of size t
$\mathbf{U}$	The set of erased positions, defined as $\mathbf{U} = \{u_i\}_{i=0}^{t-1}$
$\mathbf{H}_{en}$	A $t \times t$ matrix consists of the first t column of $\mathbf{H}$
$\widehat{\mathbf{H}}$	A $t \times k$ matrix used for encoding, $\mathbf{H} = [\ \mathbf{H}_{en}\ \vert\ \widehat{\mathbf{H}}\ ]$
$\mathbf{H}_{de}$	The matrix consisting of the i-th column of $\mathbf{H}$, for $i \in \mathbf{U}$

Lemma 2. $y_{N/2} = \sum_{i=0}^{N/2-1} x_{N/2+i}$.

Proof. From (10) and (11), $\mathbf{y}^{(1)} = \mathbf{B}_{N/2} \cdot \mathbf{x}^{(1)}$ can be obtained. By Lemma 1, we have $y_{N/2} = \sum_{i=N/2}^{N-1} x_i = \sum_{i=0}^{N/2-1} x_{N/2+i}$. □

Lemma 3. $\mathbf{y}^{(0)} = \mathbf{B}_{N/2} \cdot \mathbf{x}'$, *where* $\mathbf{x}' = \mathbf{x}^{(0)} + \mathbf{x}^{(1)} = [x_0 + x_{N/2}\ \cdots\ x_{N/2-1} + x_{N-1}]^{\mathrm{T}}$.

Proof. From (10) and (11), we have

$$\begin{bmatrix} \mathbf{y}^{(0)} \\ \mathbf{y}^{(1)} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{N/2} & \mathbf{B}_{N/2} \\ 0 & \mathbf{B}_{N/2} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}^{(0)} \\ \mathbf{x}^{(1)} \end{bmatrix}. \tag{12}$$

Hence, $\mathbf{y}^{(0)} = \mathbf{B}_{N/2}(\mathbf{x}^{(0)} + \mathbf{x}^{(1)})$. □

3 Proposed method

The RS codes considered here are shortened versions of doubly-extended RS codes. That is, the parity-check matrix of the (k,t) RS code is defined by

$$\mathbf{H} = \left[\begin{array}{c|cccc} 0 & 1 & 1 & \cdots & 1 \\ 0 & \omega_0 & \omega_1 & \cdots & \omega_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \omega_0^{t-2} & \omega_1^{t-2} & \cdots & \omega_{n-2}^{t-2} \\ 1 & \omega_0^{t-1} & \omega_1^{t-1} & \cdots & \omega_{n-2}^{t-1} \end{array}\right], \tag{13}$$

which appends an extra column $[0\ 0\ \ldots\ 0\ 1]^{\mathrm{T}}$ to the Vandermonde matrix, and $n = k + t$. Table 1 lists the notation used in RS coding, where lg denotes the binary logarithm $\log_2$.

3.1 Syndrome computation

The proposed syndrome computation algorithm utilizes the algebraic structure of the parity-check matrix $\mathbf{H}$. Equations (4) and (13) give $\mathbf{s} = [s_0\ s_1\ \cdots\ s_{t-1}]^{\mathrm{T}}$, where

$$s_i = \begin{cases} \sum_{j=0}^{n-2} c'_{j+1}\omega_j^i & i = 0,1,\cdots,t-2, \\ (\sum_{j=0}^{n-2} c'_{j+1}\omega_j^i) + c'_0 & i = t-1. \end{cases} \tag{14}$$

The input data can be denoted as $\mathbf{x} = [x_0\ x_1\ \cdots\ x_{N-1}]^{\mathrm{T}}$, where $N = 2^r \geq n-1$ is a power of two. Let

$$P_N^i := P_N^i(x_0,\ldots,x_{N-1}) := \sum_{j=0}^{N-1} x_j\omega_j^i. \tag{15}$$

From (14) and (15), it can be seen that

$$s_i = \begin{cases} P_N^i(c'_1,\ldots,c'_{n-1},0,\ldots,0), & i=0,1,\cdots,t-2, \\ P_N^i(c'_1,\ldots,c'_{n-1},0,\ldots,0) + c'_0, & i=t-1, \end{cases}$$

where

$$N/2 < n-1 \leq N. \tag{16}$$

The following gives a recursive expression for calculating P_N^i. Obviously, P_N^i can be expanded as

$$P_N^i(x_0,\cdots,x_{N-1}) = \sum_{j=0}^{N/2-1} (x_j\omega_j^i + x_{j+N/2}\omega_{j+N/2}^i). \tag{17}$$

Furthermore, let

$$x'_j := x_j + x_{j+N/2}, \quad j = 0,1,\cdots,N/2-1. \tag{18}$$

Next, we show how to apply $\mathbf{y} = \mathbf{B}_N \cdot \mathbf{x}$ in (10) to calculate P_N^i rapidly. The following reformulates (17) for $i = 0$, $i = 2^\ell, \ell \in \mathbb{N}$, and $i \in \{\ell_1 + \ell_2 | \ell_1 = 2^a, \ell_2 = 2^b, a < b, a, b \in \mathbb{N}\}$.

