Low-Complexity Implementation of RAID Based on Reed-Solomon Codes

P. TRIFONOV, Saint-Petersburg State Polytechnic University

Fast algorithms are proposed for encoding and reconstructing data in RAID based on Reed-Solomon codes. The proposed approach is based on the cyclotomic fast Fourier transform algorithm and enables one to significantly reduce the number of expensive Galois field multiplications required. The complexity of the obtained algorithms is much lower than those for existing MDS array codes. Software implementation of the proposed algorithms is discussed. The performance results show that the new algorithms provide substantially better performance compared with the standard algorithm.

Categories and Subject Descriptors: [**Storage Replication**]: RAID

General Terms: Design, Algorithms, Performance, Reliability

Additional Key Words and Phrases: Reed-Solomon codes, fast algorithms, FFT, RAID

ACM Reference Format:
P. Trifonov. 2015. Low-complexity implementation of RAID based on Reed-Solomon codes. ACM Trans. Storage 11, 1, Article 1 (February 2015), 25 pages.
DOI: http://dx.doi.org/10.1145/2700308

1. INTRODUCTION

The increasing amount of data available in digital formats requires the development of appropriate storage techniques. In many cases, the amount of data to be stored exceeds the capacity of a single disk drive. Additionally, the reliability of a single drive may not be sufficient for a particular application. These circumstances motivate the development of redundant arrays of independent disks (RAIDs). The size of these arrays may vary from a few disks to a few thousand disks. A major challenge in their design is the development of efficient algorithms for calculating parity data and for recovering the data in the case of disk failures. Similar problems also arise in the design of cloud storage systems.

RAID designs are typically based on some type of maximum distance separable (MDS) code [Moon 2005]. Reed-Solomon (RS) codes are an old and well-studied class of such codes. However, there is a common belief that the complexity of the associated algorithms is too high for practical usage. The reason for this belief is that the implementation of RS codes requires employing arithmetics over $GF(2^m)$. Furthermore,

This work was developed in collaboration with EMC Corporation. This work was also supported by grant MK-5407.2013.9 of the president of Russia.
Author's address: P. Trifonov, Distributed Computing and Networking Department, Saint-Petersburg State Polytechnic University, Polytechnicheskaya str., 21, office 104. 194021, Saint-Petersburg, Russia; email: petert@dcn.icc.spbstu.ru.

DOI: http://dx.doi.org/10.1145/2700308

most of the existing implementations of RS-based RAID make no attempt to use the structure of the codes to reduce the complexity. Indeed, the encoding techniques presented in Plank [1997] and Hsieh et al. [2004] essentially reduce to multiplication by a generator matrix of the RS code.

An implementation of the Reed-Solomon encoding and decoding algorithms having a complexity $O(n \log n)$ based on a Fermat number transform was suggested in Soro and Lacan [2010]. However, the considered codes are defined over $GF(p)$, where p is a Fermat prime, making them very difficult to use in a practical storage system due to the extensive bit packing needed. Furthermore, this approach relies on the Schönhage-Strassen fast polynomial multiplication algorithm, which becomes practical only for very large problems. A more practical approach is taken in Didier [2009], where a fast Hadamard transform is employed to obtain an erasure recovery algorithm with complexity $O(2^m m^2)$.

Due to the high complexity of the straightforward implementation of Reed-Solomon codes, several classes of MDS array codes have been proposed, for example, EvenOdd [Blaum et al. 1995], RDP [Corbett et al. 2004], X-code [Xu and Bruck 1999], and zigzag codes [Tamo et al. 2011]. However, the application of these codes in storage systems results in a huge stripe size, and the probability of a very expensive partial update operation increases with the stripe size. Although some of these codes achieve theoretically optimal partial update complexity, the need to fetch old pieces of codewords, recompute them, and write back may severely degrade the performance of a storage system.

In this article, we show that by exploiting the algebraic structure of RS codes, one can significantly decrease the encoding and reconstruction complexity to less than that of MDS array codes based on Cauchy matrices, which employ only XOR operations. The proposed approach is based on the cyclotomic FFT algorithm for finite fields [Trifonov and Fedorenko 2003; Costa et al. 2004; Fedorenko 2006, 2011b; Wu et al. 2011; Bellini et al. 2011; Wu et al. 2012; Fedorenko 2011a; Al Ghouwayel et al. 2007; Wu and Yan 2011; Chen and Yan 2009a, 2009b; Lin et al. 2007; Trifonov 2012]. The main idea is to implement encoding by invoking the erasure decoding algorithm. This approach enables the application of FFT techniques and, together with the optimized placement of information symbols within the codeword, results in a significant reduction of the encoding complexity. Moreover, some modifications to the code structure enable one to further decrease the encoding complexity at the expense of a slightly higher rebuild cost. This approach improves the average system performance because array rebuild is a relatively rare event.

The proposed approach is tailored toward software implementation. Such an implementation was developed, and its computational performance was studied. The proposed approach appears to provide approximately three times better performance compared with the standard RS encoding algorithm and is well suited for use with modern SIMD-based CPUs.

The algorithms proposed in this article are based on certain algebraic properties of Galois fields. This article presents a survey of these properties and proofs for some of them, although they can be found in the appropriate textbooks [Lidl and Niederreiter 1996; MacWilliams and Sloane 1977; Moon 2005]. The reader may skip these proofs. However, understanding the concepts discussed in Section 2.1 is essential if one decides to implement the proposed approach.

The article is organized as follows. Section 2 introduces the necessary background. Fast encoding and erasure reconstruction algorithms are presented in Section 3. The performance evaluation is provided in Section 4. The comparison with modern array codes is provided in Section 5. Finally, some conclusions are drawn.

2. BACKGROUND

2.1. Finite Fields

This section presents a brief introduction to finite fields and outlines their properties, which are used extensively in the proposed algorithms. A more detailed discussion can be found in classical textbooks [Lidl and Niederreiter 1996; MacWilliams and Sloane 1977; Moon 2005].

A Galois field $GF(2)$ is a set of integers $\{0, 1\}$ with the addition operation defined as eXclusive OR and with the multiplication operation given by the logical AND. The subtraction operation is the same as addition.

Given a polynomial $\pi(y) = \sum_{i=0}^{m} \pi_i y^i, \pi_i \in GF(2), \pi_m = 1$, such that $\pi(y)$ cannot be represented as a product of smaller degree polynomials, one defines an extended Galois field $GF(2^m)$ as a set of polynomials[1] $a(y) = \sum_{i=0}^{m-1} a_i y^i, a_i \in GF(2)$, where multiplication is performed modulo $\pi(y)$. Any element of this field can be represented as a vector $(a_0, \ldots, a_{m-1})$. The addition and subtraction (these are equivalent operations) of such vectors are performed componentwise. The set of binary vectors of length m appears to be a linear space. One can identify some linearly independent elements (i.e., a basis) $\beta_0, \ldots, \beta_{m-1} \in GF(2^m)$ such that any element of $GF(2^m)$ can be expressed as $a = \sum_{i=0}^{m-1} b_i \beta_i$, with $b_i \in GF(2)$. The two most important types of bases are the standard and normal bases.

The standard basis is given by $\beta_i = \alpha^i, i = 0, \ldots, m-1$, where α is a primitive element of $GF(2^m)$, that is, an element such that any nonzero $x \in GF(2^m)$ can be obtained as $x = \alpha^j, 0 \le j < 2^m - 1$. It is convenient to select $\pi(y)$ so that $\pi(\alpha) = 0$. Such polynomials are called primitive. Given a primitive polynomial, one can obtain all nonzero elements of $GF(2^m)$ by repeated application of the identities $\alpha^i = \alpha \cdot \alpha^{i-1}$ and $\alpha^m = \sum_{i=0}^{m-1} \pi_i \alpha^i$. It is convenient to simultaneously consider two different representations of nonzero elements $a \in GF(2^m)$: a vector representation $a = (a_0, \ldots, a_{m-1})$ (or, equivalently, $a = \sum_{i=0}^{m-1} a_i \alpha^i$) and an index representation $a = \alpha^i$. The first is convenient for implementing summation, and the second is well suited for multiplication and division.

The normal basis is given by linearly independent elements $\beta_i = \gamma^{2^i}, 0 \le i < m$, where $\gamma \in GF(2^m)$ is a normal basis generator. Not every element can be used to generate a normal basis, but it is possible to show that a suitable generator can always be found. An important property of the normal basis is that

$$\sum_{s=0}^{m-1} \gamma^{2^s} = 1. \tag{1}$$

All elements x of $GF(2^m)$ satisfy the equation $x^{2^m} - x = 0$. For example, for any element of $GF(2) = \{0, 1\}$, one obtains $x^2 = x$.

A Galois field $GF(2^m)$ has subfields $GF(2^{m'})$ such that m is divisible by m'. A subfield is a set of elements of $GF(2^m)$ that are closed under addition, multiplication, and division. For example, $GF(2) = \{0, 1\}$ and $GF(2^2) = \{0, 1, \alpha^5, \alpha^{10}\}$, where α is a primitive element of $GF(2^4)$, are subfields of $GF(2^4)$.

Furthermore, subsets of nonzero elements of $GF(2^m)$ exist that are closed only under multiplication and division operations. Such subsets are called subgroups of the multiplicative group of $GF(2^m)$. The number of elements n in such a subgroup (called the subgroup order) must be a divisor of $2^m - 1$. All elements of a subgroup of order n

[1]The algorithms presented use polynomials in the variable x with coefficients in $GF(2^m)$. To avoid confusion, the elements of $GF(2^m)$ are always considered vectors and not polynomials.

can be obtained as $x_i = \omega^i, 0 \leq i < n$, where $\omega = \alpha^{\frac{2^m-1}{n}}$ is a generating element of the subgroup.

Another property of finite fields, which greatly simplifies calculations, is that any two elements a, b of $GF(2^m)$ satisfy

$$(a+b)^2 = a^2 + b^2. \tag{2}$$

2.2. Error-Correcting Codes

To protect data in a storage system against disk failures, some redundancy needs to be introduced. Each stripe of data can be observed as a codeword of some code, where the ith codeword symbol is stored on the ith disk.[2] Disk failure results in the corresponding codeword symbol being erased. Erasure-correcting codes constitute a special case of error-correcting codes. In this section, standard notation from the theory of error-correcting codes is introduced.

A linear code $\mathcal{C}$ of length n, dimension k, and minimum distance d over $GF(q)$ is defined as

$$\mathcal{C} = \left\{c = (c_0, \ldots, c_{n-1}) \in GF(q)^n | c = wG, w \in GF(q)^k\right\},$$

where G is a $k \times n$ full-rank generator matrix, such that any two distinct codewords c differ in at least d positions. Only the case of $q = 2^m$ will be considered in this article. The parameters of a code are usually specified as a triple (n, k, d). A code with a minimum distance d can correct any configuration of $d-1$ erasures. If more symbols are erased, there may exist codewords that fully agree on nonerased positions and are therefore indistinguishable. Such erasure configurations are not correctable.

Given a code $\mathcal{C}$, one can construct many different generator matrices for the code given by $G' = QG$, where Q are nonsingular matrices. Different generator matrices define different mappings from information vectors w to codewords c, but the set of codewords and error- and erasure-correcting properties of a code are not affected by the choice of the generator matrix. In practice, it is beneficial to use the generator matrix $G' = QG = (I|A)$ containing a $k \times k$ identity submatrix I and some $k \times (n-k)$ matrix A. This results in systematic encoding; that is, the information vector w appears as a subvector of the codeword. The encoding operation becomes, in this case,

$$c' = wG' = (w|wA); \tag{3}$$

that is, one needs to calculate only the parity vector wA of length $n-k$. This calculation not only reduces the encoding complexity from $O(kn)$ to $O(k(n-k))$ but also enables one to extract the nonerased data symbols directly from a codeword without any calculations. Note that the identity matrix does not need to be placed in the first k columns within the generator matrix G' but can be distributed over the generator matrix in an arbitrary way.

