Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2008, Article ID 843634, 11 pages
doi:10.1155/2008/843634

Research Article

Complexity Analysis of Reed-Solomon Decoding over GF(2^m) without Using Syndromes

Ning Chen and Zhiyuan Yan

Department of Electrical and Computer Engineering, Lehigh University, Bethlehem, PA 18015, USA

Correspondence should be addressed to Zhiyuan Yan, yan@lehigh.edu

Received 15 November 2007; Revised 29 March 2008; Accepted 6 May 2008

Recommended by Jinhong Yuan

There has been renewed interest in decoding Reed-Solomon (RS) codes without using syndromes recently. In this paper, we investigate the complexity of syndromeless decoding, and compare it to that of syndrome-based decoding. Aiming to provide guidelines to practical applications, our complexity analysis focuses on RS codes over characteristic-2 fields, for which some *multiplicative* FFT techniques are not applicable. Due to moderate block lengths of RS codes in practice, our analysis is complete, without big O notation. In addition to fast implementation using *additive* FFT techniques, we also consider direct implementation, which is still relevant for RS codes with moderate lengths. For high-rate RS codes, when compared to syndrome-based decoding algorithms, not only syndromeless decoding algorithms require more field operations regardless of implementation, but also decoder architectures based on their direct implementations have higher hardware costs and lower throughput. We also derive tighter bounds on the complexities of fast polynomial multiplications based on Cantor's approach and the fast extended Euclidean algorithm.

1. INTRODUCTION

Reed-Solomon (RS) codes are among the most widely used error control codes, with applications in space communications, wireless communications, and consumer electronics [1]. As such, efficient decoding of RS codes is of great interest. The majority of the applications of RS codes use syndrome-based decoding algorithms such as the Berlekamp-Massey algorithm (BMA) [2] or the extended Euclidean algorithm (EEA) [3]. Alternative hard decision decoding methods for RS codes without using syndromes were considered in [4–6]. As pointed out in [7, 8], these algorithms belong to the class of frequency-domain algorithms and are related to the Welch-Berlekamp algorithm [9]. In contrast to syndrome-based decoding algorithms, these algorithms do not compute syndromes and avoid the Chien search and Forney's formula. Clearly, this difference leads to the question whether these algorithms offer lower complexity than syndrome-based decoding, especially when fast Fourier transform (FFT) techniques are applied [6].

Asymptotic complexity of syndromeless decoding was analyzed in [6], and in [7] it was concluded that syndromeless decoding has the same asymptotic complexity $O(n \log^2 n)$ (note that all the logarithms in this paper are to base two) as syndrome-based decoding [10]. However, existing asymptotic complexity analysis is limited in several aspects. For example, for RS codes over Fermat fields GF($2^{2^r} + 1$) and other prime fields [5, 6], efficient multiplicative FFT techniques lead to an asymptotic complexity of $O(n \log^2 n)$. However, such FFT techniques do not apply to characteristic-2 fields, and hence this complexity is not applicable to RS codes over characteristic-2 fields. For RS codes over arbitrary fields, the asymptotic complexity of syndromeless decoding based on multiplicative FFT techniques was shown to be $O(n \log^2 n \log\log n)$ [6]. Although they are applicable to RS codes over characteristic-2 fields, the complexity has large coefficients and multiplicative FFT techniques are less efficient than fast implementation based on additive FFT for RS codes with moderate block lengths [6, 11, 12]. As such, asymptotic complexity analysis provides little help to practical applications.

In this paper, we analyze the complexity of syndromeless decoding and compare it to that of syndrome-based decoding. Aiming to provide guidelines to system designers, we focus on the decoding complexity of RS codes over GF(2^m). Since RS codes in practice have moderate lengths, our complexity analysis provides not only the coefficients for the most significant terms, but also the following terms. Due to their moderate lengths, our comparison is based on two types of implementations of syndromeless decoding and syndrome-based decoding: direct implementation and fast implementation based on FFT techniques. Direct implementations are often efficient when decoding RS codes with moderate lengths and have widespread applications; thus, we consider both computational complexities, in terms of field operations, and hardware costs and throughputs. For fast implementations, we consider their computational complexities only and their hardware implementations are beyond the scope of this paper. We use *additive* FFT techniques based on Cantor's approach [13] since this approach achieves small coefficients [6, 11] and hence is more suitable for moderate lengths. In contrast to some previous works [12, 14], which count field multiplications and additions together, we differentiate the multiplicative and additive complexities in our analysis.

The main contributions of the papers are as follows.

(i) We derived a tighter bound on the complexities of fast polynomial multiplication based on Cantor's approach.

(ii) We also obtained a tighter bound on the complexity of the fast extended Euclidean algorithm (FEEA) for general partial greatest common divisor (GCD) computation.

(iii) We evaluated the complexities of syndromeless decoding based on different implementation approaches and compare them with their counterparts of syndrome-based decoding. Both errors-only and errors-and-erasures decodings are considered.

(iv) We compare the hardware costs and throughputs of direct implementations for syndromeless decoders with those for syndrome-based decoders.

The rest of the paper is organized as follows. To make this paper self-contained, in Section 2 we briefly review FFT algorithms over finite fields, fast algorithms for polynomial multiplication and division over GF(2^m), the FEEA, and syndromeless decoding algorithms. Section 3 presents both computational complexity and decoder architectures of direct implementations of syndromeless decoding, and compares them with their counterparts for syndrome-based decoding algorithms. Section 4 compares the computational complexity of fast implementations of syndromeless decoding with that of syndrome-based decoding. In Section 5, case studies on two RS codes are provided and errors-and-erasures decoding is discussed. The conclusions are given in Section 6.

2. BACKGROUND

2.1. Fast Fourier transform over finite fields

For any n ($n \mid q-1$) distinct elements $a_0, a_1, \ldots, a_{n-1} \in \mathrm{GF}(q)$, the transform from $\mathbf{f} = (f_0, f_1, \ldots, f_{n-1})^T$ to $\mathbf{F} \triangleq (f(a_0), f(a_1), \ldots, f(a_{n-1}))^T$, where $f(x) = \sum_{i=0}^{n-1} f_i x^i \in \mathrm{GF}(q)[x]$, is called a discrete Fourier transform (DFT), denoted by $\mathbf{F} = \mathrm{DFT}(\mathbf{f})$. Accordingly, $\mathbf{f}$ is called the inverse DFT of $\mathbf{F}$, denoted by $\mathbf{f} = \mathrm{IDFT}(\mathbf{F})$. Asymptotically fast Fourier transform (FFT) algorithm over GF(2^m) was proposed in [15]. Reduced-complexity cyclotomic FFT (CFFT) was shown to be efficient for moderate lengths in [16].

2.2. Polynomial multiplication over GF(2^m) by Cantor's approach

A fast polynomial multiplication algorithm using additive FFT was proposed by Cantor [13] for GF(q^{q^m}), where q is prime, and it was generalized to GF(q^m) in [11]. Instead of evaluating and interpolating over the multiplicative subgroups as in multiplicative FFT techniques, Cantor's approach uses additive subgroups. Cantor's approach relies on two algorithms: multipoint evaluation (MPE) [11, Algorithm 3.1] and multipoint interpolation (MPI) [11, Algorithm 3.2].

Suppose the degree of the product of two polynomials over GF(2^m) is less than h ($h \le 2^m$), the product can be obtained as follows. First, the two operand polynomials are evaluated using the MPE algorithm. The evaluation results are then multiplied pointwise. Finally, the product polynomial is obtained by the MPI algorithm to interpolate the pointwise multiplication results. The polynomial multiplication requires at most $(3/2)h \log^2 h + (15/2)h \log h + 8h$ multiplications over GF(2^m) and $(3/2)h \log^2 h + (29/2)h \log h + 4h + 9$ additions over GF(2^m) [11]. For simplicity, henceforth in this paper, all arithmetic operations are over GF(2^m) unless specified otherwise.