1) Case $i = 0$: From (15) and Lemma 1,

$$P_N^0(x_0,\cdots,x_{N-1}) = \sum_{j=0}^{N-1} x_j = y_0, \tag{19}$$

where y_i is the value converted by the Reed–Muller transform.

2) Case $i = 2^\ell, \ell \in \mathbb{N}$: From (9), we have

$$\omega_{j+N/2}^i = (\omega_j + \upsilon_{r-1})^i = \omega_j^i + \upsilon_{r-1}^i. \tag{20}$$

Then, from (17) and Lemma 2,

$$\begin{aligned} &P_N^i(x_0,\cdots,x_{N-1}) \\ &= \sum_{j=0}^{N/2-1} (x_j\omega_j^i + x_{j+N/2}\omega_{j+N/2}^i) \\ &= \sum_{j=0}^{N/2-1} [(x_j + x_{j+N/2})\omega_j^i + x_{j+N/2}\upsilon_{r-1}^i] \\ &= \sum_{j=0}^{N/2-1} x'_j\omega_j^i + \upsilon_{r-1}^i \sum_{j=0}^{N/2-1} x_{j+N/2} \\ &= P_{N/2}^i(x'_0,\cdots,x'_{N/2-1}) + \upsilon_{r-1}^i \cdot y_{N/2}, \end{aligned} \tag{21}$$

where $y_{N/2}$ is defined in Lemma 2 and x'_i is defined in (18). Notably, by applying the decomposition recursively on (21), we have

$$P_N^i(x_0,\cdots,x_{N-1}) = \sum_{j=0}^{r-1} \upsilon_j^i \cdot y_{2^j}. \tag{22}$$

Note that y_{2^j} can be obtained using the Reed–Muller transform given in (10).

3) Case $i \in \{\ell_1 + \ell_2 | \ell_1 = 2^a, \ell_2 = 2^b, a < b, a, b \in \mathbb{N}\}$: Similar to (20), we have

$$\begin{aligned} \omega_{j+N/2}^i &= (\omega_j + \upsilon_{r-1})^{\ell_1+\ell_2} \\ &= (\omega_j^{\ell_1} + \upsilon_{r-1}^{\ell_1}) \cdot (\omega_j^{\ell_2} + \upsilon_{r-1}^{\ell_2}) \\ &= \omega_j^i + \omega_j^{\ell_1} \cdot \upsilon_{r-1}^{\ell_2} + \omega_j^{\ell_2} \cdot \upsilon_{r-1}^{\ell_1} + \upsilon_{r-1}^i. \end{aligned} \tag{23}$$

Then, we have

$$\begin{aligned} &P_N^i(x_0,\cdots,x_{N-1}) \\ &= \sum_{j=0}^{N/2-1} (x_j\omega_j^i + x_{j+N/2}\omega_{j+N/2}^i) \\ &= \sum_{j=0}^{N/2-1} [(x_j + x_{j+N/2})\omega_j^i + x_{j+N/2}\upsilon_{r-1}^i \\ &\quad + x_{j+N/2}\omega_j^{\ell_1}\upsilon_{r-1}^{\ell_2} + x_{j+N/2}\omega_j^{\ell_2}\upsilon_{r-1}^{\ell_1}] \\ &= P_{N/2}^i(x'_0,\cdots,x'_{N/2-1}) + \upsilon_{r-1}^i \cdot y_{N/2} \\ &\quad + \upsilon_{r-1}^{\ell_2} \cdot P_{N/2}^{\ell_1}(x_{N/2},\cdots,x_{N-1}) \\ &\quad + \upsilon_{r-1}^{\ell_1} \cdot P_{N/2}^{\ell_2}(x_{N/2},\cdots,x_{N-1}). \end{aligned} \tag{24}$$

Substituting (22) into (24), we have

$$\begin{aligned} &P_N^i(x_0,\cdots,x_{N-1}) \\ &= P_{N/2}^i(x'_0,\cdots,x'_{N/2-1}) + \upsilon_{r-1}^i \cdot y_{N/2} \\ &\quad + \sum_{j=0}^{r-2} (\upsilon_{r-1}^{\ell_2}\upsilon_j^{\ell_1} + \upsilon_{r-1}^{\ell_1}\upsilon_j^{\ell_2}) \cdot y_{N/2+2^j} \\ &= P_{N/2}^i(x'_0,\cdots,x'_{N/2-1}) \\ &\quad + \upsilon_{r-1}^i[y_{N/2} + \sum_{j=0}^{r-2} (\upsilon_{r-1-j}^{-\ell_1} + \upsilon_{r-1-j}^{-\ell_2}) \cdot y_{N/2+2^j}]. \end{aligned} \tag{25}$$