Any linear code can also be characterized by its $(n-k) \times n$ check matrix H, whereby any codeword c satisfies $cH^T = 0$. Generator and check matrices are related by the equation $GH^T = 0$.

It is possible to show that any code satisfies $d \leq n - k + 1$. Codes achieving this bound with equality are called maximum distance separable (MDS). For MDS codes, given arbitrary k symbols of a codeword, one can reconstruct the remaining symbols. Reed-Solomon codes are the most widely known class of MDS codes.

[2]A practical storage system may use more sophisticated mapping schemes (e.g., cyclic mapping is used in RAID-5), which are beyond the scope of this article.

2.3. Discrete Fourier Transform

The proposed approach relies heavily on the discrete Fourier transform (DFT). DFTs are used extensively in digital signal processing not only as a tool for studying waveform properties but also as a computational primitive, which enables efficient calculations. A DFT is commonly implemented using some fast Fourier transform (FFT) algorithm. Although vectors of Galois field elements have little in common with real or complex waveforms, the DFT machinery can still be used for their processing.

Definition 2.1. An N-point discrete Fourier transform of the vector $(f_0, \ldots, f_{N-1})$, $f_i \in GF(q)$ (or the corresponding polynomial $f(x) = \sum_{i=0}^{N-1} f_i x^i$) is given by

$$F_j = \sum_{i=0}^{N-1} f_i \omega^{ij} = f(\omega^j), 0 \le j < N, \tag{4}$$

where $\omega \in GF(q)$ is the DFT kernel, which must be a generator of a multiplicative subgroup of $GF(q)$ of order N, that is, an element such that $\omega^N = 1$ and $\omega^i \neq 1, 0 < i < N$. Observe that the standard complex-valued DFT is obtained by setting $\omega = e^{-\frac{2\pi \mathbf{i}}{N}}$, where $\mathbf{i}^2 = -1$.

The DFT of vector f is given by $F = Wf$, where W is the Vandermonde matrix

$$W = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{(N-1)2} & \ldots & \omega^{(N-1)(N-1)} \end{pmatrix}. \tag{5}$$

The inverse discrete Fourier transform of the vector $(F_0, \ldots, F_{N-1})$ is given by

$$f_i = \frac{1}{N} \sum_{j=0}^{N-1} F_j \omega^{-ij}. \tag{6}$$

It can be verified that $IDFT(DFT(f)) = f$. Observe that DFT can be considered as the evaluation of the polynomial $f(x) = \sum_{i=0}^{N-1} f_i x^i$ at the points ω^j. Hence, IDFT corresponds to interpolation over the points (ω^j, F_j), $0 \le j < N$, that is, the recovery of the polynomial coefficients f_i from the values $F_j = f(\omega^j)$.

In this article, we assume that $N = q - 1$ and that $\omega = \alpha$ is a primitive element of $GF(q)$. The factor $1/N$ in Equation (6) should be treated as $\underbrace{(1 + 1 + \cdots + 1)}_{N \text{ times}}^{-1} \in GF(q)$. In the case of $q = 2^m$, one obtains $1/N = 1$. Because $\omega^{-i} = \omega^{N-i}$, it follows that the IDFT of a vector is equal to the DFT of the same vector with permuted output components.

2.4. Reed-Solomon Codes

A common way to define an $(n, k, n - k + 1)$ Reed-Solomon code is to take a $k \times n$ Vandermonde matrix

$$G = \begin{pmatrix} \alpha^0 & \alpha^1 & \ldots & \alpha^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^0 & \alpha^k & \ldots & \alpha^{k(n-1)} \end{pmatrix}, \tag{7}$$

where α is a primitive element of $GF(2^m)$, and define the encoding operation as

$$c = wG, \tag{8}$$

where w is the data vector [MacWilliams and Sloane 1977; Plank 1997]. The codeword symbols c_i, $0 \le i < n$ can be stored on different disks in a storage system.

Assume for the moment that $n = N = 2^m - 1$. Encoding the data with a generator matrix given by Equation (7) results in the codeword symbols $c_i = \sum_{j=1}^{k} w_j \alpha^{ij}$. Specifically, encoding is equivalent to computing the DFT of the information polynomial[3] $w(x) = \sum_{j=1}^{k} w_j x^j$, so that $c_i = w(\alpha^i)$, $i = 0, \ldots, N-1$. Alternatively, the application of an IDFT to any codeword c of an $(N, k, N-k+1)$ Reed-Solomon code produces a vector having zero components at the positions $0, k+1, k+2, \ldots, N-1$. Because $\alpha^{-j} = \alpha^{N-j}$, one determines that the check matrix of the code is given by

$$H = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha^0 & \alpha^1 & \ldots & \alpha^{N-1} \\ \alpha^0 & \alpha^2 & \ldots & \alpha^{2(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^0 & \alpha^{N-k-1} & \ldots & \alpha^{(N-k-1)(N-1)} \end{pmatrix}. \tag{9}$$

Observe that both the generator and the check matrices of the Reed-Solomon code are submatrices of the DFT matrix W with $\omega = \alpha$.

Employing a generator matrix given by Equation (7) in a storage system results in an inefficient implementation because the payload data vector w is not stored as a part of a codeword. Hence, to recover the data, one would need to solve Equation (8) for w even in the absence of disk failures. This problem can be avoided if one employs systematic encoding so that Equation (3) can be used. Doing so, however, does not affect the construction of the check matrix H.

The construction of the Reed-Solomon code ensures that one can recover the data as long as at least k of the n symbols of the codeword c are not erased. A common way to recover $t \le n-k$ erasures is to construct a $t \times t$ matrix B that relates t erased and $n-t$ nonerased symbols, which are represented as the vectors $\widetilde{c}^T$ and $\widehat{c}$, respectively, and compute $\widetilde{c}^T = B\widehat{c}^T$. The matrix B can be computed with complexity $O(t^2(n-t))$ by inverting a certain submatrix of the generator matrix, and the cost of computing $\widetilde{c}$ is $O(t(n-t))$ [Plank 1997].

The complexity of the previous encoding and decoding algorithm becomes prohibitively high for large values of the code dimension k and the number of parity symbols $n-k$.

3. REED-SOLOMON CODES FOR RAID

3.1. An Outline of the Proposed Method

We propose to improve the efficiency of the Reed-Solomon encoding and erasure recovery procedure by employing techniques that are commonly used in the field of error correction coding [Moon 2005]. These techniques reduce to the DFT, a classical computational primitive. Numerous fast algorithms for computing the DFT are available. Furthermore, we propose to adjust the encoding scheme so that a specific fast Fourier transform algorithm becomes particularly efficient. This approach treats information and parity symbols uniformly. Hence, it can be used not only for rebuilding the data stored on failed disks but also at the encoding step, where one can employ the proposed erasure recovery method to obtain parity symbols from information symbols.

The proposed approach is based on syndrome decoding. Given a codeword with erased symbols, one can compute its syndrome. Informally, the syndrome is a small digest of

[3]We assume here that $w_0 = 0$ because this enables one to save a few operations in the algorithms derived later.

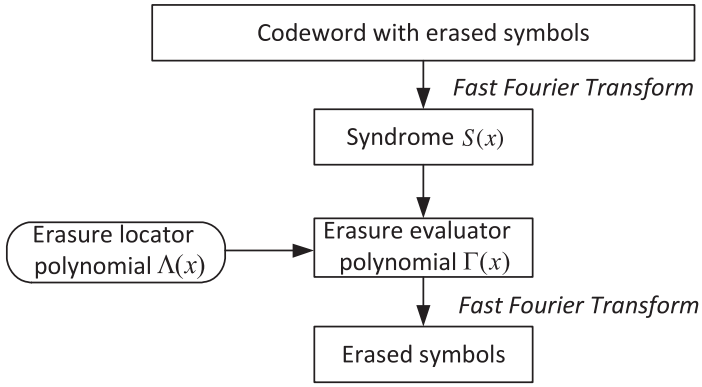

Fig. 1. Summary of the syndrome decoding method.

the vector being processed, which appears to depend only on unknown erased symbols, although it is computed only from the available data. Figure 1 illustrates the proposed approach.

The syndrome vector is computed from nonerased data by applying a fast Fourier transform algorithm to it. The syndrome vector is combined with an erasure locator polynomial, which describes a particular configuration of erased symbols, resulting in an erasure evaluator polynomial. The evaluation of this polynomial at appropriate points produces the values of the erased symbols. For the encoding step, an FFT can be used for its evaluation.

Essentially, the algorithms proposed in this article recover the erased symbols by computing them as

$$\widetilde{c} = T\,Q\widehat{c}, \tag{10}$$

where $\widehat{c}$ is the vector of nonerased symbols and T and Q are some fixed matrices. These matrices have a nice algebraic structure, which enables the construction of efficient algorithms for computing the product of these matrices and arbitrary vectors. It appears to be much easier to design two separate fast algorithms for multiplication by Q and T instead of directly searching for an efficient way to compute $\widetilde{c}^T = B\widehat{c}^T$, where $B = TQ$. The particular trick for enabling fast multiplication by T and Q is factoring them into products of two matrices. One of the matrices in these factorizations contains many zeroes and a few elements in $GF(2^m)$ arranged in a highly regular pattern. The second matrix in the factorizations contains only 0s and 1s.

3.2. Erasure Recovery via Syndrome Decoding

Given any k distinct points (α^i, c_i), one can use classical interpolation techniques (e.g., the Lagrange formula) to recover $w(x) : w(\alpha^i) = c_i$. However, this can be performed more efficiently by employing DFT techniques.

Assume for the moment that $n = N = 2^m - 1$. Then, any codeword of the Reed-Solomon code satisfies

$$Hc^T = 0, \tag{11}$$

where H is the check matrix given by Equation (9). Let $J = \{j_0, \ldots, j_{t-1}\}$ be the set of erased symbols. Then, Equation (11) is equivalent to

$$S_i = \sum_{j \in J} c_j \alpha^{ij} = \sum_{j \notin J} c_j \alpha^{ij}, 0 \le i < N - k, \tag{12}$$

where S_i are syndrome values. This equation can be represented in matrix notation as $\widetilde{H}\widetilde{c}^T = \widehat{H}\widehat{c}^T$, where $\widetilde{H}$ and $\widehat{H}$ are submatrices of H obtained by taking columns with indices corresponding to erased and nonerased symbols, respectively, and $\widetilde{c}$ and $\widehat{c}$ are the corresponding subvectors of c. Hence, the erased symbols can be recovered as

$$\widetilde{c}^T = \overline{H}\widehat{H}\widehat{c}^T, \tag{13}$$

where $\overline{H}$ is a $t \times (N-k)$ matrix such that $\overline{H}\widetilde{H} = I$. Observe that $\widehat{H}$ is the matrix Q in Equation (10). In what follows, efficient techniques are derived for the evaluation of Equation (13).