2.3. Polynomial division by Newton iteration

Suppose $a, b \in \mathrm{GF}(q)[x]$ are two polynomials of degrees $d_0 + d_1$ and d_1 ($d_0, d_1 \ge 0$), respectively. To find the quotient polynomial q and the remainder polynomial r satisfying $a = qb + r$, where $\deg r < d_1$, a fast polynomial division algorithm is available [12]. Suppose $\mathrm{rev}_h(a) \triangleq x^h a(1/x)$, the fast algorithm first computes the inverse of $\mathrm{rev}_{d_1}(b) \bmod x^{d_0+1}$ by Newton iteration. Then, the reverse quotient is given by $q^* = \mathrm{rev}_{d_0+d_1}(a)\mathrm{rev}_{d_1}(b)^{-1} \bmod x^{d_0+1}$. Finally, the actual quotient and remainder are given by $q = \mathrm{rev}_{d_0}(q^*)$ and $r = a - qb$.

Thus, the complexity of polynomial division with remainder of a polynomial a of degree $d_0 + d_1$ by a monic polynomial b of degree d_1 is at most $4\mathsf{M}(d_0) + \mathsf{M}(d_1) + O(d_1)$ multiplications/additions when $d_1 \ge d_0$ [12, Theorem 9.6], where $\mathsf{M}(h)$ stands for the numbers of multiplications/additions required to multiply two polynomials of degree less than h.

2.4. Fast extended Euclidean algorithm

Let r_0 and r_1 be two monic polynomials with $\deg r_0 > \deg r_1$ and we assume $s_0 = t_1 = 1$, $s_1 = t_0 = 0$. Step i ($i = 1, 2, \ldots, l$) of the EEA computes $\rho_{i+1} r_{i+1} = r_{i-1} - q_i r_i$, $\rho_{i+1} s_{i+1} = s_{i-1} - q_i s_i$, and $\rho_{i+1} t_{i+1} = t_{i-1} - q_i t_i$ so that the sequence r_i are monic polynomials with strictly decreasing degrees. If the GCD of r_0 and r_1 is desired, the EEA terminates when $r_{l+1} = 0$. For $1 \le i \le l$, $R_i \triangleq Q_i \cdots Q_1 R_0$, where $Q_i = \begin{bmatrix} 0 & 1 \\ 1/\rho_{i+1} & -q_i/\rho_{i+1} \end{bmatrix}$ and $R_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Then, it can be easily verified that $R_i = \begin{bmatrix} s_i & t_i \\ s_{i+1} & t_{i+1} \end{bmatrix}$ for $0 \le i \le l$. In RS decoding, the EEA stops when the degree of r_i falls below a certain threshold for the first time, and we refer to this as partial GCD.

The FEEA in [12, 17] costs no more than $(22\mathrm{M}(h) + O(h)) \log h$ multiplications/additions when $n_0 \le 2h$ [14].

2.5. Syndrome-based and syndromeless decoding

Over a finite field GF(q), suppose $a_0, a_1, \ldots, a_{n-1}$ are n ($n \le q$) distinct elements and $g_0(x) \triangleq \prod_{i=0}^{n-1}(x - a_i)$. Let us consider an RS code over GF(q) with length n, dimension k, and minimum Hamming distance $d = n - k + 1$. A message polynomial $m(x)$ of degree less than k is encoded to a codeword $(c_0, c_1, \ldots, c_{n-1})$ with $c_i = m(a_i)$, and the received vector is given by $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$.

The syndrome-based hard decision decoding consists of the following Steps: syndrome computation, key equation solver, the Chien search, and Forney's formula. Further details are omitted, and interested readers are referred to [1, 2, 18]. We also consider the following two syndromeless algorithms.

Algorithm 1. [4, 5], [6, Algorithm 1]

(1.1) Interpolation. Construct a polynomial $g_1(x)$ with $\deg g_1(x) < n$ such that $g_1(a_i) = r_i$ for $i = 0, 1, \ldots, n-1$.

(1.2) Partial GCD. Apply the EEA to $g_0(x)$ and $g_1(x)$, and find $g(x)$ and $v(x)$ that maximize $\deg g(x)$ while satisfying $v(x)g_1(x) \equiv g(x) \bmod g_0(x)$ and $\deg g(x) < (n+k)/2$.

(1.3) Message recovery. If $v(x) \mid g(x)$, the message polynomial is recovered by $m(x) = g(x)/v(x)$, otherwise output "decoding failure."

Algorithm 2. [6, Algorithm 1a]

(2.1) Interpolation. Construct a polynomial $g_1(x)$ with $\deg g_1(x) < n$ such that $g_1(a_i) = r_i$ for $i = 0, 1, \ldots, n-1$.

(2.2) Partial GCD. Find $s_0(x)$ and $s_1(x)$ satisfying $g_0(x) = x^{n-d+1} s_0(x) + r_0(x)$ and $g_1(x) = x^{n-d+1} s_1(x) + r_1(x)$, where $\deg r_0(x) \le n - d$ and $\deg r_1(x) \le n - d$. Apply the EEA to $s_0(x)$ and $s_1(x)$, and stop when the remainder $g(x)$ has degree less than $(d-1)/2$. Thus, we have $v(x)s_1(x) + u(x)s_0(x) = g(x)$.

(2.3) Message recovery. If $v(x) \nmid g_0(x)$, output "decoding failure;" otherwise, first compute $q(x) = g_0(x)/v(x)$, and then obtain $m'(x) = g_1(x) + q(x)u(x)$. If $\deg m'(x) < k$, output $m'(x)$; otherwise output "decoding failure."

Compared with Algorithm 1, the partial GCD Step of Algorithm 2 is simpler but its message recovery Step is more complex [6].

3. DIRECT IMPLEMENTATION OF SYNDROMELESS DECODING

3.1. Complexity analysis

We analyze the complexity of direct implementation of Algorithms 1 and 2. For simplicity, we assume $n - k$ is even and hence $d - 1 = 2t$.

First, $g_1(x)$ in Steps (1.1) and (2.1) is given by IDFT($\mathbf{r}$). Direct implementation of Steps (1.1) and (2.1) follows Horner's rule and requires $n(n-1)$ multiplications and $n(n-1)$ additions [19].

Steps (1.2) and (2.2) both use the EEA. The Sugiyama tower (ST) [3, 20] is well known as an efficient direct implementation of the EEA. For Algorithm 1, the ST is initialized by $g_1(x)$ and $g_0(x)$, whose degrees are at most n. Since the number of iterations is $2t$, Step (1.2) requires $4t(n+2)$ multiplications and $2t(n+1)$ additions. For Algorithm 2, the ST is initialized by $s_0(x)$ and $s_1(x)$, whose degrees are at most $2t$ and the iteration number is at most $2t$.

Step (1.3) requires one polynomial division, which can be implemented by using k iterations of cross multiplications in the ST. Since $v(x)$ is actually the error locator polynomial [6], $\deg v(x) \le t$. Hence, this requires $k(k + 2t + 2)$ multiplications and $k(t+2)$ additions. However, the result of the polynomial division is scaled by a nonzero constant. That is, cross multiplications lead to $\overline{m}(x) = am(x)$. To remove the scaling factor a, we can first compute $1/a = \mathrm{lc}(g(x))/(\mathrm{lc}(\overline{m}(x))\mathrm{lc}(v(x)))$, where $\mathrm{lc}(f)$ denotes the leading coefficient of a polynomial f, and then obtains $m(x) = (1/a)\overline{m}(x)$. This process requires one inversion and $k+2$ multiplications.