Similarly, by applying the decomposition recursively on (25), we have

$$\begin{aligned}&P_N^i(x_0,\cdots,x_{N-1})\\ =\;&\sum_{j=0}^{r-1}\upsilon_j^i\cdot y_{2^j}+\sum_{j=1}^{r-1}\sum_{\xi=0}^{j-1}\upsilon_j^i(\upsilon_{j-\xi}^{-\ell_1}+\upsilon_{j-\xi}^{-\ell_2})\cdot y_{2^j+2^\xi}.\end{aligned} \tag{26}$$

In summary, the first case covers $i=0$, the second case covers $i\in\{1,2,4\}$, and the third case covers $i\in\{3,5,6\}$. Hence, these three cases cover $i\leq 6$. Thus, the algorithm supports syndrome sizes of $t\leq 7$. Moreover, it can be observed that the algorithm only uses a portion of $\mathbf{y}$ in (10) to compute the syndrome.

3.2 Coding for Reed–Solomon codes

In this section, we present the encoding/decoding algorithm for RS codes with parity numbers of less than or equal to 7 based on the above fast syndrome computation.

3.2.1 *Encoding*

From the definition, $\mathbf{p}=[p_0\ p_1\ \cdots\ p_{t-1}]^{\mathrm{T}}$ is the vector of t parity symbols and $\mathbf{m}=[m_0\ m_1\ \cdots\ m_{k-1}]^{\mathrm{T}}$ is the vector of k data symbols. From (3) and (13), we have

$$\mathbf{p}=\mathbf{H}_{en}^{-1}\cdot\widehat{\mathbf{H}}\mathbf{m}, \tag{27}$$

where

$$\mathbf{H}_{en}=\begin{bmatrix}0&1&\cdots&1\\0&\omega_0&\cdots&\omega_{t-2}\\\vdots&\vdots&\ddots&\vdots\\1&\omega_0^{t-1}&\cdots&\omega_{t-2}^{t-1}\end{bmatrix}, \tag{28}$$

$$\widehat{\mathbf{H}}=\mathrm{Vand}_t(\omega_{t-1},\cdots,\omega_{n-2}). \tag{29}$$

Thus, the encoding consists of two processes. The first calculates the syndrome $\mathbf{s}=\widehat{\mathbf{H}}\mathbf{m}$, and the second calculates $\mathbf{p}=\mathbf{H}_{en}^{-1}\cdot\mathbf{s}$.

In the first process, the syndrome is denoted as $\mathbf{s}=[s_0\ \cdots\ s_{t-1}]^{\mathrm{T}}$, and $N\geq n-1$ is a power of two. The N-symbol $\mathbf{x}$ is then defined as

$$\mathbf{x}=[\underbrace{0\ \cdots\ 0}_{t-1}\ \mathbf{m}\ \underbrace{0\ \cdots\ 0}_{N-n+1}]^{\mathrm{T}}, \tag{30}$$

which consists of three parts. Precisely, the first part has $t-1$ zeros, the second part contains $n-t$ data symbols, and the last part has $N-n+1$ zeros. Then, we have

$$s_i=\sum_{j=t-1}^{n-2}x_j\omega_j^i=\sum_{j=0}^{N-1}x_j\omega_j^i=P_N^i(\mathbf{x})$$

for $i=0,1,\cdots,t-1$.

In the second process, we calculate

$$\mathbf{p}=\mathbf{H}_{en}^{-1}\cdot\mathbf{s}=\mathbf{H}_{en}^{-1}\cdot[P_N^0\ P_N^1\ \cdots\ P_N^{t-1}]^{\mathrm{T}}, \tag{31}$$

where P_N^i is the abbreviation of $P_N^i(\mathbf{x})$. Note that $\mathbf{H}_{en}^{-1}$ exists, because it is an extended Vandermonde matrix (see (28)). In (31), the matrix is given by

$$\mathbf{H}_{en}^{-1}=\begin{bmatrix}0&h_{0,1}&\cdots&h_{0,t-2}&1\\1&h_{1,1}&\cdots&h_{1,t-2}&0\\0&h_{2,1}&\cdots&h_{2,t-2}&0\\\vdots&\vdots&\ddots&\vdots&\vdots\\0&h_{t-1,1}&\cdots&h_{t-1,t-2}&0\end{bmatrix}. \tag{32}$$

From (31) and (32), the parity symbols can be obtained via

$$p_i=\sum_{j=1}^{t-2}h_{i,j}\cdot P_N^j+\begin{cases}P_N^{t-1}, & i=0,\\ P_N^0, & i=1,\\ 0, & i=2,3,\cdots,t-1.\end{cases} \tag{33}$$

In summary, the proposed (k,t) RS encoding algorithm of length $n=k+t$ has the following three steps.

(1) Given the data symbols $\mathbf{m}$ and the parity number t, the matrix $\mathbf{H}_{en}^{-1}$ is determined and $\mathbf{x}$ is obtained from (30). Let $N=2^r$, where $r=\lceil\log_2(n-1)\rceil$.
(2) Given $\mathbf{x}$ and t, calculate $\{P_N^i(\mathbf{x})\}_{i=0,1,\cdots,t-1}$ from (19), (22), (26).
(3) The parity symbols $\mathbf{p}$ are calculated from (33).