It is convenient to introduce a vector y that is equal to c on all nonerased positions and that has $y_{j_i} = 0, 0 \le i < t$. Then, the syndrome vector $(S_0, \ldots, S_{N-k-1})$ can be equivalently defined as

$$S_i = \sum_{j=0}^{N-1} y_j \alpha^{ij}, 0 \le i < N-k. \tag{14}$$

It is convenient to represent these values as a *syndrome polynomial* $S(x) = \sum_{i=0}^{N-k-1} S_i x^i$. Let us represent information about erased positions $j_0, \ldots, j_{t-1}$ via an *erasure locator polynomial*

$$\Lambda(x) = \prod_{i=0}^{t-1}(1-\alpha^{j_i}x) = \sum_{i=0}^{t}\Lambda_i x^i. \tag{15}$$

The *erasure evaluator polynomial* is defined as

$$\Gamma(x) = \Gamma_0 + \Gamma_1 x + \cdots + \Gamma_{N-k-1}x^{N-k-1} \equiv \Lambda(x)S(x) \bmod x^{N-k}. \tag{16}$$

This can be equivalently written as

$$\begin{pmatrix} \Gamma_0 \\ \Gamma_1 \\ \vdots \\ \Gamma_{N-k-1} \end{pmatrix} = \underbrace{\begin{pmatrix} \Lambda_0 & 0 & 0 & \ldots & 0 \\ \Lambda_1 & \Lambda_0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \Lambda_{N-k-1} & \Lambda_{N-k-2} & \Lambda_{N-k-3} & \ldots & \Lambda_0 \end{pmatrix}}_{\Lambda} \begin{pmatrix} S_0 \\ S_1 \\ \vdots \\ S_{N-k-1} \end{pmatrix}, \tag{17}$$

where $\Lambda_0 = 1$ and $\Lambda_i = 0, i > t$.

THEOREM 3.1 (FORNEY ALGORITHM). *The values of the erased codeword symbols can be obtained as*

$$c_{j_s} = \frac{\Gamma(\alpha^{-j_s})}{\prod_{p \ne s}(1-\alpha^{j_p}\alpha^{-j_s})} = \frac{\alpha^{j_s}\Gamma(\alpha^{-j_s})}{\Lambda'(\alpha^{-j_s})}, \tag{18}$$

where $\Lambda'(x) = \sum_{j=0}^{\lfloor (t-1)/2 \rfloor} \Lambda_{2j+1}x^{2j}$ *is the formal derivative of the erasure locator polynomial* $\Lambda(x)$.

PROOF. To show the first equality, observe that

$$1 \equiv 1 - (\alpha^{j_s}x)^{N-k} = (1-\alpha^{j_s}x)\sum_{i=0}^{N-k-1}\alpha^{ij_s}x^i \bmod x^{N-k}.$$

Hence, Equation (12) implies

$$S(x) = \sum_{s=0}^{t-1} c_{j_s} \sum_{i=0}^{N-k-1} \alpha^{ij_s} x^i \equiv \sum_{s=0}^{t-1} \frac{c_{j_s}}{1-\alpha^{j_s}x} \bmod x^{N-k},$$

and $\Gamma(x) = \sum_{s=0}^{t-1} c_{j_s} \prod_{p\neq s}(1-\alpha^{j_p}x)$. The second equality in Equation (18) follows because $\Lambda'(x) = ((\prod_{s=0}^{t-1}(-\alpha^{j_s}))(\prod_{s=0}^{t-1}(x-\alpha^{-j_s})))' = (-1)^t(\prod_{s=0}^{t-1}\alpha^{j_s})\sum_{s=0}^{t-1}\prod_{p\neq s}(x-\alpha^{-j_p}) = \sum_{s=0}^{t-1}\alpha^{j_s}\prod_{p\neq s}(\alpha^{j_p}x-1)$ so that $\Lambda'(\alpha^{-j_s}) = \alpha^{j_s}\prod_{p\neq s}(\alpha^{j_p}\alpha^{-j_s}-1)$. □

The previous proof shows that $\deg\Gamma(x) \leq t-1$. Therefore, Equation (16) can be simplified to

$$\Gamma(x) \equiv \Lambda(x)S(x) \bmod x^t. \tag{19}$$

Observe that Equation (18) can be represented as

$$\widetilde{c} = \begin{pmatrix} c_{j_0} \\ c_{j_1} \\ \vdots \\ c_{j_{t-1}} \end{pmatrix} = \underbrace{\begin{pmatrix} \beta_0 & \beta_0\alpha^{-j_0} & \dots & \beta_0\alpha^{-j_0(t-1)} \\ \beta_1 & \beta_1\alpha^{-j_1} & \dots & \beta_1\alpha^{-j_1(t-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{t-1} & \beta_{t-1}\alpha^{-j_{t-1}} & \dots & \beta_{t-1}\alpha^{-j_{t-1}(t-1)} \end{pmatrix}}_{Z} \begin{pmatrix} \Gamma_0 \\ \Gamma_1 \\ \vdots \\ \Gamma_{t-1} \end{pmatrix} = \underbrace{Z\Lambda}_{T} S, \tag{20}$$

where $\beta_s = \alpha^{j_s}/\Lambda'(\alpha^{-j_s}), 0 \leq s < t$. Here, the matrix T is the one introduced in Equation (10).

Hence, erasure decoding of the vector $(y_0, \dots, y_{N-1})$ can be implemented as follows:

(1) Construct the erasure locator polynomial $\Lambda(x)$ according to Equation (15). This step only needs to be performed once for a particular erasure configuration.
(2) Compute the syndrome polynomial $S(x)$ with coefficients given by Equation (14).
(3) Calculate the erasure evaluator polynomial according to Equation (19).
(4) Obtain the erased symbols c_{j_s} via Equation (18).

Observe that step 2 reduces to computing a few components of the DFT of the vector y. Fast algorithms can be used to implement this step. Furthermore, the evaluation of Equation (18), in some cases, can also be implemented with a DFT, as discussed later.

Steps 2 to 3 of this method can be implemented jointly as $\widetilde{c} = TS$. However, it is convenient to analyze them separately to obtain an efficient algorithm.

3.3. Cyclotomic Fast Fourier Transform

Numerous fast Fourier transform (FFT) algorithms are available [Chu and George 2000]. However, in most cases, they rely on the ability to factor the DFT dimension N into a few small prime numbers, and the most efficient implementation for the case of a complex field $\mathbb{C}$ is obtained in the case of $N = 2^l$. In the case of finite fields, the constraints on N do not allow one to obtain good factorizations. Furthermore, most of the standard FFT algorithms cannot efficiently address the case when only a few DFT components need to be computed. An alternative approach is to construct the FFT for finite fields from scratch by exploiting their algebraic properties.

A cyclotomic algorithm has been shown to be a very efficient way to compute the discrete Fourier transform over a finite field [Trifonov and Fedorenko 2003; Costa et al. 2004; Fedorenko 2006; Bellini et al. 2011]. The algorithm is aimed at the minimization of the number of finite field multiplications, which are assumed to be costly operations. Modern CPUs provide table lookup instructions, which enable one to reduce the

multiplication cost significantly, but as shown in Section 3.8, it remains substantially higher than the cost of XOR.

The main idea behind the proposed approach is to factor the DFT matrix W (see Equation (5)) into a product of a matrix over $GF(2^m)$ having very small nonzero blocks on the main diagonal and zero elsewhere and a binary matrix. The nonzero blocks of the first matrix appear to be highly structured, and classical fast cyclic convolution algorithms can be used to implement multiplication by the nonzero blocks. For the second matrix, computer optimization can be used to identify and eliminate common subexpressions arising while computing a product of this matrix and a vector.

3.3.1. The Algorithm.

Definition 3.2. Cyclotomic coset modulo N over $GF(2)$ is a set of integers $C_s = \{s, 2s, \ldots, 2^{m_s-1}s\}$, where $2^{m_s}s \equiv s \bmod N$. s is called the cyclotomic coset leader.

Observe that any cyclotomic coset element can be considered a coset leader.

Example 3.3. The following are cyclotomic cosets modulo 15 over $GF(2)$:

$$\{0\}, \{1, 2, 4, 8\}, \{3, 6, 12, 9\}, \{5, 10\}, \{7, 14, 13, 11\}.$$

Consider the construction of a DFT algorithm for the polynomial $f(x) = \sum_{i=0}^{N-1} f_i x^i$, that is, computing $F_j = f(\omega^j), 0 \le j < N$. Let us partition the set of output indices $\{0, \ldots, N-1\}$ into a number of cyclotomic cosets modulo N over $GF(2)$. Then, one obtains

$$F_{s2^j \bmod N} = f(\omega^{2^j s}) = \sum_{i=0}^{N-1} f_i \omega^{si2^j}, 0 \le j < m_s, s \in \mathcal{S}, \tag{21}$$

where $\mathcal{S}$ is the set of cyclotomic coset leaders. It is possible to show that ω^s is an element of $GF(2^{m_s}) \subset GF(2^m)$ for some m_s being a divisor of m. The same holds for ω^{si} for any i. For any finite field $GF(2^{m_s})$, there exists a normal basis $\{\gamma_s, \gamma_s^2, \ldots, \gamma_s^{2^{m_s-1}}\} \subset GF(2^{m_s})$ such that $\gamma_s^{2^{m_s}} = \gamma_s$, and any element $x \in GF(2^{m_s})$ can be represented as $x = \sum_{t=0}^{m_s-1} x_t \gamma_s^{2^t}, x_t \in GF(2)$. In particular, one obtains $\omega^{si} = \sum_{t=0}^{m_s-1} a_{s,t,i} \gamma_s^{2^t}, a_{s,t,i} \in GF(2)$. By the repeated application of Equation (2) and because $a_{s,t,i}^2 = a_{s,t,i}$, one can rewrite Equation (21) as

$$F_{s2^j \bmod N} = \sum_{i=0}^{N-1} f_i \sum_{t=0}^{m_s-1} a_{s,t,i} \gamma_s^{2^{t+j}}, \tag{22}$$

or, in matrix notation,

$$\tilde{F} = LAf, \tag{23}$$

where $\tilde{F}$ is the vector of the DFT components rearranged according to the cyclotomic cosets, L is a block-diagonal $N \times N$ matrix with the blocks $\mathbf{L}_s = \begin{pmatrix} \gamma_s^{2^0} & \gamma_s^{2^1} & \cdots & \gamma_s^{2^{m_s-1}} \\ \gamma_s^{2^1} & \gamma_s^{2^2} & \cdots & \gamma_s^{2^0} \\ \gamma_s^{2^2} & \gamma_s^{2^3} & \cdots & \gamma_s^{2^1} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_s^{2^{m_s-1}} & \gamma_s^{2^0} & \cdots & \gamma_s^{2^{m_s-2}} \end{pmatrix}$ on its main diagonal, and A is the $N \times N$ matrix consisting of $a_{s,t,i}$ coefficients.