Step (2.3) involves one polynomial division, one polynomial multiplication, and one polynomial addition, and their complexities depend on the degrees of $v(x)$ and $u(x)$, denoted as d_v and d_u, respectively. In the polynomial division, let the result of the ST be $\overline{q}(x) = aq(x)$. The scaling factor is recovered by $1/a = 1/(\mathrm{lc}(\overline{q}(x))\mathrm{lc}(v(x)))$. Thus, it requires one inversion, $(n - d_v + 1)(n + d_v + 3) + n - d_v + 2$ multiplications, and $(n - d_v + 1)(d_v + 2)$ additions to obtain $q(x)$. The polynomial multiplication needs $(n - d_v + 1)(d_u + 1)$ multiplications and $(n - d_v + 1)(d_u + 1) - (n - d_v + d_u + 1)$ additions, and the polynomial addition needs n additions since $g_1(x)$ has degree at most $n-1$. The total complexity of Step (2.3) includes $(n - d_v + 1)(n + d_v + d_u + 5) + 1$ multiplications, $(n - d_v + 1)(d_v + d_u + 2) + n - d_u$ additions, and one inversion. Consider the worst case for multiplicative complexity, where d_v should be as small as possible. But

$d_v > d_u$, so the highest multiplicative complexity is $(n - d_u)(n + 2d_u + 6) + 1$, which maximizes when $d_u = (n - 6)/4$. And we know $d_u < d_v \leq t$. Let R denote the code rate. So for RS codes with $R > 1/2$, the maximum complexity is $n^2 + nt - 2t^2 + 5n - 2t + 5$ multiplications, $2nt - 2t^2 + 2n + 2$ additions, and one inversion. For codes with $R \leq 1/2$, the maximum complexity is $(9/8)n^2 + (9/2)n + 11/2$ multiplications, $(3/8)n^2 + (3/2)n + 3/2$ additions, and one inversion.

Table 1 lists the complexity of direct implementation of Algorithms 1 and 2, in terms of operations in GF(2^m). The complexity of syndrome-based decoding is given in Table 2. The numbers for syndrome computation, the Chien search, and Forney's formula are from [21]. We assume that the EEA is used for the key equation solver since it was shown to be equivalent to the BMA [22]. The ST is used to implement the EEA. Note that the overall complexity of syndrome-based decoding can be reduced by sharing computations between the Chien search and Forney's formula. However, this is not taken into account in Table 2.

3.2. Complexity comparison

For any application with fixed parameters n and k, the comparison between the algorithms is straightforward using the complexities in Tables 1 and 2. Below we try to determine which algorithm is more suitable for a given code rate. The comparison between different algorithms is complicated by three different types of field operations. However, the complexity is dominated by the number of multiplications: in hardware implementation, both multiplication and inversion over GF(2^m) require an area-time complexity of $O(m^2)$ [23], whereas an addition requires an area-time complexity of $O(m)$; the complexity due to inversions is negligible since the required number of inversions is much smaller than that of multiplications; the numbers of multiplications and additions are both $O(n^2)$. Thus, we focus on the number of multiplications for simplicity.

Since $t = (1/2)(1 - R)n$ and $k = Rn$, the multiplicative complexities of Algorithms 1 and 2 are $(3-R)n^2+(3-R)n+2$ and $(1/2)(3R^2 - 7R + 8)n^2 + (7 - 3R)n + 5$, respectively, while the complexity of syndrome-based decoding is $(1/2)(5R^2 - 13R + 8)n^2 + (2 - 3R)n$. It is easy to verify that in all these complexities, the quadratic and linear coefficients are of the same order of magnitude; hence, we consider only the quadratic terms. Considering only the quadratic terms, Algorithm 1 is less efficient than syndrome-based decoding when $R > 1/5$. If the Chien search and Forney's formula share computations, this threshold will be even lower. Comparing the highest terms, Algorithm 2 is less efficient than the syndrome-based algorithm regardless of R. It is easy to verify that the most significant term of the difference between Algorithms 1 and 2 is $(1/2)(1 - R)(3R - 2)n^2$. So when implemented directly, Algorithm 1 is less efficient than Algorithm 2 when $R > 2/3$. Thus, Algorithm 1 is more suitable for codes with very low rate, while syndrome-based decoding is the most efficient for high-rate codes.

3.3. Hardware costs, latency, and throughput

We have compared the computational complexities of syndromeless decoding algorithms with those of syndrome-based algorithms. Now we compare these two types of decoding algorithms from a hardware perspective: we will compare the hardware costs, latency, and throughput of decoder architectures based on direct implementations of these algorithms. Since our goal is to compare syndrome-based algorithms with syndromeless algorithms, we select our architectures so that the comparison is on a level field. Thus, among various decoder architectures available for syndrome-based decoders in the literature, we consider the hypersystolic architecture in [20]. Not only it is an efficient architecture for syndrome-based decoders, but also some of its functional units can be easily adapted to implement syndromeless decoders. Thus, decoder architectures for both types of decoding algorithms have the same structure with some functional units the same; this allows us to focus on the difference between the two types of algorithms. For the same reason, we do not try to optimize the hardware costs, latency, or throughput using circuit-level techniques since such techniques will benefit from the architectures for both types of decoding algorithms in a similar fashion and hence does not affect the comparison.

The hypersystolic architecture [20] contains three functional units: the power sums tower (PST) computing the syndromes, the ST solving the key equation, and the correction tower (CT) performing the Chien search and Forney's formula. The PST consists of $2t$ systolic cells, each of which comprises of one multiplier, one adder, five registers, and one multiplexer. The ST has $\delta + 1$ (δ is the maximal degree of the input polynomials) systolic cells, each of which contains one multiplier, one adder, five registers, and seven multiplexers. The latency of the ST is 6γ clock cycles [20], where γ is the number of iterations. For the syndrome-based decoder architecture, δ and γ are both $2t$. The CT consists of $3t + 1$ evaluation cells, two delay cells, along with two joiner cells, which also perform inversions. Each evaluation cell needs one multiplier, one adder, four registers, and one multiplexer. Each delay cell needs one register. The two joiner cells altogether need two multipliers, one inverter, and four registers. Table 3 summarizes the hardware costs of the decoder architecture for syndrome-based decoders described above. For each functional unit, we also list the latency (in clock cycles), as well as the number of clock cycles it needs to process one received word, which is proportional to the inverse of the throughput. In theory, the computational complexities of Steps of RS decoding depend on the received word, and the total complexity is obtained by first computing the *sum* of complexities for all the Steps and then considering the worst case scenario (cf. Section 3.1). In contrast, the hardware costs, latency, and throughput of *every* functional unit are dominated by the worst case scenario; the numbers in Table 3 all correspond to the worst case scenario. The critical path delay (CPD) is the same, $T_{\text{mult}} + T_{\text{add}} + T_{\text{mux}}$, for the PST, ST, and CT. In addition to the registers required by the PST, ST, and CT, the total number of registers in Table 3

Table 1: Direct implementation complexities of syndromeless decoding algorithms

		Multiplications	Additions	Inversions
Interpolation		$n(n-1)$	$n(n-1)$	0
Partial GCD	Algorithm 1	$4t(n+2)$	$2t(n+1)$	0
	Algorithm 2	$4t(2t+2)$	$2t(2t+1)$	0
Message recovery	Algorithm 1	$(k+2)(k+1)+2kt$	$k(t+2)$	1
	Algorithm 2	$n^2+nt-2t^2+5n-2t+5$	$2nt-2t^2+2n+2$	1
Total	Algorithm 1	$2n^2+2nt+2n+2t+2$	$n^2+3nt-2t^2+n-2t$	1
	Algorithm 2	$2n^2+nt+6t^2+4n+6t+5$	$n^2+2nt+2t^2+n+2t+2$	1

Table 2: Direct implementation complexity of syndrome-based decoding

	Multiplications	Additions	Inv.
Syndrome computation	$2t(n-1)$	$2t(n-1)$	0
Key equation solver	$4t(2t+2)$	$2t(2t+1)$	0
Chien search	$n(t-1)$	nt	0
Forney's formula	$2t^2$	$t(2t-1)$	t
Total	$3nt+10t^2-n+6t$	$3nt+6t^2-t$	t

also account for the registers needed by the delay line called Main Street [20].