3.2.2 *Decoding*

This subsection describes the decoding method for codes with the parity-check matrix given in (13). Referring to the above symbols, assume that $\mathbf{c}'$ denotes the codeword with t erasures located in $\mathbf{U}=\{u_i\}_{i=0}^{t-1}$, and $0\leq u_0<u_1<\cdots<u_{t-1}\leq n-1$. From (4), (5) and (13), the final result of decoding is

$$[c_{u_0}\ \cdots\ c_{u_{t-1}}]^{\mathrm{T}}=\mathbf{H}_{de}^{-1}\cdot\mathbf{s}, \tag{34}$$

where $\mathbf{s}=\mathbf{H}\mathbf{c}'$ and $\mathbf{H}_{de}$ is the submatrix consisting of the u_i-th column of $\mathbf{H}$ for $u_i\in\mathbf{U}$. Here, we construct the input data $\mathbf{x}$ as

$$\mathbf{x}=[\underbrace{c'_1,\ldots,c'_{n-1}}_{n-1},\underbrace{0,\ldots,0}_{N-n+1}]. \tag{35}$$

From (14), the fast syndrome computation is then used to improve the decoding efficiency. The following results are obtained from Section 3.1.

$$s_i=\begin{cases}P_N^i, & i=0,1,\cdots,t-2,\\ P_N^i+c'_0 & i=t-1.\end{cases} \tag{36}$$

Finally, the erased value can be calculated by substituting (36) into (34).

3.3 Instance

We consider a $(10,4)$ RS code over $\mathbb{F}_{2^8}$, in which the primitive root is α. From Section 3.2.1, the encoding process consists of three steps. We have $N=16=2^4$ and the input sequence $\mathbf{x}=[0\ 0\ 0\ \mathbf{m}\ 0\ 0\ 0]^{\mathrm{T}}$ from (16) and (30). The 4×4 constant matrices used for encoding are given by

$$\mathbf{H}_{en}=\begin{bmatrix}0&1&1&1\\0&0&1&\alpha\\0&0&1&\alpha^2\\1&0&1&\alpha^3\end{bmatrix},\mathbf{H}_{en}^{-1}=\begin{bmatrix}0&\alpha&1+\alpha&1\\1&\frac{1+\alpha}{\alpha}&\frac{1}{\alpha}&0\\0&\frac{\alpha}{1+\alpha}&\frac{1}{1+\alpha}&0\\0&\frac{1}{\alpha+\alpha^2}&\frac{1}{\alpha+\alpha^2}&0\end{bmatrix}.$$

We first calculate $\{P_{16}^i\}_{i=0}^3$ from the input $\mathbf{x}$. Specifically, the y_i required by the algorithm can be obtained from Fig. 3, and the results are from (19), (22), (26).

$$\begin{aligned}&P_{16}^0=y_0,\\&P_{16}^1=y_1+\alpha\cdot y_2+\alpha^2\cdot y_4+\alpha^3\cdot y_8,\\&P_{16}^2=y_1+\alpha^2\cdot y_2+\alpha^4\cdot y_4+\alpha^6\cdot y_8,\\&P_{16}^3=y_1+\alpha^3\cdot y_2+\alpha^6\cdot y_4+\alpha^9\cdot y_8+\sum_{0\leqslant\xi<j\leqslant 3}\varphi_{j,\xi}y_{2^j+2^\xi},\end{aligned}$$