Observe that $\mathbf{L}_s$ is a circulant matrix; that is, all of its rows can be obtained as cyclic shifts of the first row. Hence, computing $(F_s, F_{s2}, \ldots, F_{s2^{m_s-1}})^T = \mathbf{L}_s(u_{s,0}, u_{s,1}, \ldots, u_{s,m_s-1})^T$, where $u_{s,i}$ are the corresponding elements of the Af vector,

is equivalent (recall that $s2^{m_s} \equiv s \bmod N$ and $\gamma_s^{2^{m_s}} = \gamma_s$) to computing a cyclic convolution

$$\underbrace{F_{s2^{m_s} \bmod N} + F_{s2^{m_s-1} \bmod N} x^1 + \cdots + F_{s2^1 \bmod N} x^{m_s-1}}_{b_s(x)}$$
$$\equiv \underbrace{\left(\gamma_s^{2^{m_s}} + \gamma_s^{2^{m_s-1}} x^1 + \cdots + \gamma_s^{2^1} x^{m_s-1}\right)}_{g_s(x)} \underbrace{\left(u_{s,0} + u_{s,1} x^1 + \cdots + u_{s,m_s-1} x^{m_s-1}\right)}_{u_s(x)} \bmod (x^{m_s} - 1).$$

A straightforward method to compute[4]

$$b_s(x) = b_{s,m_s} + b_{s,m_s-1} x + b_{s,m_s-2} x^2 + \cdots + b_{s,1} x^{m_s-1} \equiv g_s(x) u_s(x) \bmod (x^{m_s} - 1),$$

where $b_{s,i} = F_{s2^i \bmod N}$, is given by

$$b_{s,m_s-i} = \sum_{j=0}^{m_s-1} u_{s,j} g_{s,i-j \bmod m_s},\ 0 \le i < m_s. \tag{24}$$

This expression requires $m_s(m_s - 1)$ summations and m_s^2 multiplications. Even if one proceeds in this manner, a complexity savings would already be provided compared to the straightforward evaluation of $F = Wf$. However, it is possible to further reduce the complexity by identifying common subexpressions in Equation (24), which can be evaluated once and used many times. This task can be accomplished by employing fast cyclic convolution algorithms [Knuth 1973; Blahut 1984]. Most of these algorithms can be represented as

$$b_s = \mathbf{P}_s((\mathbf{Q}_s g_s) \cdot (\mathbf{R}_s u_s)), \tag{25}$$

where $u_s, g_s,$ and b_s are vectors of coefficients of the polynomials $u_s(x), g_s(x),$ and $b_s(x)$, respectively; $\mathbf{Q}_s$ and $\mathbf{R}_s$ are $v_s \times m_s$ presummation matrices; $\cdot$ denotes the componentwise product of two vectors; $\mathbf{P}_s$ is a $m_s \times v_s$ postsummation matrix; and v_s is the number of multiplications needed by the cyclic convolution algorithm. The expression in brackets provides values for common subexpressions.

Observe that the polynomials $g_s(x)$ are always fixed in the case considered. Furthermore, many fast cyclic convolution algorithms have a matrix $\mathbf{Q}_s$ with a number of rows completely filled with 1s. It follows from the property (1) of normal bases that the corresponding entries of the $\mathbf{Q}_s g_s$ vector are always equal to 1; that is, some multiplications are effectively eliminated. For completeness, some cyclic convolution algorithms are provided in the appendix.

Combining Equations (23) with (25), one obtains

$$\tilde{F} = P\,(d \cdot Vf), \tag{26}$$

where P is a block-diagonal matrix with $\mathbf{P}_s$ on the main diagonal, d is obtained by concatenating the vectors $\mathbf{Q}_s g_s$, $V = RA$, and R is a block-diagonal matrix with $\mathbf{R}_s$ on the main diagonal. Assuming that the cost of computing an m-point cyclic convolution is $O(m \log^\lambda m), \lambda \ge 1$, one obtains that the number of multiplications needed by the cyclotomic FFT algorithm is given by $O(\frac{N}{m} m \log^\lambda m) = O(N \log^\lambda \log N)$. Here, it was further assumed that almost all cyclotomic cosets have a size $m_s = m \approx \log N$. The multiplicative cost of this algorithm is much less than $O(N \log N)$, the complexity of generic FFT algorithms.

[4]The nonstandard ordering of the coefficients of $b_s(x)$ is selected so that it matches the expression $b_s = \mathbf{L}_s u_s$, which needs to be evaluated by the cyclotomic FFT algorithm.

Table I. Different Representations of $GF(2^3)$ Elements

Exponential	Polynomial	Integer Bit Mask	Normal Basis Expansion	Subfield
0	0	0	0	$GF(2)$
α^0	1	1	1	$GF(2)$
α^1	α	2	$\gamma^2+\gamma^4$	$GF(2^3)$
α^2	α^2	4	$\gamma+\gamma^4$	$GF(2^3)$
α^3	$\alpha+1$	3	γ	$GF(2^3)$
α^4	$\alpha^2+\alpha$	6	$\gamma+\gamma^2$	$GF(2^3)$
α^5	$\alpha^2+\alpha+1$	7	γ^2	$GF(2^3)$
α^6	α^2+1	5	γ^4	$GF(2^3)$

However, the number of summations needed by the cyclotomic algorithm, in general, appears to be higher than the number of summations needed by generic FFT techniques. Computer optimization can be used to construct an efficient algorithm for computing Vf [Trifonov 2007; Bellini et al. 2011]. The complexity reduction is achieved by identifying common subexpressions in the system of equations $w = Vf$ and then eliminating them. For small to moderate values of N, this approach was shown to result in FFT algorithms with record-breaking complexity. Alternatively, one can use fast multiple modulo reduction algorithms to implement this step with complexity $O(N \log^2 N)$ [Trifonov 2012]. The latter approach, however, may only be practical for very large N.

Example 3.4. Consider the construction of a 7-point DFT over $GF(2^3)$ with kernel $\omega = \alpha$, where α is a primitive root of $y^3 + y + 1$. Table I shows different representations of elements of this field. The first column gives the representations as a power of a primitive element, and the second column is obtained from the first one by applying the identity $\alpha^3 = \alpha + 1$. The third column is an integer representation that is commonly used in software implementations of Galois field arithmetic, which is obtained by substituting $\alpha = 2$, and the fourth column gives the normal basis expansion in the corresponding subfield, which is specified in the last column.

The normal basis of $GF(2) \subset GF(2^3)$ is $\{1\}$, and the normal basis of $GF(2^3)$ is $\{\gamma, \gamma^2, \gamma^4\}, \gamma = \alpha^3$. The elements 0 and 1 are in subfield $GF(2) \subset GF(2^3)$, and the remaining field elements are strictly in $GF(2^3)$. It can be observed from Table I that

$$
\begin{aligned}
F_0 &= f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6\\
F_1 &= f_0 + f_1(\gamma^2+\gamma^4) + f_2(\gamma^4+\gamma) + f_3\gamma + f_4(\gamma^2+\gamma) + f_5\gamma^4 + f_6\gamma^2\\
F_2 &= f_0 + f_1(\gamma+\gamma^4) + f_2(\gamma+\gamma^2) + f_3\gamma^2 + f_4(\gamma^2+\gamma^4) + f_5\gamma + f_6\gamma^4\\
F_4 &= f_0 + f_1(\gamma+\gamma^2) + f_2(\gamma^2+\gamma^4) + f_3\gamma^4 + f_4(\gamma+\gamma^4) + f_5\gamma^2 + f_6\gamma\\
F_3 &= f_0 + f_1\gamma + f_2\gamma^2 + f_3(\gamma+\gamma^4) + f_4\gamma^4 + f_5(\gamma^2+\gamma^4) + f_6(\gamma+\gamma^2)\\
F_6 &= f_0 + f_1\gamma^2 + f_2\gamma^4 + f_3(\gamma+\gamma^2) + f_4\gamma + f_5(\gamma+\gamma^4) + f_6(\gamma^2+\gamma^4)\\
F_5 &= f_0 + f_1\gamma^4 + f_2\gamma + f_3(\gamma^2+\gamma^4) + f_4\gamma^2 + f_5(\gamma+\gamma^2) + f_6(\gamma+\gamma^4).
\end{aligned}
$$

In matrix notation, this becomes

$$
\begin{pmatrix} F_0\\F_1\\F_2\\F_4\\F_3\\F_6\\F_5 \end{pmatrix} =
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & \gamma^1 & \gamma^2 & \gamma^4 & 0 & 0 & 0\\
0 & \gamma^2 & \gamma^4 & \gamma^1 & 0 & 0 & 0\\
0 & \gamma^4 & \gamma^1 & \gamma^2 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & \gamma^1 & \gamma^2 & \gamma^4\\
0 & 0 & 0 & 0 & \gamma^2 & \gamma^4 & \gamma^1\\
0 & 0 & 0 & 0 & \gamma^4 & \gamma^1 & \gamma^2
\end{pmatrix}
\underbrace{\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 0 & 1 & 1 & 1 & 0 & 0\\
1 & 1 & 0 & 0 & 1 & 0 & 1\\
1 & 1 & 1 & 0 & 0 & 1 & 0\\
1 & 1 & 0 & 1 & 0 & 0 & 1\\
1 & 0 & 1 & 0 & 0 & 1 & 1\\
1 & 0 & 0 & 1 & 1 & 1 & 0
\end{pmatrix}}_{A}
\begin{pmatrix} f_0\\f_1\\f_2\\f_3\\f_4\\f_5\\f_6 \end{pmatrix}.
$$

The fast 3-point cyclic convolution algorithm

$$b_s(x) = b_{s,3} + b_{s,2}x + b_{s,1}x^2 = (\gamma + \gamma^4 x + \gamma^2 x^2)(u_{s,0} + u_{s,1}x + u_{s,2}x^2) \bmod (x^3 - 1)$$

is given by Blahut [1984]

$$b_s = \begin{pmatrix} b_{s,3} \\ b_{s,1} \\ b_{s,2} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}}_{\mathbf{P}_s} \left(\left[\underbrace{\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}}_{\mathbf{Q}_s} \begin{pmatrix} \gamma \\ \gamma^4 \\ \gamma^2 \end{pmatrix} \right] \cdot \left[\underbrace{\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}}_{\mathbf{R}_s} \begin{pmatrix} u_{s,0} \\ u_{s,1} \\ u_{s,2} \end{pmatrix} \right] \right), s = 1, 3,$$

where $\cdot$ denotes the componentwise product of two vectors.

Therefore, one obtains

$$\begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_4 \\ F_3 \\ F_6 \\ F_5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & \end{pmatrix} \left(\begin{pmatrix} 1 \\ 1 \\ \gamma^4 + \gamma^2 \\ \gamma + \gamma^4 \\ \gamma + \gamma^2 \\ 1 \\ \gamma^4 + \gamma^2 \\ \gamma + \gamma^4 \\ \gamma + \gamma^2 \end{pmatrix} \cdot \left[\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} A \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{pmatrix} \right] \right)$$

$$= \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & \end{pmatrix}}_{P} \left(\underbrace{\begin{pmatrix} 1 \\ 1 \\ \alpha \\ \alpha^2 \\ \alpha + \alpha^2 \\ 1 \\ \alpha \\ \alpha^2 \\ \alpha + \alpha^2 \end{pmatrix}}_{d} \cdot \left[\underbrace{\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}}_{V} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{pmatrix} \right] \right).$$

Multiplication by 1 costs nothing; therefore, computing a 7-point DFT requires only six multiplications.

3.3.2. Fast Syndrome Evaluation. The cyclotomic FFT algorithm can be tailored for computing a truncated DFT as needed for computing the syndrome values in Equation (14). To do this, unneeded rows should be stricken from the matrix P, resulting in a matrix with a number of zero columns. These columns together with the corresponding rows of d and V should be eliminated. This method for computing the syndrome polynomial for Reed-Solomon codes was shown to exhibit the smallest complexity compared with the existing techniques [Costa et al. 2004]. In what follows, we denote by $\mathcal{F}_{t,\Theta}(f)$ a truncated cyclotomic FFT algorithm capable of computing $F_0, \ldots, F_{t-1}$ for any vector $f = (f_0, \ldots, f_{N-1})$ such that $f_i \in GF(2^m)$ for $i \in \Theta$, and $f_i = 0$ otherwise.