Both the PST and the ST can be adapted to implement decoder architectures for syndromeless decoding algorithms. Similar to syndrome computation, interpolation in syndromeless decoders can be implemented by Horner's rule, and thus the PST can be easily adapted to implement this Step. For the architectures based on syndromeless decoding, the PST contains n cells, and the hardware costs of each cell remain the same. The partial GCD is implemented by the ST. The ST can implement the polynomial division in message recovery as well. In Step (1.3), the maximum polynomial degree of the polynomial division is $k+t$ and the iteration number is at most k. As mentioned in Section 3.1, the degree of $q(x)$ in Step (2.3) ranges from 1 to t. In the polynomial division $g_0(x)/v(x)$, the maximum polynomial degree is n and the iteration number is at most $n-1$. Given the maximum polynomial degree and iteration number, the hardware costs and latency for the ST can be determined as for the syndrome-based architecture.

The other operations of syndromeless decoders do not have corresponding functional units available in the hypersystolic architecture, and we choose to implement them in a straightforward way. In the polynomial multiplication $q(x)u(x)$, $u(x)$ has degree at most $t-1$ and the product has degree at most $n-1$. Thus, it can be done by n multiply-and-accumulate circuits, n registers in t cycles (see, e.g., [24]). The polynomial addition in Step (2.3) can be done in one clock cycle with n adders and n registers. To remove the scaling factor, Step (1.3) is implemented in four cycles with at most one inverter, $k+2$ multipliers, and $k+3$ registers; Step (2.3) is implemented in three cycles with at most one inverter, $n+1$ multipliers, and $n+2$ registers. We summarize the hardware costs, latency, and throughput of the decoder architectures based on Algorithms 1 and 2 in Table 4.

Now we compare the hardware costs of the three decoder architectures based on Tables 3 and 4. The hardware costs are measured by the numbers of various basic circuit elements. All three decoder architectures need only one inverter. The syndrome-based decoder architecture requires fewer multiplexers than the decoder architecture based on Algorithm 1, regardless of the rate, and fewer multipliers, adders, and registers when $R > 1/2$. The syndrome-based decoder architecture requires fewer registers than the decoder architecture based on Algorithm 2 when $R > 21/43$, and fewer multipliers, adders, and multiplexers regardless of the rate. Thus, for high rate codes, the syndrome-based decoder has lower hardware costs than syndromeless decoders. The decoder architecture based on Algorithm 1 requires fewer multipliers and adders than that based on Algorithm 2, regardless of the rate, but more registers and multiplexers when $R > 9/17$.

In these algorithms, each Step starts with the results of the previous Step. Due to this data dependency, their corresponding functional units have to operate in a pipelined fashion. Thus, the decoding latency is simply the sum of the latency of all the functional units. The decoder architecture based on Algorithm 2 has the longest latency, regardless of the rate. The syndrome-based decoder architecture has shorter latency than the decoder architecture based on Algorithm 1 when $R > 1/7$.

All three decoders have the same CPD, so the throughput is determined by the number of clock cycles. Since the functional units in each decoder architecture are pipelined, the throughput of each decoder architecture is determined by the functional unit that requires the largest number of cycles. Regardless of the rate, the decoder based on Algorithm 2 has the lowest throughput. When $R > 1/2$, the syndrome-based decoder architecture has higher throughput than the decoder architecture based on Algorithm 1. When the rate is lower, they have the same throughput.

Hence, for high-rate RS codes, the syndrome-based decoder architecture requires less hardware and achieves higher throughput and shorter latency than those based on syndromeless decoding algorithms.

4. FAST IMPLEMENTATION OF SYNDROMELESS DECODING

In this section, we implement the three Steps of Algorithms 1 and 2: interpolation, partial GCD, and message recovery,

Table 3: Decoder architecture based on syndrome-based decoding (CPD is $T_{\text{mult}} + T_{\text{add}} + T_{\text{mux}}$)

	Multipliers	Adders	Inverters	Registers	Muxes	Latency	Throughput^{-1}
Syndrome computation	$2t$	$2t$	0	$10t$	$2t$	$n + 6t$	$6t$
Key equation solver	$2t + 1$	$2t + 1$	0	$10t + 5$	$14t + 7$	$12t$	$12t$
Correction	$3t + 3$	$3t + 1$	1	$12t + 10$	$3t + 1$	$3t$	$3t$
Total	$7t + 4$	$7t + 2$	1	$n + 53t + 15$	$19t + 8$	$n + 21t$	$12t$

Table 4: Decoder architectures based on syndromeless decoding (CPD is $T_{\text{mult}} + T_{\text{add}} + T_{\text{mux}}$)

		Multipliers	Adders	Inverters	Registers	Muxes	Latency	Throughput^{-1}
Interpolation		n	n	0	$5n$	n	$4n$	$3n$
Partial GCD	Algorithm 1	$n + 1$	$n + 1$	0	$5n + 5$	$7n + 7$	$12t$	$12t$
	Algorithm 2	$2t + 1$	$2t + 1$	0	$10t + 5$	$14t + 7$	$12t$	$12t$
Message recovery	Algorithm 1	$2k + t + 3$	$k + t + 1$	1	$6k + 5t + 8$	$7k + 7t + 7$	$6k + 4$	$6k$
	Algorithm 2	$3n + 2$	$3n + 1$	1	$7n + 7$	$7n + 7$	$6n + t - 2$	$6n$
Total	Algorithm 1	$2n + 2k + t + 4$	$2n + k + t + 2$	1	$10n + 6k + 5t + 13$	$8n + 7k + 7t + 14$	$4n + 6k + 12t + 4$	$6k$
	Algorithm 2	$4n + 2t + 3$	$4n + 2t + 2$	1	$12n + 10t + 12$	$8n + 14t + 14$	$10n + 13t - 2$	$6n$

by fast algorithms described in Section 2 and evaluate their complexities. Since both the polynomial division by Newton iteration and the FEEA depend on efficient polynomial multiplication, the decoding complexity relies on the complexity of polynomial multiplication. Thus, in addition to field multiplications and additions, the complexities in this section are also expressed in terms of polynomial multiplications.

4.1. Polynomial multiplication

We first derive a tighter bound on the complexity of the fast polynomial multiplication based on Cantor's approach.

Let the degree of the product of two polynomials be less than n. The polynomial multiplication can be done by two FFTs and one inverse FFT if length-n FFT is available over GF(2^m), which requires $n \mid 2^m - 1$. If $n \nmid 2^m - 1$, one option is to pad the polynomials to length n' ($n' > n$) with $n' \mid 2^m - 1$. Compared with fast polynomial multiplication based on multiplicative FFT, Cantor's approach uses additive FFT and does not require $n \mid 2^m - 1$, so it is more efficient than FFT multiplication with padding for most degrees. For $n = 2^m - 1$, their complexities are similar. Although asymptotically worse than Schönhage's algorithm [12], which has $O(n \log n \log\log n)$ complexity, Cantor's approach has small implicit constants, and hence, it is more suitable for practical implementation of RS codes [6, 11]. Gao claimed an improvement on Cantor's approach in [6], but we do not pursue this due to lack of details.