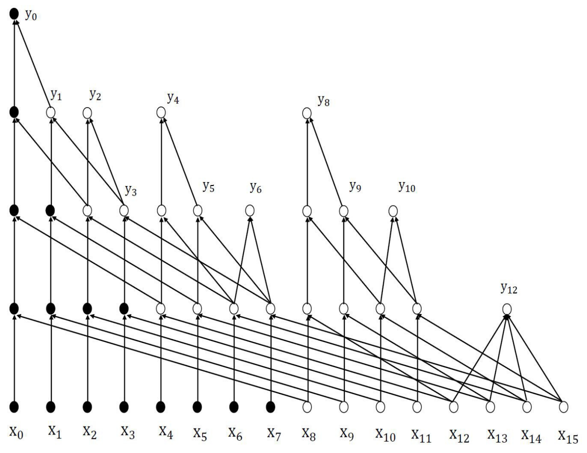

Fig. 3: The RM transform for $N = 16$

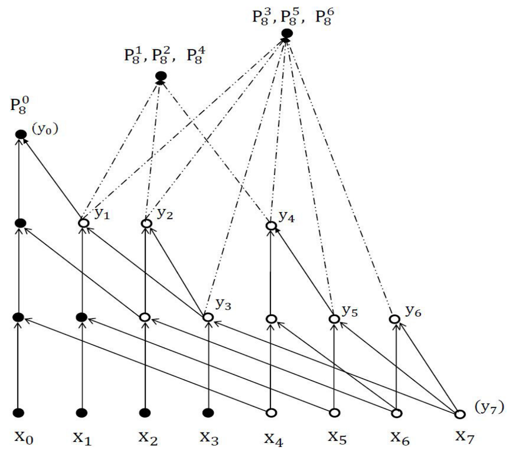

Fig. 4: The signal-flow diagram of (15) with $N = 8$

TABLE 2: Definition of notations for complexity analysis

Symbol	Definition
$\Theta(N)$	The additive complexity of calculating all required y_i from $\mathbf{x}$
$\Theta'(N)$	The additive complexity of calculating required y_i from the last half of $\mathbf{x}$
$\mathrm{A}_i(N)$	The additive complexity of coding with $i+3$ parities, for $i = 1, 2, 3, 4$
$\mathrm{M}_i(N)$	The multiplicative complexity of coding with $i+3$ parities, for $i = 1, 2, 3, 4$

where $\varphi_{j,\xi} = \alpha^{\xi+2j} + \alpha^{2\xi+j}$. Then, the constant matrix $\mathbf{H}_{en}^{-1}$ is multiplied by $[P_{16}^0\ P_{16}^1\ P_{16}^2\ P_{16}^3]^{\mathrm{T}}$ to obtain the following parity symbols:

$$\begin{aligned} p_0 &= \alpha P_{16}^1 + (1+\alpha)P_{16}^2 + P_{16}^3, \quad & p_2 &= \frac{\alpha P_{16}^1 + P_{16}^2}{1+\alpha}, \\ p_1 &= P_{16}^0 + \frac{(1+\alpha)P_{16}^1 + P_{16}^2}{\alpha}, \quad & p_3 &= \frac{P_{16}^1 + P_{16}^2}{\alpha+\alpha^2}, \end{aligned} \tag{37}$$

In decoding, we assume that the received codeword $\mathbf{c}' = [c_0' \ \cdots \ c_{13}']$ has 4 erasures at c_i, $i \in \{2, 3, 5, 7\}$. We define $\mathbf{x} = [c_1' \ \cdots \ c_{13}' \ 0\ 0\ 0]^{\mathrm{T}}$ from (35) and set the erased value to zero, then obtain $\{P_{16}^i\}_{i=0}^3$ in the same way as encoding. We have $[s_0\ s_1\ s_2\ s_3]$ from (36). Finally, the erased symbols can be calculated via (34). The matrix $\mathbf{H}_{de}$ is a submatrix consisting of the corresponding columns of erasures in $\mathbf{H}$. Some optimization techniques and further extensions for the proposed algorithm are provided in Appendix A.1.

4 Theoretical analysis

This section analyzes the complexity of the proposed algorithm. We show that the proposed algorithm asymptotically requires 3 XORs per data bit. Note that some of the notations follow the definition in Table 1, and the new notations for complexity analysis are defined in Table 2.

4.1 Complexity analysis

From Section 3.2, the encoding and decoding of the proposed algorithm first require the syndrome to be calculated through the Reed–Muller transform, and then the computed syndrome is multiplied by an inverse matrix. As the matrix–vector product requires $\mathcal{O}(1)$ operations for $t \leq 7$, the complexity of the proposed algorithm is dominated by the syndrome computation. Nevertheless, at low coding rates, the complexity caused by the fixed inverse matrix accounts for a high proportion of the algorithm. Therefore, we analyze the complexity of the algorithm from the above two processes.

We first consider the syndrome computation for $N = 2^r = 2^{\lceil \log_2(n-1) \rceil}$. From (19), (22), and (26), the proposed algorithm only requires the result of the Reed–Muller transform, where a portion of $\mathbf{y}$ is the output to the transform. Specifically, the proposed algorithm requires y_0, $\{y_{2^i}\}_{0\leq i<r}$, and $\{y_{2^i+2^j}\}_{0\leq i<j<r}$. Fig. 4 shows the signal-flow diagram for the case $\{P_N^i\}_{i=0}^6$ with $N = 8$. Each node with two incoming solid arrows denotes a field addition. If the number of additions required to compute y_0, $\{y_{2^i}\}_{0\leq i<r}$, and $\{y_{2^i+2^j}\}_{0\leq i<j<r}$ is denoted as $\Theta(N)$. Then, we have $\Theta(2) = 1$ and

$$\Theta(N) = \Theta(N/2) + \Theta'(N/2) + N/2, \tag{38}$$

where $N/2$ refers to the number of cross-additions (e.g., $x_i + x_{i+4}$ for $i = 0, \cdots, 3$), and $\Theta'(N/2)$ refers to the operations on the latter half of the input data. Note that $\Theta'(\bullet)$ has the recurrence relation

$$\begin{cases} \Theta'(2) = 1, \\ \Theta'(N) = \Theta'(N/2) + N - 1, \end{cases} \tag{39}$$

and we have

$$\Theta'(N) = 2N - r - 2. \tag{40}$$

Thus, (38) can be written as

$$\begin{aligned} &\Theta(N) = \Theta(N/2) + 3N/2 - r - 1 \\ \Rightarrow\ &\Theta(N) = 3N - r(r+3)/2 - 3. \end{aligned} \tag{41}$$