Alternatively, given a polynomial $f(x)$ of degree $d - 1$, one may be interested in computing its DFT components $F_j = f(\omega^j), j \in \Omega$ for some set Ω. Let $\tilde{\mathcal{F}}_{\Omega,d}(f)$ denote the algorithm obtained by truncating matrices in a similar way. Observe that $\mathcal{F}_{t,\{0,\ldots,d-1\}}(f) = \tilde{\mathcal{F}}_{\{0,\ldots,t-1\},d}(f)$ for any $f = (f_0, \ldots, f_{d-1})$.

Example 3.5. Consider computing the components F_0, F_1, F_2 of the 7-point DFT. Given the expression derived in the previous example, one must remove the last four rows and last four columns of the matrix P, the last four entries of the vector d, and the last four rows of the matrix V, which results in the following expression:

$$\begin{pmatrix} F_0 \\ F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \left(\begin{pmatrix} 1 \\ 1 \\ \alpha \\ \alpha^2 \\ \alpha + \alpha^2 \end{pmatrix} \cdot \left[\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{pmatrix} \right] \right).$$

Therefore, the truncated FFT algorithm $\mathcal{F}_{2,\{0,\ldots,6\}}(f) = \tilde{\mathcal{F}}_{\{0,\ldots,2\},7}(f)$ requires three multiplications by nontrivial elements of $GF(2^3)$. This is an improvement over the straightforward evaluation of $F_0, F_1,$ and F_2, which requires 12 multiplications.

3.4. Fast Encoding Algorithm

3.4.1. Encoding via Decoding. In practical storage systems, systematic encoding should be used; that is, the vector of information symbols should be a subvector of a codeword. Recall that the Reed-Solomon codes are maximum distance separable codes; therefore, information symbols can be placed anywhere within the codeword. We propose to implement systematic encoding by constructing a vector consisting of information symbols placed at certain positions within the codeword and by applying the previous erasure recovery method to compute the corresponding parity symbols. Let r be the number of parity symbols to be computed, and let $K = N - r$. Let $\Phi = \{\phi_0, \ldots, \phi_{r-1}\} \subset \{0, \ldots, 2^m - 2\}$ be the set of positions whereby parity symbols are placed within the codeword. The remaining $K = N - r$ symbols at the positions given by $\Psi' = \{0, \ldots, 2^m - 2\} \setminus \Phi$ can be used as information symbols.

Let the *check symbol locator polynomial* be defined as

$$\Lambda_c(x) = \prod_{i=0}^{r-1} (1 - \alpha^{\phi_i} x). \tag{27}$$

Then, the encoding of the payload data vector $(w_0, \ldots, w_{K-1})$ can be performed by computing its syndrome

$$S_i = \sum_{j=0}^{K-1} w_j \alpha^{\psi_j i}, i = 0, \ldots, r-1, \psi_j \in \Psi', \tag{28}$$

where ψ_j is the position of the jth information symbol and calculating the check symbol evaluator polynomial is performed by

$$\Gamma_c(x) \equiv \Lambda_c(x) S(x) \bmod x^r. \tag{29}$$

The check symbols can be reconstructed as

$$p_i = \beta_i \Gamma_c(\alpha^{-\phi_i}), 0 \leq i < r, \tag{30}$$

where $\beta_i = \frac{\alpha^{\phi_i}}{\Lambda_c'(\alpha^{-\phi_i})}$ are Forney coefficients and $\Lambda_c'(x)$ is the formal derivative of $\Lambda_c(x)$. These coefficients can be precomputed together with $\Lambda_c(x)$.

The most difficult step of this method is computing the syndrome components in Equation (28). This computation can be implemented efficiently using the truncated cyclotomic FFT algorithm $\mathcal{F}_{r,\Psi'}(w)$.

3.4.2. Optimized Allocation of Check Symbols. The previously described encoding method can be further simplified by the careful selection of check symbol locators ϕ_i.

THEOREM 3.6. *Let* $C_s = \{s, 2s, \ldots, 2^{m_s-1}s\}$ *be a cyclotomic coset modulo* $N = 2^m - 1$. *Then,*

$$\mu(x) = \prod_{j \in C_s} (x - \alpha^j) = \sum_{i=0}^{m_s} \mu_i x^i,$$

where α *is a primitive element of* $GF(2)$*, is a polynomial with binary coefficients* μ_i.

PROOF. Recall that $\alpha^N = 1$ and that $s2^{m_s} \equiv s \bmod N$. Then, $\sum_{i=0}^{m_s} \mu_i^2 x^{2i} = (\sum_{i=0}^{m_s} \mu_i x^i)^2 = \prod_{j=0}^{m_s-1}(x - \alpha^{s2^j})^2 = \prod_{j=0}^{m_s-1}(x^2 - \alpha^{s2^j}) = \sum_{i=0}^{m_s} \mu_i x^2$. Hence, one obtains $\mu_i^2 = \mu_i$, that is, $\mu_i \in \{0, 1\}$. □

Hence, if Φ is a union of a number of cyclotomic cosets over $GF(2)$ modulo $N = 2^m - 1$, then $\Lambda_c(x)$ becomes a product of polynomials with binary coefficients; therefore, its coefficients are also binary. In this case, the calculation of Equation (29) does not require any multiplications in $GF(2^m)$. In the special case where $\{\alpha^i | i \in \Phi\}$ is a set of rth roots of unity (i.e., $\Phi = \{j\frac{N}{r} | j = 0, \ldots, r-1\}$, where N is divisible by r), one obtains $\Lambda_c(x) = 1 - x^r$, and $\Gamma_c(x) \equiv S(x) \bmod x^r$, that is, the check symbol evaluator polynomial is essentially obtained for free. In addition, observe that in this case, $\Lambda_c'(x) = -rx^{r-1} \equiv x^{r-1} \bmod 2$, and one obtains $\beta_i = \alpha^{\phi_i r} = 1$.

Furthermore, the values $-\phi_j \bmod N$ also span a number of cyclotomic cosets, and the evaluation of $\Gamma_c(\alpha^{-\phi_i})$, as needed by Equation (30), can also be performed with the truncated cyclotomic FFT algorithm $\tilde{\mathcal{F}}_{\tilde{\Phi},r}(\Gamma)$, where $\tilde{\Phi} = \{-i \bmod N | i \in \Phi\}$. It becomes particularly efficient in this case because one does not need to truncate the blocks $\mathbf{L}_c$ and because the dimension of the input vector is only $r \ll N$.

Observe that one can construct Φ as a union of cyclotomic cosets only if there exists the decomposition

$$r = \sum_{\substack{m_i \geq 1 \\ m_i | m}} r_i m_i, r_i \geq 0,$$

where r_i does not exceed the number of cyclotomic cosets of size m_i. Such a decomposition can indeed be constructed for any r in the practically important case of $m = 2^l$.

Example 3.7. Consider the case of $GF(2^8)$, and let $r = 3$. Construct $\Phi = \{0, 85, 170\}$. It can be verified that $\Lambda_c(x) = (1 - x)(1 - \alpha^{85}x)(1 - \alpha^{170}x) = x^3 + 1$.

The elements α^{-85} and α^{-170} belong to the subfield $GF(2^2)$ of $GF(2^8)$, while $\alpha^0 \in GF(2)$. The normal basis of the subfield $GF(2) \subset GF(2^8)$ is given by $\gamma_0 = 1$, whereas the one for $GF(2^2) \subset GF(2^8)$ is generated by $\gamma_1 = \alpha^{85}$, where α is a primitive element of $GF(2^8)$. Hence, the problem of evaluating $\Gamma_c(x) = \Gamma_0 + \Gamma_1 x + \Gamma_2 x^2$ at the points $\alpha^{-\phi_i}, \phi_i \in \Phi$, can be rewritten as

$$\Gamma(\alpha^0) = \Gamma_0 + \Gamma_1 + \Gamma_2 \tag{31}$$

$$\Gamma(\alpha^{-85}) = \Gamma_0 + \Gamma_1\alpha^{170} + \Gamma_2\alpha^{85} \tag{32}$$

$$\Gamma(\alpha^{-170}) = \Gamma_0 + \Gamma_1\alpha^{85} + \Gamma_2\alpha^{170}, \tag{33}$$

Table II. Optimized Locators of Check Symbols

r	Φ_r	$\Lambda_c(x)$	Number of Summations	Number of Multiplications
1	{0}	$x+1$	0	0
2	{85,170}	x^2+x+1	2	1
3	{0,85,170}	x^3+1	6	1
4	{17,34,68,136}	x^4+x^3+1	17	5
5	{0,51,102,153,204}	x^5+1	18	5
6	{170,85, 17,34,68,136}	$x^6+x^3+x^2+x+1$	22	6

or, in matrix notation,

$$\begin{pmatrix} \Gamma(\alpha^0) \\ \Gamma(\alpha^{-85}) \\ \Gamma(\alpha^{-170}) \end{pmatrix} = \begin{pmatrix} \gamma_0 & 0 & 0 \\ 0 & \gamma_1 & \gamma_1^2 \\ 0 & \gamma_1^2 & \gamma_1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} \Gamma_0 \\ \Gamma_1 \\ \Gamma_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \left(\begin{pmatrix} 1 \\ \gamma_1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} \Gamma_0 \\ \Gamma_1 \\ \Gamma_2 \end{pmatrix} \right),$$

where the Karatsuba fast 2-point cyclic convolution algorithm was used to compute

$$\begin{pmatrix} \gamma_1 & \gamma_1^2 \\ \gamma_1^2 & \gamma_1 \end{pmatrix} \begin{pmatrix} u_{1,0} \\ u_{1,1} \end{pmatrix} = \begin{pmatrix} (\gamma_1+\gamma_1^2)u_{1,1} + \gamma_1(u_{1,0}+u_{1,1}) \\ (\gamma_1+\gamma_1^2)u_{1,0} + \gamma_1(u_{1,0}+u_{1,1}) \end{pmatrix},$$

together with the identity $\gamma_1+\gamma_1^2=1$. Evaluating $\Gamma(x)$ at the given points requires only six summations (i.e., XOR operations) and one multiplication by γ_1. Comparing this algorithm with the straightforward implementation given by Equations (31) through (33), one can see that the proposed approach enables one to save three multiplications, which are quite expensive operations.

Table II presents the sets $\Phi=\Phi_r$ for the case of $m=8$ and different values of the number of check symbols r as well as the complexity of the corresponding algorithms $\tilde{\mathcal{F}}_{\Phi,r}(\Gamma)$.

3.4.3. Deferred Forney Scaling. Typically, disk arrays operate in a normal state, and parity symbols are not used. To reduce the encoding complexity, one can avoid multiplication by β_i in Equation (30) and compute parity symbols as[5] $p_i'=\Gamma_c(\alpha^{-\phi_i})$. If some disks fail and if erasure recovery is needed, one can reconstruct the original RS check symbols as $p_i=\beta_i p_i'$ to improve the RAID performance in the normal state at the cost of a small decrease in the rebuild performance.

3.4.4. Shortened Reed-Solomon Codes. The previous derivations were performed for the case of codes of length $N=2^m-1$. However, this constraint is not necessary. One can simply set arbitrary $N-n$ codeword symbols to zero and consider them to be information symbols in the previously described encoding method. Obviously, codeword symbols that are always zero do not need to be stored anywhere, resulting in an $(n,k=n-r,r+1)$ shortened Reed-Solomon code that still possesses the MDS property. It is convenient to select the set of actual information symbols $\Psi=\{\psi_0,\ldots,\psi_{k-1}\}$ as the k smallest nonnegative integers not in Φ.