A tighter bound on the complexity of Cantor's approach is given in Theorem 1. Here we make the same assumption as in [11] that the auxiliary polynomials s_i and the values $s_i(\beta_j)$ are precomputed. The complexity of precomputation was given in [11].

Theorem 1. *By Cantor's approach, two polynomials $a, b \in GF(2^m)[x]$ whose product has a degree less than h ($1 \le h \le 2^m$) can be multiplied using less than $(3/2)h\log^2 h + (7/2)h\log h - 2h + \log h + 2$ multiplications, $(3/2)h\log^2 h + (21/2)h\log h - 13h + \log h + 15$ additions, and $2h$ inversions over GF(2^m).*

Proof. There exists $0 \le p \le m$ satisfying $2^{p-1} < h \le 2^p$. Since both the MPE and MPI algorithms are recursive, we denote the numbers of additions of the MPE and MPI algorithms for input i ($0 \le i \le p$) as $S_E(i)$ and $S_I(i)$, respectively. Clearly, $S_E(0) = S_I(0) = 0$. Following the approach in [11], it can be shown that for $1 \le i \le p$,

$$S_E(i) \le i(i+3)2^{i-2} + (p-3)(2^i - 1) + i, \tag{1}$$

$$S_I(i) \le i(i+5)2^{i-2} + (p-3)(2^i - 1) + i. \tag{2}$$

Let $M_E(h)$ and $A_E(h)$ denote the numbers of multiplications and additions, respectively, that the MPE algorithm requires for polynomials of a degree less than h. When $i = p$ in the MPE algorithm, $f(x)$ has a degree less than $h \le 2^p$, while s_{p-1} is of degree 2^{p-1} and has at most p nonzero coefficients. Thus, $g(x)$ has a degree less than $h - 2^{p-1}$. Therefore, the numbers of multiplications and additions for the polynomial division in [11, Step 2 of Algorithm 3.1] are both $p(h - 2^{p-1})$, while $r_1(x) = r_0(x) + s_{i-1}(\beta_i)g(x)$ needs at most $h - 2^{p-1}$ multiplications and the same number of additions. Substituting the bound on $M_E(2^{p-1})$ in [11], we obtain $M_E(h) \le 2M_E(2^{p-1}) + p(h - 2^{p-1}) + h - 2^{p-1}$, and thus $M_E(h)$ is at most $(1/4)p^2 2^p - (1/4)p2^p - 2^p + (p+1)h$. Similarly, substituting the bound on $S_E(p-1)$ in (1), we obtain $A_E(h) \le 2S_E(p-1) + p(h - 2^{p-1}) + h - 2^{p-1}$, and hence $A_E(h)$ is at most $(1/4)p^2 2^p + (3/4)p2^p - 4\cdot 2^p + (p+1)h + 4$.

Let $M_I(h)$ and $A_I(h)$ denote the numbers of multiplications and additions, respectively, which the MPI algorithm requires when the interpolated polynomial has a degree less than h. When $i = p$ in the MPI algorithm, $f(x)$ has a degree less than $h \le 2^p$. It implies that $r_0(x) + r_1(x)$ has a degree less than $h - 2^{p-1}$. Thus, it requires at most $h - 2^{p-1}$ additions to obtain $r_0(x) + r_1(x)$ and $h - 2^{p-1}$ multiplications for $s_{i-1}(\beta_i)^{-1}(r_0(x) + r_1(x))$. The numbers of multiplications and

additions for the polynomial multiplication in [11, Step 3 of Algorithm 3.2] to obtain $f(x)$ are both $p(h-2^{p-1})$. Adding $r_0(x)$ also needs 2^{p-1} additions. Substituting the bound on $M_I(2^{p-1})$ in [11], we have $M_I(h) \le 2M_I(2^{p-1}) + p(h-2^{p-1}) + h - 2^{p-1}$, and hence $M_I(h)$ is at most $(1/4)p^2 2^p - (1/4)p2^p - 2^p + (p+1)h$. Similarly, substituting the bound on $S_I(p-1)$ in (2), we have $A_I(h) \le 2S_I(p-1) + p(h-2^{p-1}) + h + 1$, and hence $A_E(h)$ is at most $(1/4)p^2 2^p + (5/4)p2^p - 4\cdot 2^p + (p+1)h + 5$. The interpolation Step also needs 2^p inversions.

Let $\mathsf{M}(h_1, h_2)$ be the complexity of multiplication of two polynomials of degrees less than h_1 and h_2. Using Cantor's approach, $\mathsf{M}(h_1, h_2)$ includes $M_E(h_1) + M_E(h_2) + M_I(h) + 2^p$ multiplications, $A_E(h_1) + A_E(h_2) + A_I(h)$ additions, and 2^p inversions, when $h = h_1 + h_2 - 1$. Finally, we replace 2^p by $2h$ as in [11]. □

Compared with the results in [11], our results have the same highest degree term but smaller terms for lower degrees.

By Theorem 1, we can easily compute $\mathsf{M}(h_1) \triangleq \mathsf{M}(h_1, h_1)$. A by-product of the above proof is the bounds for the MPE and MPI algorithms. We also observe some properties for the complexity of fast polynomial multiplication that hold for not only Cantor's approach but also for other approaches. These properties will be used in our complexity analysis next. Since all fast polynomial multiplication algorithms have higher-than-linear complexities, $2\mathsf{M}(h) \le \mathsf{M}(2h)$. Also note that $\mathsf{M}(h+1)$ is no more than $\mathsf{M}(h)$ plus $2h$ multiplications and $2h$ additions [12, Exercise 8.34]. Since the complexity bound is determined only by the degree of the product polynomial, we assume $\mathsf{M}(h_1, h_2) \le \mathsf{M}(\lceil (h_1 + h_2)/2 \rceil)$. We note that the complexities of Schönhage's algorithm as well as Schönhage and Strassen's algorithm, both based on multiplicative FFT, are also determined by the degree of the product polynomial [12].

4.2. Polynomial division

Similar to [12, Exercise 9.6], in characteristic-2 fields, the complexity of Newton iteration is at most

$$\sum_{0 \le j \le r-1} \left(\mathsf{M}\left(\lceil (d_0+1)2^{-j} \rceil\right) + \mathsf{M}\left(\lceil (d_0+1)2^{-j-1} \rceil\right)\right), \quad (3)$$

where $r = \lceil \log(d_0+1) \rceil$. Since $\lceil (d_0+1)2^{-j} \rceil \le \lfloor (d_0+1)2^{-j} \rfloor + 1$ and $\mathsf{M}(h+1)$ is no more than $\mathsf{M}(h)$, plus $2h$ multiplications and $2h$ additions [12, Exercise 8.34], it requires at most $\sum_{1 \le j \le r} (\mathsf{M}(\lfloor (d_0+1)2^{-j} \rfloor) + \mathsf{M}(\lfloor (d_0+1)2^{-j-1} \rfloor))$, plus $\sum_{0 \le j \le r-1} (2\lfloor (d_0+1)2^{-j} \rfloor + 2\lfloor (d_0+1)2^{-j-1} \rfloor)$ multiplications and the same number of additions. Since $2\mathsf{M}(h) \le \mathsf{M}(2h)$, Newton iteration costs at most $\sum_{0 \le j \le r-1} ((3/2)\mathsf{M}(\lfloor (d_0+1)2^{-j} \rfloor)) \le 3\mathsf{M}(d_0+1)$, $6(d_0+1)$ multiplications, and $6(d_0+1)$ additions. The second Step to compute the quotient needs $\mathsf{M}(d_0+1)$ and the last Step to compute the remainder needs $\mathsf{M}(d_1+1, d_0+1)$ and d_1+1 additions. By $\mathsf{M}(d_1+1, d_0+1) \le \mathsf{M}(\lceil (d_0+d_1)/2 \rceil + 1)$, the total cost is at most $4\mathsf{M}(d_0) + \mathsf{M}(\lceil (d_0+d_1)/2 \rceil)$, $15d_0 + d_1 + 7$ multiplications, and $11d_0 + 2d_1 + 8$ additions. Note that this bound does not require $d_1 \ge d_0$ as in [12].