Moreover, in Fig. 4, the dashed arrows denote that the value will be multiplied by a constant factor prior to the addition. Specifically, for $i = 1, 2, 4$, computing P_N^i from $\{y_{2^i}\}_{0\leq i<r}$ requires $r-1$ additions and r multiplications. For $i = 3, 5, 6$, computing P_N^i from $\{y_{2^i}\}_{0\leq i<r}$ and $\{y_{2^i+2^j}\}_{0\leq i<j<r}$ requires $(r^2+r-2)/2$ additions and $(r^2+r)/2$ multiplications.

Then we consider the complexity of multiplying the fixed inverse matrix by the syndrome. In the decoding stage, calculating $\mathbf{H}_{de}^{-1}$s requires $t^2 - t$ additions and t^2 multiplications. In the encoding stage, computing (31) requires

TABLE 3: The avg.XORs of MDS encoding algorithms, for $t = 4, 5, 6, 7$

Encoding algorithms	avg.XORs
Circulant Cauchy Codes [17]	$3t$
Rabin-like Codes [18]	$2t$
Trifonov's alg. [3]	t
FFT-based alg. [19], [23]	$1 + \lg t$
Ours	3

$t^2 - 3t + 2$ additions and multiplications. The details are shown in Appendix A.1 and the zeros in $\mathbf{H}_{en}^{-1}$.

Let $\{\mathrm{A}_i(N)\}_{i=1}^4$ and $\{\mathrm{M}_i(N)\}_{i=1}^4$ denote the number of additions and multiplications in the computations of parities for $t = i + 3$. From the above analysis, we then have

$$\begin{aligned} &\mathrm{A}_i(N) = \Theta(N) + t^2 + \mathcal{O}(r^2/2 - t), \\ \Rightarrow\quad &\mathrm{A}_i(N) = 3N + t^2 + \mathcal{O}(r^2/2 - t), \end{aligned} \tag{42}$$

for $i = 1, 2, 3, 4$. Furthermore, the multiplicative complexity is given by

$$\mathrm{M}_i(N) = t^2 + \mathcal{O}(r^2/2 - t), \quad i = 1, 2, 3, 4. \tag{43}$$

4.2 Average number of Operations

Table 3 lists the number of XORs for each number of data bits in various encoding methods. In particular, the average number of XORs (avg.XORs) is defined as

$$\text{avg.XORs} = \frac{\text{Total number of XORs}}{\text{Number of data bits}}.$$

The proposed algorithm is analyzed as follows. When the elements of $\mathbb{F}_{2^m}$ are represented by binary polynomials, the field addition requires m XORs. However, the field multiplication is much more complicated, and it requires $\mathcal{O}(m^2)$ XORs from direct implementation. In addition, the multiplication can be implemented by using a logarithm table and an exponential table. Table 4 lists the costs of field multiplications with different implementations. In particular, [3] gives a C function to calculate 16 field multiplications in $\mathbb{F}_{2^8}$ in parallel with SIMD instruction set. In this case, Table 4 gives the number of instructions used in the C function. Notably, In Table 3, the field multiplication is implemented by pure XORs. From (16), the maximum codeword length is $n.max = N + 1$. When $m = \mathcal{O}(r)$, the proposed algorithm requires

$$\begin{aligned} \text{avg.XORs} &= \frac{\mathrm{A}_i(N) \cdot m + \mathrm{M}_i(N) \cdot \mathcal{O}(m^2)}{(n.max - t) \cdot m} \\ &\approx 3 + \frac{1}{N} \cdot \mathcal{O}(r^3/2 + r \cdot t^2), \end{aligned} \tag{44}$$

which approaches 3 when N approaches infinity. Notably, one can obtain different analysis results from (44) by considering different implementations of field multiplications in Table 4.

In addition, the average number of additions and multiplications for certain cases (k, t) is shown in Table 5, where k and t denote the number of data symbols and parity symbols, respectively. From [19], the encoding algorithm based on FFT requires a total of $n \lg t + k - t$ field additions and $\frac{n}{2} \lg t$ field multiplications. The encoding algorithm [3] requires around t additions and one multiplication per data symbol. For the proposed algorithm with optimization, the number of additions is close to 3 per symbol and the multiplicative complexity is similar to that given in [3].

TABLE 4: Cost of the field multiplication in $\mathbb{F}_{2^m}$

Methods	Costs
Pure XORs	$\mathcal{O}(m^2)$ XORs
Table lookup	1 integer addition and 2 table lookups
Method in [3] ($\mathbb{F}_{2^8}$)	1 shift-right ops, 2 bitwise AND ops, 1 bitwise XOR, and 2 table lookups.