Figure 2 illustrates the structure of the codeword obtained using the proposed approach. Given a "virtual" codeword of length $N=2^m-1$, one assigns parity symbols to r positions given by the set $\Phi, r=n-k$ so that an FFT can be used to compute the values of the check symbols. k information symbols are assigned to the leftmost positions not occupied by parity symbols. The remaining $N-n$ positions are not used (they are filled with zeroes). The actual mapping of codeword symbols to disks can be arbitrary.

[5]Formally, this results in a generalized Reed-Solomon code.

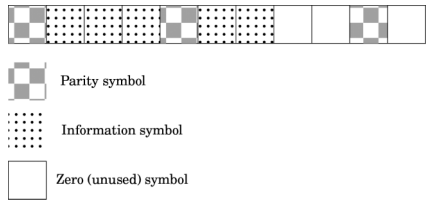

Fig. 2. Virtual codeword structure.

3.4.5. The Algorithm. Summarizing the previous discussion, one obtains the following algorithm for encoding the information vector $(w_0, \ldots, w_{k-1})$ with an $(n, k, n-k+1)$ Reed-Solomon code (with deferred Forney scaling):

(1) Compute $S_i = \sum_{j=0}^{k-1} w_j \alpha^{\psi_j i}, i = 0, \ldots, r-1, \psi_j \in \Psi$, using a truncated cyclotomic FFT algorithm as $S = \mathcal{F}_{r,\Psi}(W)$, where W is a vector that has w_j at positions ψ_j and 0 elsewhere.
(2) Calculate the check symbol evaluator polynomial $\Gamma_c(x) \equiv \Lambda_c(x) S(x) \bmod x^r$.
(3) Compute the check symbols as $p_i = \Gamma_c(\alpha^{-\phi_i}), 0 \le i < r$, using the truncated cyclotomic FFT algorithm as $p = \tilde{\mathcal{F}}_{\tilde{\Phi},r}(\Gamma_c)$, where Γ_c is the vector of coefficients of $\Gamma_c(x)$ and $\tilde{\Phi} = \{-j \bmod N | j \in \Phi\}$.
(4) Construct codewords as $(w_0, \ldots, w_{k-1}, p_0, \ldots, p_{r-1})$.

Here, $r = n - k$ is the number of check symbols to be computed; Φ is the set of the r positions of the check symbols within the virtual codeword, which should be taken as a union of cyclotomic cosets; and Ψ is the set of the k smallest integers not in Φ.

Observe that the second step can be represented as multiplication of the vector S by a binary matrix given by coefficients of $\Lambda_c(x)$ (see Equation (17)). The first step in $\tilde{\mathcal{F}}_{\Phi,r}$ is multiplication by the truncated binary matrix V. It is possible to reduce the complexity of the encoding operation further by premultiplying these matrices at design time so that steps 2 and 3 are combined.

3.5. Partial Stripe Update

Consider the scenario in which the information symbols in positions $j_0, \ldots, j_0 + k' - 1$ should be updated. This can be implemented by computing the difference between the new symbols w'_j and the old symbols w_j as $\delta_j = w'_j - w_j, j_0 \le j < j_0 + k'$, and calculating the check symbols $\rho_0, \ldots, \rho_{n-k-1}$ for the information vector $\Delta = (0, \ldots, 0, \delta_{j_0}, \ldots, \delta_{j_0+k'-1}, 0, \ldots, 0)$ using the algorithm described in Section 3.4. Finally, new values of the check symbols can be obtained as $p'_i = p_i + \rho_i, 0 \le i < n-k$, where p_i are the values of the check symbols corresponding to the old values of the information symbols $w_j, 0 \le j < k$.

However, the cyclotomic FFT algorithm used to compute the syndrome in Equation (28) for the case of full stripe encoding may not be efficient when only a fraction of the information symbols (i.e., correction values δ_j) are different from zero. Obviously, generating appropriate partial FFT algorithms for all possible combinations of j_0 and k' is not practical. If the set of information symbol positions $\Psi = \Psi^{(k)}$ consists of the k smallest distinct nonnegative integers not used for the check symbols (i.e., not in Φ), as described earlier, a much smaller number of distinct partial FFT algorithms $\mathcal{F}_{r,k} = \mathcal{F}_{r,\Psi^{(k)}}$ is needed. Let $\psi : \{0, \ldots, k-1\} \to \Psi$ be the corresponding mapping, that

is, $\psi(j) = \psi_j$. In this case, the syndrome of Δ is given by

$$S_i = \alpha^{i\psi_{j_0}} \sum_{j=0}^{\psi_{k'-1}-\psi_{j_0}} \tilde{\delta}_j \alpha^{ji}, \tag{34}$$

where $\tilde{\delta}_j = \begin{cases} 0, & \text{if } \psi_{j_0} + j \notin \Psi \\ \delta_{\psi^{-1}(\psi_{j_0}+j)}, & \text{otherwise.} \end{cases}$ Equation (34) can be evaluated by applying the partial FFT algorithm $\mathcal{F}_{r,\psi_{k'-1}-\psi_{j_0}+1}(\tilde{\Delta})$ to the vector $\tilde{\Delta} = (\tilde{\delta}_j, \ldots, \tilde{\delta}_{\psi_{k'-1}-\psi_{j_0}})$ and performing $n-k$ additional multiplications in $GF(2^m)$. This approach requires one to derive partial FFT algorithms $\mathcal{F}_{r,k'}(\tilde{\Delta})$ for only up to $\psi(k-1)+1$ different values of k'.

For small values of k', it may be more efficient to perform encoding of Δ by explicitly multiplying it by the corresponding rows of the code generator matrix. For k' close to k, one may fetch the symbols that will not be updated from disks, combine them with the new symbols into a single vector, and apply to the resulting vector the full stripe encoding procedure described previously.

3.6. Rebuild of Failed Disks

If $t \leq r$ of n disks fail, one can use the algorithm presented in Section 2.4 to recover the erased codeword symbols. As shown earlier, $\deg \Gamma(x) < t$; therefore, one can use Equation (19) instead of Equation (16), implying that only[6] $S_0, \ldots, S_{t-1}$ need to be evaluated. The truncated cyclotomic FFT algorithm $\mathcal{F}_{t,\Psi\cup\Phi}$ can be used to compute them. Because the positions of the erased symbols are not likely to correspond to the nice algebraic structures considered earlier, it may be beneficial to use FFTs only for syndrome evaluation and recover the erased symbols as $\widetilde{c} = TS$, where the matrix T is defined in Equation (20).

Remark 3.8. Observe that in the most common case of $t = 1$, one obtains S_0 as a sum of all nonerased symbols without any multiplications. Furthermore, in this case, $\Gamma(x) = S_0$, and $\omega_i = 1$; therefore, the value of the erased symbol is simply S_0.

3.7. Summary of the Proposed Approach

The proposed software implementation of RAID based on Reed-Solomon codes consists of the following components:

(1) A library of partial cyclotomic FFT algorithms $\mathcal{F}_{r,\{0,\ldots,\psi(k-1)\}}(f), 1 \leq r \leq r_{max}, 2 \leq k \leq k_{max}$, where r_{max} and k_{max} are the maximum supported number of parity and payload disks, respectively.
(2) A library of partial cyclotomic FFT algorithms $\tilde{\mathcal{F}}_{\tilde{\Phi}_r,r}(\Gamma)$, where $\tilde{\Phi}_r = \{-i \bmod N | i \in \Phi_r\}$ and Φ_r is the set of positions of r check symbols within the virtual codeword.
(3) A function for encoding the full stripe of k information symbols, which employs $\mathcal{F}_{r,\{0,\ldots,\psi(k-1)\}}((w_0, \ldots, w_{k-1}))$ to compute $S_0, \ldots, S_{r-1}$, Equation (16) to compute $\Gamma_c(x)$, and $\tilde{\mathcal{F}}_{\tilde{\Phi}_r,r}(\Gamma_c)$ to compute r check symbols via Equation (30).
(4) A function for updating k' symbols at the offset j_0 within a stripe, which utilizes $\mathcal{F}_{r,\{0,\ldots,\psi(k'-1)-\psi_{j_0}\}}((w_{j_0}, \ldots, w_{j_0+k'-1}))$ to evaluate Equation (34).
(5) A function for rebuilding $t \leq r$ failed disks, which utilizes $\mathcal{F}_{t,\{0,\ldots,\psi(k-1)\}}((w_0, \ldots, w_{k-1}))$ to compute Equation (14) and Equations (16) to (18) to obtain the erased codeword symbols.

[6]If one needs to minimize the number of disk accesses, one can ignore $n-k-t$ surviving disks.

Table III. Full Stripe Encoding Cost

Configuration		Number of XORs		Number of $GF(2^8)$ Multiplications		Cauchy MDS Array Codes, XORs per Byte [Plank 2005a]
k	r	Total	Per Byte	Total	Per Byte	
4	2	11	2.75	9	2.25	2.5
16	3	54	3.38	11	0.69	6.425
32	4	151	4.72	26	0.81	13.075
48	5	223	4.65	25	0.52	17.32
62	6	366	5.9	39	0.63	-

3.8. Implementation Issues

Implementing Reed-Solomon encoding and decoding requires one to be able to perform arithmetic in $GF(2^m)$. In this section, the most important case in practice of $m = 8$ is considered. The proposed encoding and erasure recovery algorithms reduce to multiplying the data values by some constants, which either are hard coded (in the case of cyclotomic FFT) or depend only on the particular erasure pattern (Equations (19) and (18)), which rarely changes. Hence, one can implement multiplications via table lookup. Modern CPUs provide efficient hardware support for small table lookup, enabling one to perform the multiplication of $x \in GF(2^8)$ by a fixed constant $a \in GF(2^8)$ as follows (see Bhaskar et al. [2003] and Plank et al. [2013] and references therein). Let $x = \sum_{i=0}^{7} x_i \alpha^i, x_i \in GF(2)$ be the standard basis decomposition of x, where α is a primitive element. Let $x' = \sum_{i=0}^{3} x_i \alpha^i, x'' = \sum_{i=4}^{7} x_i \alpha^i$. Then, $y = ax = ax' + ax''$, and the summands in this expression can be obtained by looking up two 16-byte precomputed tables containing the values of ax' and ax'' for all 16 possible values of x' and x''. Modern CPUs provide support for SIMD table lookup to enable one to simultaneously multiply by a fixed scalar a all components of the vector $X \in GF(2^8)^l$, where l is the CPU register size (in bytes).

For example, this approach can be implemented using Intel SSE2 intrinsics as follows:

```
const __m128i Mask0F=_mm_set1_epi8(0x0F);
__m128i X0=_mm_and_si128(X,&Mask0F); //extract the first half-byte
__m128i X1=_mm_srli_epi16(X,4); //extract the second half-byte
X1=_mm_and_si128(X1,*pMask0F);
X0=_mm_shuffle_epi8(pHelper[a].Lookup0,X0);//multiply half-bytes by a
X1=_mm_shuffle_epi8(pHelper[a].Lookup1,X1);
Dest=_mm_xor_si128(X0,X1); //add them
```

Here, $pHelper[a].Lookup0$ and $pHelper[a].Lookup1$ are 128-bit variables containing lookup tables for ax' and ax'', respectively. Multiplication requires six instructions (which may cause cache misses while accessing lookup tables), whereas summation can be implemented via a single XOR instruction, justifying the need to minimize the number of multiplications.

4. PERFORMANCE EVALUATION

Because the cyclotomic FFT algorithm relies on a computer-optimized algorithm for multiplication by the A matrix, it is not easy to derive an exact analytical complexity estimate for the general case. In this section, the number of operations for particular instances of the algorithms considered is provided.