4.3. Partial GCD

The partial GCD Step can be implemented in three approaches: the ST, the classical EEA with fast polynomial multiplication and Newton iteration, and the FEEA with fast polynomial multiplication and Newton iteration. The ST is essentially the classical EEA. The complexity of the classical EEA is asymptotically worse than that of the FEEA. Since the FEEA is more suitable for long codes, we will use the FEEA in our complexity analysis of fast implementations.

In order to derive a tighter bound on the complexity of the FEEA, we first present a modified FEEA in Algorithm 3. Let $\eta(h) \triangleq \max\{j : \sum_{i=1}^{j} \deg q_i \le h\}$, which is the number of Steps of the EEA satisfying $\deg r_0 - \deg r_{\eta(h)} \le h < \deg r_0 - \deg r_{\eta(h)+1}$. For $f(x) = f_n x^n + \cdots + f_1 x + f_0$ with $f_n \ne 0$, the truncated polynomial $f(x) \upharpoonright h \triangleq f_n x^h + \cdots + f_{n-h+1}x + f_{n-h}$, where $f_i = 0$ for $i < 0$. Note that $f(x) \upharpoonright h = 0$ if $h < 0$.

Algorithm 3. (modified fast extended Euclidean algorithm)

Input: two *monic* polynomials r_0 and r_1, with $\deg r_0 = n_0 > n_1 = \deg r_1$, as well as integer h $(0 < h \le n_0)$

Output: $l = \eta(h)$, ρ_{l+1}, R_l, r_l, and $\widetilde{r}_{l+1}$

(3.1) If $r_1 = 0$ or $h < n_0 - n_1$, then return $0, 1, \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$, r_0, and r_1.

(3.2) $h_1 = \lfloor h/2 \rfloor$, $r_0^* = r_0 \upharpoonright 2h_1$, $r_1^* = r_1 \upharpoonright (2h_1 - (n_0 - n_1))$.

(3.3) $(j-1, \rho_j^*, R_{j-1}^*, r_{j-1}^*, \widetilde{r}_j^*) = \mathrm{FEEA}(r_0^*, r_1^*, h_1)$.

(3.4) $\left[\begin{smallmatrix} r_{j-1} \\ \widetilde{r}_j \end{smallmatrix}\right] = R_{j-1}^* \left[\begin{smallmatrix} r_0 - r_0^* x^{n_0-2h_1} \\ r_1 - r_1^* x^{n_0-2h_1} \end{smallmatrix}\right] + \left[\begin{smallmatrix} r_{j-1}^* x^{n_0-2h_1} \\ \widetilde{r}_j^* x^{n_0-2h_1} \end{smallmatrix}\right]$, $R_{j-1} = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1/\mathrm{lc}(\widetilde{r}_j) \end{smallmatrix}\right] R_{j-1}^*$, $\rho_j = \rho_j^* \mathrm{lc}(\widetilde{r}_j)$, $r_j = \widetilde{r}_j / \mathrm{lc}(\widetilde{r}_j)$, $n_j = \deg r_j$.

(3.5) If $r_j = 0$ or $h < n_0 - n_j$, then return $j-1$, ρ_j, R_{j-1}, r_{j-1}, and $\widetilde{r}_j$.

(3.6) Perform polynomial division with remainder as $r_{j-1} = q_j r_j + \widetilde{r}_{j+1}$, $\rho_{j+1} = \mathrm{lc}(\widetilde{r}_{j+1})$, $r_{j+1} = \widetilde{r}_{j+1}/\rho_{j+1}$, $n_{j+1} = \deg r_{j+1}$, $R_j = \left[\begin{smallmatrix} 0 & 1 \\ 1/\rho_{j+1} & -q_j/\rho_{j+1} \end{smallmatrix}\right] R_{j-1}$.

(3.7) $h_2 = h - (n_0 - n_j)$, $r_j^* = r_j \upharpoonright 2h_2$, $r_{j+1}^* = r_{j+1} \upharpoonright (2h_2 - (n_j - n_{j+1}))$.

(3.8) $(l-j, \rho_{l+1}^*, S^*, r_{l-j}^*, \widetilde{r}_{l-j+1}^*) = \mathrm{FEEA}(r_j^*, r_{j+1}^*, h_2)$.

(3.9) $\left[\begin{smallmatrix} r_l \\ \widetilde{r}_{l+1} \end{smallmatrix}\right] = S^* \left[\begin{smallmatrix} r_j - r_j^* x^{n_j-2h_2} \\ r_{j+1} - r_{j+1}^* x^{n_j-2h_2} \end{smallmatrix}\right] + \left[\begin{smallmatrix} r_{l-j}^* x^{n_j-2h_2} \\ \widetilde{r}_{l-j+1}^* x^{n_j-2h_2} \end{smallmatrix}\right]$, $S = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1/\mathrm{lc}(\widetilde{r}_{l+1}) \end{smallmatrix}\right] S^*$, $\rho_{l+1} = \rho_{l+1}^* \mathrm{lc}(\widetilde{r}_{l+1})$.

(3.10) Return l, ρ_{l+1}, SR_j, r_l, $\widetilde{r}_{l+1}$.

It is easy to verify that Algorithm 3 is equivalent to the FEEA in [12, 17]. The difference between Algorithm 3 and the FEEA in [12, 17] lies in Steps (3.4), (3.5), (3.8), and (3.10): in Steps (3.5) and (3.10), two additional polynomials are returned, and they are used in the updates of Steps (3.4) and (3.8) to reduce complexity. The modification in Step (3.4) was suggested in [14] and the modification in Step (3.9) follows the same idea.

In [12, 14], the complexity bounds of the FEEA are established assuming $n_0 \le 2h$. Thus, we first establish a bound of the FEEA for the case $n_0 \le 2h$ below in Theorem 2,

using the bounds we developed in Sections 4.1 and 4.2. The proof is similar to those in [12, 14] and hence omitted; interested readers should have no difficulty filling in the details.

Theorem 2. *Let $T(n_0, h)$ denote the complexity of the FEEA. When $n_0 \leq 2h$, $T(n_0, h)$ is at most $17\mathsf{M}(h)\log h$ plus $(48h + 2)\log h$ multiplications, $(51h + 2)\log h$ additions, and $3h$ inversions. Furthermore, if the degree sequence is normal, $T(2h, h)$ is at most $10\mathsf{M}(h)\log h$, $((55/2)h + 6)\log h$ multiplications, and $((69/2)h + 3)\log h$ additions.*

Compared with the complexity bounds in [12, 14], our bound not only is tighter, but also specifies all terms of the complexity and avoid the big O notation. The saving over [14] is due to lower complexities of Steps (3.6), (3.9), and (3.10) as explained above. The saving for the normal case over [12] is due to lower complexity of Step (3.9).