TABLE 5: The complexities of RS encoding algorithms

Config.		Number of additions/multiplications per symbol		
k	t	Our alg.	Trifonov's alg. [3]	FFT-based alg. in [19]
32	4	3.13/0.75	4.72/0.81	3.13/1.00
48	5	3.25/0.65	4.65/0.52	4.23/1.73
62	6	3.58/0.87	5.9/0.63	4.19/1.68

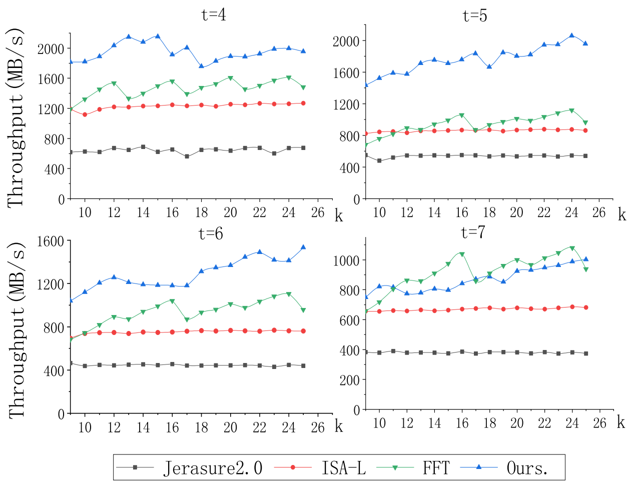

Fig. 5: Encoding performance for $t = 4, 5, 6, 7$

5 Simulations and Evaluation

To demonstrate the real performance of the proposed method on general-purpose CPUs, simulations were performed using an Intel Core i7-4500U with a 1.80 GHz CPU, 8 GB DDR3L, and disable Intel SpeedStep technology. We implemented the proposed approaches over $\mathbb{F}_{2^8}$ with Intel's SSE Notably, the primitive polynomial was chosen to be $\pi(x) = x^8 + x^4 + x^3 + x^2 + 1$. We conducted several simulations that compared the proposed algorithm with the FFT-based approaches [19], the erasure coding libraries ISA-L [20] and Jerasure 2.0 [21]. Note that Jerasure 2.0 adopts the Cauchy-RS scheme for coding, which uses the optimization in [10], [24].

In the first simulation, we measured the throughput of coding cache data with the various procedures. The size of the cache was $\lambda \cdot k$ bytes, where $\lambda = 4096$ is the stripe unit size. Fig. 5 and Fig. 6 depict the throughput of the procedures with $t = 4, 5, 6, 7$, which improves with the length of the code, and the proposed encoding at $t = 5$ is about twice as fast as the other methods. Moreover, as shown in Fig. 6, the proposed decoding method is better than other methods

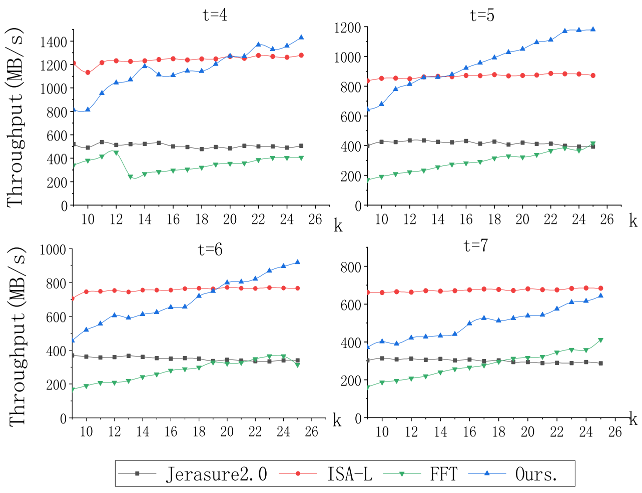

Fig. 6: Decoding performance for $t = 4, 5, 6, 7$

when k is above some threshold. Specifically, the proposed decoding algorithm has two steps (see Section 3.2.2), namely the syndrome computation $\mathbf{s} = \mathbf{H}\mathbf{c}'$ and the erasure calculation $\mathbf{H}_{de}^{-1}\mathbf{s}$. The first step is performed by the fast syndrome computation presented in Section 3, and the second step is performed by a conventional matrix–vector multiplication. Thus, the complexity is $\mathcal{O}(3(k+t)+t^2)$ (when $t \leq 7$). The RS decoding of ISA-L (and Jerasure) calculates $\mathbf{H}_0\mathbf{c}'$ using a conventional matrix–vector multiplication, where $\mathbf{H}_0 = (\mathbf{H}_{de}^{-1}\mathbf{H})$. Thus, the complexity is $\mathcal{O}(kt)$. When k is less than t, the $\mathcal{O}(kt)$ of ISA-L is faster than the $\mathcal{O}(3(k+t)+t^2)$ of the proposed algorithm. In addition, ISA-L is written in assembly language and is highly optimized. In contrast, the proposed approach is implemented in C. Hence, the threshold in Fig. 6 is slightly larger than the theoretical value. In the proposed procedure, the encoding is faster than the decoding. This is due to the input $\mathbf{x}$ used in encoding has more zeros than that used for decoding and the complexity of calculating (31) can be reduced following optimization.