Table III presents the number of operations needed for full stripe encoding in some RAID configurations. Here, k and r denote the number of payload and parity disks, respectively. The proposed approach requires approximately r XOR operations and approximately one multiplication per one payload symbol (byte). Recall that the standard encoding algorithm based on Equation (3) requires r XOR and r multiplication

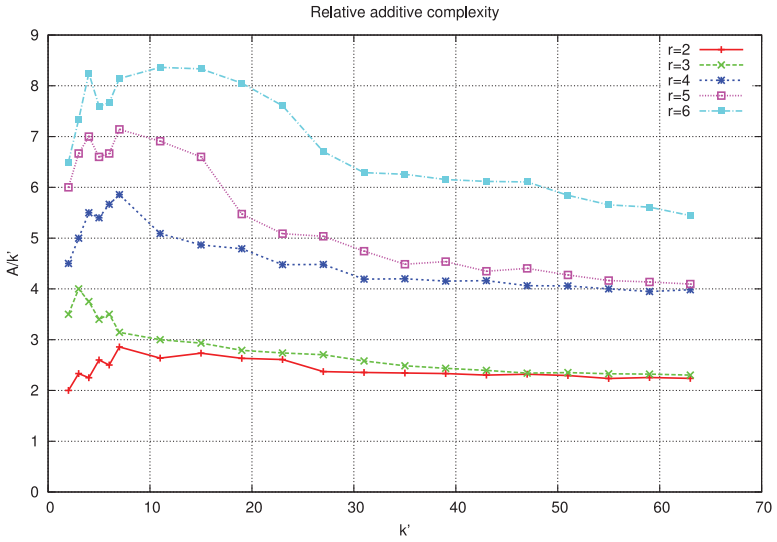

Fig. 3. Relative additive complexity of syndrome evaluation.

operations per data symbol. Because a Galois field multiplication is much more expensive compared to XOR, the FFT-based method can provide a significant performance improvement compared with the straightforward implementation.

Figure 3 presents the relative additive complexity (i.e., the number of XOR operations divided by k', the number of symbols being updated) of $\mathcal{F}_{r,\{0,\ldots,k'-1\}}(w)$, which is used for the partial stripe update.[7] For very small k', the relative number of operations needed by the update operation increases by up to 50% compared with the case of full stripe encoding. However, the number of multiplications is essentially independent of k', as shown in Figure 4.

The proposed algorithms were implemented in software, and their performance was studied using a RAID simulator, which represents the disks as ordinary memory-mapped files. The SIMD techniques described in Section 3.8 were used to implement the Galois field arithmetic. The measurements were performed on an Intel Core-i7 3930K CPU, and the stripe unit size λ was set to 512 bytes. Figure 5 presents a full stripe encoding bandwidth of Reed-Solomon codes with different encoding algorithms. FFT-based syndrome evaluation (see Section 3.3.2) provides up to three times the performance improvement compared to the standard encoding algorithm. The optimized selection of check symbols (see Section 3.4.2) enables additional speedup by a factor of 2. The deferred Forney scaling increases the encoding bandwidth by approximately 4%, except in the case of $r|255$, where the optimized allocation of check symbols causes the Forney factors to be equal to 1. The optimal performance is achieved in the case of $k = 48$, that is, a stripe size of 24,576 bytes. This optimal performance is in good agreement with the level 1 data cache size of the benchmarking PC, which is equal to 32,768 bytes.

Table IV presents a comparison of the performance of the developed implementation with the one given in the Jerasure 2.0 library [Plank 1997; Plank and Greenan 2014]. Here, the stripe unit size was set to $\lambda = 4{,}096$ bytes. Employing the proposed encoding algorithm provides up to 3.6 times the speedup, and the most significant gain is achieved in the case of high k.

[7]Recall that the complexity of computing Equations (29) to (30) does not depend on the input size.

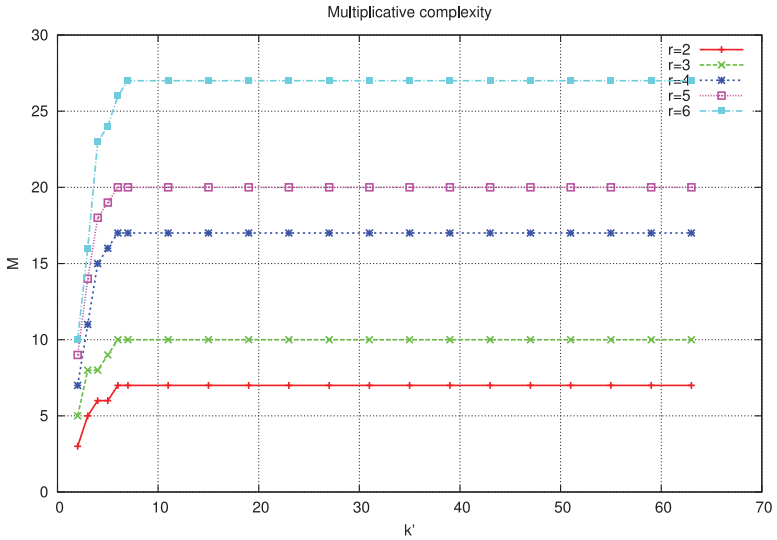

Fig. 4. Multiplicative complexity of syndrome evaluation.

Table IV. Encoding Performance, GB/s

k	$r = n - k$	Jerasure 2.0	Proposed Approach
9	3	4.7	10.3
16	3	4.9	14.1
30	5	2.8	10.0
10	6	2.3	3.3
10	8	1.7	2.2
20	11	1.2	2.5

5. COMPARISON WITH OTHER CODES

It was suggested in Blomer et al. [1995] to construct an (n, k, d) MDS array code by replacing the elements of $GF(2^m)$ in the $k \times r$ Cauchy matrix A (see Equation (3)) with the corresponding $m \times m$ binary matrices. The complexity of encoding mk data units with these codes is given by $w - m(n - k)$, where w is the number of 1s in the binary matrix that is obtained. The structure of the Cauchy matrices was further optimized in Plank [2005b] and Blomer et al. [1995] to minimize their weight w. Table III also presents the normalized encoding cost for these codes. One can see that the cost is much higher than the complexity of the proposed method for the Reed-Solomon codes, except for very small k and r.

An important advantage of the Reed-Solomon codes with respect to array codes, such as EvenOdd [Blaum et al. 1995], RDP [Corbett et al. 2004], and their extensions to three or more parity symbols, is that in general, RS codes require a smaller stripe size. Indeed, in the case of array codes, each symbol consists of m bits (or stripe units), and m must satisfy certain constraints (e.g., $m + 1$ must be prime and must be greater than the number of payload disks k). Hence, the total payload stripe size is $mk\lambda$, where λ is the size of one stripe unit. Furthermore, different bits are processed in different ways, causing the exploitation of the SIMD features of modern CPUs in the implementation of the corresponding algorithms to be a challenging problem, and, in general, requires one to use a sufficiently large λ. For RS codes, each symbol consists of m bits (1 byte for $m = 8$), and any code of length $n < 2^m$ and dimension $k < n$ is supported. For $m = 8$,

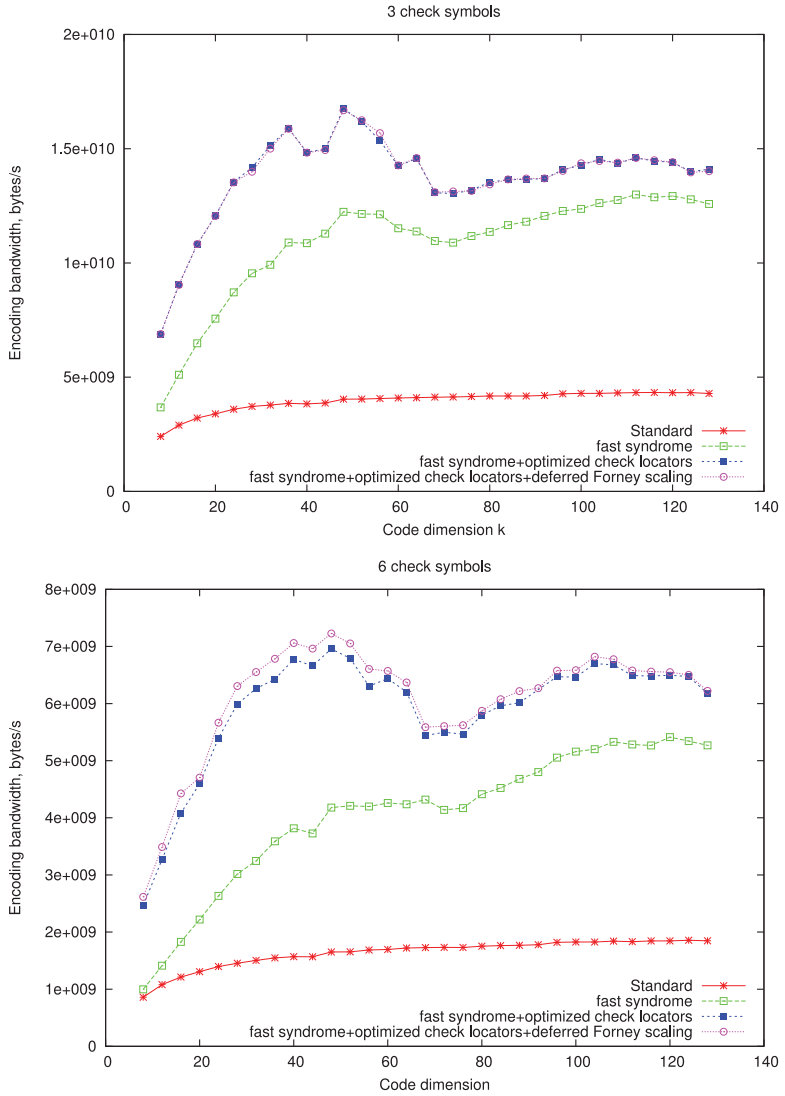

Fig. 5. Full stripe encoding bandwidth with Reed-Solomon codes.

the payload stripe size is $k\lambda$ bytes, and the maximum number of allowed values for the number of payload disks k far exceeds today's practical requirements. Because the applications tend to write data in relatively small blocks, a smaller stripe size results in a smaller probability of an expensive partial stripe update.

A major problem in the design of RAID with multiple parity disks is minimizing the rebuild ratio ρ, that is, the fraction of surviving disks that need to be accessed while repairing t failed ones. A disadvantage of the Reed-Solomon codes is that their rebuild ratio is given by $\rho_{RS} = \frac{n-k}{n-t}$. It is possible to achieve a much lower rebuild ratio of $\rho_z = \frac{t}{n-k}$ by employing zigzag codes [Tamo et al. 2011]. These codes, however, require performing multiplications in $GF(q), q > 2$, in general. No fast algorithms for encoding and decoding these codes are currently available.

The Reed-Solomon codes can be used in the construction of pyramid codes [Huang et al. 2007], which provide a tradeoff between storage and access efficiency. Hence, the algorithms proposed in this article can be used to improve the efficiency of systems employing pyramid codes.

6. CONCLUSIONS

In this article, it was shown that by exploiting the algebraic structure of the Reed-Solomon codes and the cyclotomic FFT algorithm, it is possible to construct a very efficient implementation of RAID with a complexity much lower than that of codes based on Cauchy MDS array codes. The Reed-Solomon codes enable one to obtain any number of parity check symbols easily, thus allowing for the construction of very large disk arrays. This is an important advantage with respect to many of the existing MDS array codes, which do not provide simple methods for constructing very large disk arrays. The proposed approach can be extended to other codes used in storage systems (e.g., pyramid codes) that employ Reed-Solomon codes as a building block. Other codes with strong algebraic structures may exist that admit employing a similar approach for the construction of efficient encoding and erasure repair algorithms, strongly motivating employing such codes in high-performance storage systems.