Applying the FEEA to $g_0(x)$ and $g_1(x)$ to find $v(x)$ and $g(x)$ in Algorithm 1, we have $n_0 = n$ and $h \leq t$ since $\deg v(x) \leq t$. For RS codes, we always have $n > 2t$. Thus, the condition $n_0 \leq 2h$ for the complexity bound in [12, 14] is not valid. It was pointed out in [6, 12] that $s_0(x)$ and $s_1(x)$ as defined in Algorithm 2 can be used instead of $g_0(x)$ and $g_1(x)$, which is the difference between Algorithms 1 and 2. Although such a transform allows us to use the results in [12, 14], it introduces extra cost for message recovery [6]. To compare the complexities of Algorithms 1 and 2, we establish a more general bound in Theorem 3.

Theorem 3. *The complexity of FEEA is no more than $34\mathsf{M}(\lfloor h/2\rfloor)\log\lfloor h/2\rfloor + \mathsf{M}(\lfloor n_0/2\rfloor) + 4\mathsf{M}(\lceil n_0/2 - h/4\rceil) + 2\mathsf{M}(\lfloor (n_0-h)/2\rfloor)+4\mathsf{M}(h)+2\mathsf{M}(\lfloor (3/4)h\rfloor)+4\mathsf{M}(\lfloor h/2\rfloor)$, $(48h+4)\log\lfloor h/2\rfloor + 9n_0 + 22h$ multiplications, $(51h + 4)\log\lfloor h/2\rfloor + 11n_0 + 17h + 2$ additions, and $3h$ inversions.*

The proof is also omitted for brevity. The main difference between this case and Theorem 2 lies in the top level call of the FEEA. The total complexity is obtained by adding $2T(h, \lfloor h/2\rfloor)$ and the top-level cost.

It can be verified that, when $n_0 \leq 2h$, Theorem 3 presents a tighter bound than Theorem 2 since saving on the top level is accounted for. Note that the complexity bounds in Theorems 2 and 3 assume that the FEEA solves $s_{l+1}r_0 + t_{l+1}r_1 = \widetilde{r}_{l+1}$ for both t_{l+1} and s_{l+1}. If s_{l+1} is not necessary, the complexity bounds in Theorems 2 and 3 are further reduced by $2\mathsf{M}(\lfloor h/2\rfloor)$, $3h + 1$ multiplications, and $4h + 1$ additions.

4.4. Complexity comparison

Using the results in Sections 4.1, 4.2, and 4.3, we first analyze and then compare the complexities of Algorithms 1 and 2 as well as syndrome-based decoding under fast implementations.

In Steps (1.1) and (2.1), $g_1(x)$ can be obtained by an inverse FFT when $n \mid 2^m - 1$ or by the MPI algorithm. In the latter case, the complexity is given in Section 4.1. By Theorem 3, the complexity of Step (1.2) is $T(n, t)$ minus the complexity to compute s_{l+1}. The complexity of Step (2.2) is $T(2t, t)$. The complexity of Step (1.3) is given by the bound in Section 4.2. Similarly, the complexity of Step (2.3) is readily obtained by using the bounds of polynomial division and multiplication.

All the steps of syndrome-based decoding can be implemented using fast algorithms. Both syndrome computation and the Chien search can be done by n-point evaluations. Forney's formula can be done by two t-point evaluations plus t inversions and t multiplications. To use the MPE algorithm, we choose to evaluate on all n points. By Theorem 3, the complexity of the key equation solver is $T(2t, t)$ minus the complexity to compute s_{l+1}.

Note that to simplify the expressions, the complexities are expressed in terms of three kinds of operations: polynomial multiplications, field multiplications, and field additions. Of course, with our bounds on the complexity of polynomial multiplication in Theorem 1, the complexities of the decoding algorithms can be expressed in terms of field multiplications and additions.

Given the code parameters, the comparison among these algorithms is quite straightforward with the above expressions. As in Section 3.2, we attempt to compare the complexities using only R. Such a comparison is of course not accurate, but it sheds light on the comparative complexity of these decoding algorithms without getting entangled in the details. To this end, we make four assumptions. First, we assume the complexity bounds on the decoding algorithms as approximate decoding complexities. Second, we use the complexity bound in Theorem 1 as approximate polynomial multiplication complexities. Third, since the numbers of multiplications and additions are of the same degree, we only compare the numbers of multiplications. Fourth, we focus on the difference of the second highest degree terms since the highest degree terms are the same for all three algorithms. This is because the partial GCD Steps of Algorithms 1 and 2, as well as the key equation solver in syndrome-based decoding, differ only in the top level of the recursion of FEEA. Hence, Algorithms 1 and 2 as well as the key equation solver in syndrome-based decoding have the same highest degree term.

We first compare the complexities of Algorithms 1 and 2. Using Theorem 1, the difference between the second highest degree terms is given by $(3/4)(25R - 13)n\log^2 n$, so Algorithm 1 is less efficient than Algorithm 2 when $R > 0.52$. Similarly, the complexity difference between syndrome-based decoding and Algorithm 1 is given by $(3/4)(1 - 31R)n\log^2 n$. Thus, syndrome-based decoding is more efficient than Algorithm 1 when $R > 0.032$. Comparing syndrome-based decoding and Algorithm 2, the complexity difference is roughly $-(9/2)(2+R)n\log^2 n$. Hence, syndrome-based decoding is more efficient than Algorithm 2 regardless of the rate.

We remark that the conclusion of the above comparison is similar to those obtained in Section 3.2 except the thresholds are different. Based on fast implementations, Algorithm 1 is more efficient than Algorithm 2 for low rate codes, and the syndrome-based decoding is more efficient than Algorithms 1 and 2 in virtually all cases.

Table 5: Complexity of syndromeless decoding

(n,k)		Direct implementation								Fast implementation							
		Algorithm 1				Algorithm 2				Algorithm 1				Algorithm 2			
		Mult.	Add.	Inv.	Overall	Mult.	Add.	Inv.	Overall	Mult.	Add.	Inv.	Overall	Mult.	Add.	Inv.	Overall
(255, 233)	Interpolation	64770	64770	0	1101090	64770	64770	0	11101090	586	6900	0	16276	586	6900	0	16276
	Partial GCD	16448	8192	0	271360	2176	1056	0	35872	8224	8176	16	140016	1392	1328	16	23856
	Msg recovery	57536	4014	1	924606	69841	8160	1	1125632	3791	3568	1	64240	8160	7665	1	138241
	Total	138754	76976	1	2297056	136787	73986	1	2262594	12601	18644	17	220532	10138	15893	17	178373
(511, 447)	Interpolation	260610	260610	0	4951590	260610	260610	0	4951590	1014	23424	0	41676	1014	23424	0	41676
	Partial GCD	65664	32768	0	1214720	8448	4160	0	156224	32832	32736	32	624288	5344	5216	32	101984
	Msg recovery	229760	15198	1	4150896	277921	31680	1	5034276	14751	14304	1	279840	31680	30689	1	600947
	Total	556034	308576	1	10317206	546979	296450	1	10142090	48597	70464	33	945804	38038	59329	33	744607

Table 6: Complexity of syndrome-based decoding

(n,k)		Direct implementation				Fast implementation			
		Mult.	Add.	Inv.	Overall	Mult.	Add.	Inv.	Overall
(255, 223)	Syndrome computation	8128	8128	0	138176	149	4012	0	6396
	Key equation solver	2176	1056	0	35872	1088	1040	16	18704
	Chien search	3825	4080	0	65280	586	6900	0	16276
	Forney's formula	512	496	16	8944	512	496	16	8944
	Total	14641	13760	16	248272	2335	12448	32	50320
(511, 447)	Syndrome computation	32640	32640	0	620160	345	16952	0	23162
	Key equation solver	8448	4160	0	156224	4224	4128	32	80736
	Chien search	15841	16352	0	301490	1014	23424	0	41676
	Forney's formula	2048	2016	32	39456	2048	2016	32	39456
	Total	58977	55168	32	1117330	7631	46520	64	185030

5. CASE STUDY AND DISCUSSIONS

5.1. Case study

We examine the complexities of Algorithms 1 and 2 as well as syndrome-based decoding for the $(255, 223)$ CCSDS RS code [25] and a $(511, 447)$ RS code which have roughly the same rate $R = 0.87$. Again, both direct and fast implementations are investigated. Due to the moderate lengths, in some cases direct implementation leads to lower complexity, and hence in such cases, the complexity of direct implementation is used for both.