From [22], the performance of field multiplication takes about four times as long as field addition. Due to page limits, the number of operations used in various procedures is discussed in Appendix B, and the performance of different methods to coding a 1-GB file in practical application is tested.

6 Conclusion

In this paper, we have presented efficient RS encoding algorithms for $t = 4, 5, 6$, and 7. The proposed algorithm asymptotically requires 3 XORs per data bit, representing an improvement over prior results (see Table 3). Furthermore, an SIMD implementation was also demonstrated. The performance of the proposed algorithm improves as the code length increases, and is superior to that of other cutting-edge methods. In particular, when the parity number is 5, the proposed algorithm is about 2 times faster than the FFT algorithm for finite fields. In the future, the proposed method will be implemented in a real system to verify its performance.

References

[1] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM SIGCOMM computer communication review*, vol. 27, no. 2, pp. 24–36, 1997.

[2] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5227–5239, Aug 2011.

[3] P. Trifonov, "Low-complexity implementation of RAID based on Reed-Solomon codes," *ACM Transactions on Storage (TOS)*, vol. 11, no. 1, p. 1, 2015.

[4] N. Desai, "Erasure code support - swift 2.9.1.dev229 documentation," 2016. [Online]. Available: http://docs.openstack.org/developer/swift/overview erasure code.html

[5] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320.

[6] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI'10)*, vol. 10, 2010, pp. 1–7.

[7] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci *et al.*, "Windows Azure Storage: a highly available cloud storage service with strong consistency," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 143–157.

[8] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang *et al.*, "f4: Facebooks warm BLOB storage system," in *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*. USENIX Association, 2014, pp. 383–398.

[9] R. Bhaskar, P. K. Dubey, V. Kumar, and A. Rudra, "Efficient Galois field arithmetic on SIMD architectures," in *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*. ACM, 2003, pp. 256–257.

[10] J. Blmer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," 1995.

[11] M. Blaum and R. M. Roth, "On lowest density MDS codes," *IEEE Transactions on Information Theory*, vol. 45, no. 1, pp. 46–59, Jan 1999.

[12] J. S. Plank, "XOR's, lower bounds and MDS codes for storage," in *Information Theory Workshop (ITW), 2011 IEEE*, Oct 2011, pp. 503–507.

[13] S. J. Lin, A. Alloum, and T. Y. Al-Naffouri, "Raid-6 Reed-Solomon codes with asymptotically optimal arithmetic complexities," in *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Sept 2016, pp. 1–5.

[14] S. J. Lin, "An encoding algorithm of triply extended Reed-Solomon codes with asymptotically optimal complexities," *IEEE Transactions on Communications*, vol. PP, no. 99, pp. 1–1, 2017.

[15] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: Novel erasure codes for big data," in *Proceedings of the VLDB Endowment*, vol. 6, no. 5. VLDB Endowment, 2013, pp. 325–336.

[16] A. Mazzoleni, "Btrfs: lib: raid: New RAID library supporting up to six parities," 2014. [Online]. Available: https://lwn.net/Articles/579034/

[17] C. Schindelhauer and C. Ortolf, "Maximum distance separable codes based on circulant cauchy matrices," in *Structural Information and Communication Complexity*, T. Moscibroda and A. A. Rescigno, Eds. Cham: Springer International Publishing, 2013, pp. 334–345.

[18] G. L. Feng, R. H. Deng, F. Bao, and J. C. Shen, "New efficient MDS array codes for RAID. part II. Rabin-like codes for tolerating multiple (/spl ges/4) disk failures," *IEEE Transactions on Computers*, vol. 54, no. 12, pp. 1473–1483, 2005.

[19] S. J. Lin, T. Y. Al Naffouri, and Y. S. Han, "FFT algorithm for binary extension finite fields and its application to Reed–Solomon codes," *IEEE Transactions on Information Theory*, vol. 62, no. 10, pp. 5343–5358, 2016.

[20] Intel, "Intelligent storage acceleration library," https://github.com/01org/isa-l.

[21] J. Plank and K. Greenan, "Jerasure: A library in c facilitating erasure coding for storage applications _ version 2.0. technical report ut-eecs-14–721," *University of Tennessee*, 2014.

[22] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, Z. Wilcox-O'Hearn *et al.*, "A performance evaluation and examination of open-source erasure coding libraries for storage." in *Fast*, vol. 9, 2009, pp. 253–265.

[23] S. J. Lin, W. H. Chung, and Y. S. Han, "Novel polynomial basis and its application to Reed-Solomon erasure codes," in *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. IEEE, 2014, pp. 316–325.

[24] J. S. Plank, "Optimizing Cauchy Reed-Solomon codes for fault-tolerant storage applications," *University of Tennessee, Tech. Rep. CS-05-569*, 2005.