APPENDIX

This appendix presents fast m_s-point cyclic convolution algorithms for computing

$$\sum_{i=0}^{m_s-1} b_{s,m_s-i}x^i \equiv \left(\sum_{i=0}^{m_s-1} g_{s,i}x^i\right)\left(\sum_{i=0}^{m_s-1} u_{s,i}x^i\right) \bmod x^{m_s}-1$$

for some values of m_s, which may be used often in the implementation of the proposed approach.

The 2-point cyclic convolution is given by Blahut [1984]

$$\begin{pmatrix} b_{s,2} \\ b_{s,1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}\left(\left(\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} g_{s,0} \\ g_{s,1} \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} u_{s,0} \\ u_{s,1} \end{pmatrix}\right)\right).$$

The 4-point cyclic convolution is given by (see the Russian translation of Blahut [1984])

$$\begin{pmatrix} b_{s,4} \\ b_{s,1} \\ b_{s,2} \\ b_{s,3} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}\left(\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}\begin{pmatrix} g_{s,0} \\ g_{s,1} \\ g_{s,2} \\ g_{s,3} \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}\begin{pmatrix} u_{s,0} \\ u_{s,1} \\ u_{s,2} \\ u_{s,3} \end{pmatrix}\right).$$

The 8-point cyclic convolution is given by Bellini et al. [2011]

$$\begin{pmatrix} b_{s,8} \\ b_{s,1} \\ b_{s,2} \\ b_{s,3} \\ b_{s,4} \\ b_{s,5} \\ b_{s,6} \\ b_{s,7} \end{pmatrix} = \begin{pmatrix} 1&1&1&1&1&1&1&1 \\ 1&0&1&0&1&0&1&0 \\ 0&1&0&1&0&1&0&1 \\ 1&0&0&1&1&0&0&1 \\ 1&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&1 \\ 1&0&0&0&0&1&1&1 \\ 1&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0&1 \\ 1&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \\ 0&1&1&0&0&1&1&0 \\ 0&0&1&0&0&0&1&0 \\ 0&1&0&0&0&1&0&0 \\ 0&1&1&1&1&0&0&0 \\ 0&0&1&0&1&0&0&0 \\ 0&1&0&1&0&0&0&0 \\ 0&0&0&1&1&1&1&0 \\ 0&0&0&0&1&0&1&0 \\ 0&0&0&1&0&1&0&0 \\ 0&1&1&1&1&1&1&0 \\ 0&0&1&0&1&0&1&0 \\ 0&1&0&1&0&1&0&0 \end{pmatrix}^T \left(\begin{pmatrix} 1&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&1 \\ 1&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&1&0 \\ 1&0&0&0&0&1&1&1 \\ 1&1&0&0&0&0&1&1 \\ 1&0&0&0&1&0&0&0 \\ 1&0&0&1&1&0&0&1 \\ 1&1&0&0&1&1&0&0 \\ 1&0&1&0&1&0&1&0 \\ 1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&1&0&1&0 \\ 0&0&0&1&1&1&1&0 \\ 0&0&0&0&1&1&1&1 \\ 0&0&1&0&0&0&1&0 \\ 0&1&1&0&0&1&1&0 \\ 0&0&1&1&0&0&1&1 \\ 1&0&1&0&1&0&1&0 \\ 1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1 \\ 1&0&1&0&1&0&1&0 \\ 1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1 \end{pmatrix} \cdot \begin{pmatrix} g_{s,0} \\ g_{s,1} \\ g_{s,2} \\ g_{s,3} \\ g_{s,4} \\ g_{s,5} \\ g_{s,6} \\ g_{s,7} \end{pmatrix} \cdot \begin{pmatrix} 1&1&1&1&1&1&1&1 \\ 0&1&0&1&0&1&0&1 \\ 1&0&1&0&1&0&1&0 \\ 0&0&1&1&0&0&1&1 \\ 0&0&0&1&0&0&0&1 \\ 0&0&1&0&0&0&1&0 \\ 0&0&0&0&1&1&1&1 \\ 0&0&0&0&0&1&0&1 \\ 0&0&0&0&1&0&1&0 \\ 0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&0 \\ 1&1&0&0&1&1&0&0 \\ 0&1&0&0&0&1&0&0 \\ 1&0&0&0&1&0&0&0 \\ 1&1&1&1&0&0&0&0 \\ 0&1&0&1&0&0&0&0 \\ 1&0&1&0&0&0&0&0 \\ 0&0&1&1&1&1&0&0 \\ 0&0&0&1&0&1&0&0 \\ 0&0&1&0&1&0&0&0 \\ 1&1&1&1&1&1&0&0 \\ 0&1&0&1&0&1&0&0 \\ 1&0&1&0&1&0&0&0 \end{pmatrix} \cdot \begin{pmatrix} u_{s,0} \\ u_{s,1} \\ u_{s,2} \\ u_{s,3} \\ u_{s,4} \\ u_{s,5} \\ u_{s,6} \\ u_{s,7} \end{pmatrix} \right).$$

ACKNOWLEDGMENT

The author is indebted to the anonymous reviewers for their helpful comments, which have greatly improved the quality of the article. The author would like to thank Inga Petryaevskaya, Artem Aliev, and Alexander Alexeev for many stimulating discussions and the EMC Corporation for supporting this research.

REFERENCES

A. Al Ghouwayel, Y. Louet, A. Nafkha, and J. Palicot. 2007. On the FPGA implementation of the Fourier transform over finite fields $GF(2^m)$. In *Proceedings of the International Symposium on Communications and Information Technologies (ISCIT'07).* 146–151.

S. Bellini, M. Ferrari, and A. Tomasoni. 2011. On the structure of cyclotomic Fourier transforms and their applications to Reed-Solomon codes. *IEEE Transactions on Communications* 59, 8, 2110–2118.

R. Bhaskar, P. Dubey, V. Kumar, A. Rudra, and A. Sharma. 2003. Efficient Galois field arithmetic on SIMD architectures. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures*. 256–257.

R. Blahut. 1984. *Theory and Practice of Error Control Codes*. Addison-Wesley.

M. Blaum, J. Brady, J. Bruck, and J. Menon. 1995. EVENODD: An optimal scheme for tolerating double disk failures in RAID architectures. *IEEE Transactions on Computers* 44, 2.

J. Blomer, M. Kalfanej, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. 1995. *An XOR-Based Erasure-Resilient Coding Scheme*. Tech. Rep., International Computer Science Institute, Berkeley, California.

N. Chen and Z. Yan. 2009a. Cyclotomic FFTs with reduced additive complexities based on a novel common subexpression elimination algorithm. *IEEE Transactions on Signal Processing* 57, 3, 1010–1020.

N. Chen and Z. Yan. 2009b. Reduced-complexity Reed-Solomon decoders based on cyclotomic FFTs. *IEEE Signal Processing Letters* 16, 4, 279–282.

E. Chu and A. George. 2000. *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*. CRC Press.

P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. 2004. Row-diagonal parity for double disk failure correction. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*. USENIX Association, Berkeley, CA, 1–14.

E. Costa, S. V. Fedorenko, and P. V. Trifonov. 2004. On computing the syndrome polynomial in Reed-Solomon decoder. *European Transactions on Telecommunications* 15, 4, 337–342.

F. Didier. 2009. Efficient erasure decoding of reed-solomon codes. arXiv:1301.5536v1.

S. Fedorenko. 2006. A method for computation of the discrete Fourier transform over a finite field. *Problems of Information Transmission* 42, 2, 139–151.

S. Fedorenko. 2011a. The discrete Fourier transform over a finite field with reduced multiplicative complexity. In *Proceedings of IEEE International Symposium on Information Theory*. 1200–1204.

S. V. Fedorenko. 2011b. A novel method for computation of the discrete Fourier transform over characteristic two finite field of even extension degree. *IEEE Transactions on Information Theory*. Available at http://arxiv.org/abs/1112.1639.

P.-H. Hsieh, I.-Y. Chen, Y.-T. Lin, and S.-Y. Kuo. 2004. An XOR based Reed-Solomon algorithm for advanced RAID systems. In *Proceedings of 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*.

C. Huang, M. Chen, and J. Li. 2007. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. In *Proceedings of 6th IEEE International Symposium on Network Computing and Applications*. 79–86.

D. E. Knuth. 1973. *The Art of Computer Programming*. Vol. 2. Addison-Wesley.

R. Lidl and H. Niederreiter. 1996. *Finite Fields*. Cambridge University Press.

T.-C. Lin, T. Truong, and P. Chen. 2007. A fast algorithm for the syndrome calculation in algebraic decoding of Reed-Solomon codes. *IEEE Transactions on Communications* 55, 12, 2240–2244.

F. J. MacWilliams and N. J. A. Sloane. 1977. *The Theory of Error-Correcting Codes*. North-Holland Publishing Company.

T. K. Moon. 2005. *Error Correction Coding. Mathematical Methods and Algorithms*. Wiley.

J. S. Plank. 1997. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice and Experience* 27, 9, 995–1012.

J. S. Plank. 2005a. *Enumeration of Optimal and Good Cauchy Matrices for Reed-Solomon Coding*. Tech. Rep. CS-05-570, University of Tennessee.

J. S. Plank. 2005b. *Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Storage Applications*. Tech. Rep. CS-05-569, University of Tennessee.

J. S. Plank and K. M. Greenan. 2014. *Jerasure: A Library in C Facilitating Erasure Coding for Storage Applications – version 2.0*. Tech. Rep. UT-EECS-14-721, University of Tennessee.

J. S. Plank, K. M. Greenan, and E. L. Miller. 2013. Screaming fast Galois field arithmetic using intel SIMD instructions. In *Proceedings of 11th Usenix Conference on File and Storage Technologies*.

A. Soro and J. Lacan. 2010. FNT-based Reed-Solomon erasure codes. In *7th IEEE Consumer Communications and Networking Conference*.

I. Tamo, Z. Wang, and J. Bruck. 2011. MDS array codes with optimal rebuilding. In *Proceedings of IEEE International Symposium on Information Theory*. 1240–1244.

P. Trifonov. 2007. Matrix-vector multiplication via erasure decoding. In *Proceedings of the 11th International Symposium on Problems of Redundancy in Information and Control Systems*.

P. Trifonov. 2012. On the additive complexity of the cyclotomic FFT algorithm. In *Proceedings of IEEE Information Theory Workshop*.

P. V. Trifonov and S. V. Fedorenko. 2003. A method for fast computation of the Fourier transform over a finite field. *Problems of Information Transmission* 39, 3, 231–238. Translation of Problemy Peredachi Informatsii.

X. Wu, M. Wagh, N. Chen, Y. Wang, and Z. Yan. 2011. Composite cyclotomic Fourier transforms with reduced complexities. *IEEE Transactions on Signal Processing* 59, 5, 2136–2145.

X. Wu, Y. Wang, and Z. Yan. 2012. On algorithms and complexities of cyclotomic fast Fourier transforms over arbitrary finite fields. *IEEE Transactions on Signal Processing* 60, 3, 1149–1158.

X. Wu and Z. Yan. 2011. Computational complexity of cyclotomic fast Fourier transforms over characteristic-2 fields. In *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS'11)*. 1–6.

L. Xu and J. Bruck. 1999. X-code: MDS array codes with optimal encoding. *IEEE Transactions on Information Theory* 45, 1.

Received November 2013; revised March 2014; accepted September 2014