Tables 5 and 6 list the total decoding complexity of Algorithms 1 and 2 as well as syndrome-based decoding, respectively. In the fast implementations, cyclotomic FFT [16] is used for interpolation, syndrome computation, and the Chien search. The classical EEA with fast polynomial multiplication and division is used in fast implementations since it is more efficient than the FEEA for these lengths. We assume normal degree sequence, which represents the worst case scenario [12]. The message recovery Steps use long division in fast implementation since it is more efficient than Newton iteration for these lengths. We use Horner's rule for Forney's formula in both direct and fast implementations.

We note that for each decoding Step, Tables 5 and 6 not only provide the numbers of finite field multiplications, additions, and inversions, but also list the overall complexities to facilitate comparisons. The overall complexities are computed based on the assumptions that multiplication and inversion are of equal complexity, and that as in [15], one multiplication is equivalent to $2m$ additions. The latter assumption is justified by both hardware and software implementations of finite field operations. In hardware implementation, a multiplier over GF(2^m) generated by trinomials requires $m^2 - 1$ XOR and m^2 AND gates [26], while an adder requires m XOR gates. Assuming that XOR and AND gates have the same complexity, the complexity of a multiplier is $2m$ times that of an adder over GF(2^m). In software implementation, the complexity can be measured by the number of word-level operations [27]. Using the shift and add method as in [27], a multiplication requires $m - 1$ shift and m XOR word-level operations, respectively, while an addition needs only one XOR word-level operation. Henceforth, in software implementations the complexity of a multiplication over GF(2^m) is also roughly $2m$ times as that of an addition. Thus, the total complexity of each decoding Step in Tables 5 and 6 is obtained by $N = 2m(N_{\text{mult}} + N_{\text{inv}}) + N_{\text{add}}$, which is in terms of field additions.

Comparisons between direct and fast implementations for each algorithm show that fast implementations considerably reduce the complexities of both syndromeless and syndrome-based decoding, as shown in Tables 5 and 6. The comparison between these tables shows that for these two high-rate codes, both direct and fast implementations of syndromeless decoding are not as efficient as their counterparts of syndrome-based decoding. This observation is consistent with our conclusions in Sections 3.2 and 4.4.

For these two codes, hardware costs and throughput of decoder architectures based on direct implementations of syndrome-based and syndromeless decoding can be easily obtained by substituting the parameters in Tables 3 and 4; thus for these two codes, the conclusions in Section 3.3 apply.

5.2. Errors-and-erasures decoding

The complexity analysis of RS decoding in Sections 3 and 4 has assumed errors-only decoding. We extend our complexity analysis to errors-and-erasures decoding below.

Syndrome-based errors-and-erasures decoding has been well studied, and we adopt the approach in [18]. In this approach, first erasure locator polynomial and modified syndrome polynomial are computed. After the error locator polynomial is found by the key equation solver, the errata locator polynomial is computed and the error-and-erasure values are computed by Forney's formula. This approach is used in both direct and fast implementations.

Syndromeless errors-and-erasures decoding can be carried out in two approaches. Let us denote the number of erasures as ν $(0 \leq \nu \leq 2t)$, and up to $f = \lfloor (2t - \nu)/2 \rfloor$ errors can be corrected given ν erasures. As pointed out in [5, 6], the first approach is to ignore the ν erased coordinates, thereby transforming the problem into errors-only decoding of an $(n - \nu, k)$ shortened RS code. This approach is more suitable for direct implementation. The second approach is similar to syndrome-based errors-and-erasures decoding described above, which uses the erasure locator polynomial [5]. In the second approach, only the partial GCD Step is affected, while the same fast implementation techniques described in Section 4 can be used in the other Steps. Thus, the second approach is more suitable for fast implementation.

We readily extend our complexity analysis for errors-only decoding in Sections 3 and 4 to errors-and-erasures decoding. Our conclusions for errors-and-erasures decoding are the same as those for errors-only decoding: Algorithm 1 is the most efficient only for very low rate codes; syndrome-based decoding is the most efficient algorithm for high rate codes. For brevity, we omit the details and interested readers will have no difficulty filling in the details.

6. CONCLUSION

We analyze the computational complexities of two syndromeless decoding algorithms for RS codes using both direct implementation and fast implementation, and compare them with their counterparts of syndrome-based decoding. With either direct or fast implementation, syndromeless algorithms are more efficient than the syndrome-based algorithms only for RS codes with very low rate. When implemented in hardware, syndrome-based decoders also have lower complexity and higher throughput. Since RS codes in practice are usually high-rate codes, syndromeless decoding algorithms are not suitable for these codes. Our case study also shows that fast implementations can significantly reduce the decoding complexity. Errors-and-erasures decoding is also investigated although the details are omitted for brevity.

ACKNOWLEDGMENTS

This work was supported in part by Thales Communications Inc. and in part by a grant from the Commonwealth of Pennsylvania, Department of Community and Economic Development, through the Pennsylvania Infrastructure Technology Alliance (PITA). The authors are grateful to Dr. Jürgen Gerhard for valuable discussions. The authors would also like to thank the reviewers for their constructive comments, which have resulted in significant improvements in the manuscript. The material in this paper was presented in part at the IEEE Workshop on Signal Processing Systems, Shanghai, China, October 2007.

REFERENCES

[1] S. B. Wicker and V. K. Bhargava, Eds., *Reed–Solomon Codes and Their Applications*, IEEE Press, New York, NY, USA, 1994.

[2] E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, NY, USA, 1968.

[3] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A method for solving key equation for decoding Goppa codes," *Information and Control*, vol. 27, no. 1, pp. 87–99, 1975.

[4] A. Shiozaki, "Decoding of redundant residue polynomial codes using Euclid's algorithm," *IEEE Transactions on Information Theory*, vol. 34, no. 5, part 1, pp. 1351–1354, 1988.

[5] A. Shiozaki, T. K. Truong, K. M. Cheung, and I. S. Reed, "Fast transform decoding of nonsystematic Reed–Solomon codes," *IEE Proceedings: Computers and Digital Techniques*, vol. 137, no. 2, pp. 139–143, 1990.

[6] S. Gao, "A new algorithm for decoding Reed–Solomon codes," in *Communications, Information and Network Security*, V. K. Bhargava, H. V. Poor, V. Tarokh, and S. Yoon, Eds., pp. 55–68, Kluwer Academic Publishers, Norwell, Mass, USA, 2003.

[7] S. V. Fedorenko, "A simple algorithm for decoding Reed–Solomon codes and its relation to the Welch–Berlekamp algorithm," *IEEE Transactions on Information Theory*, vol. 51, no. 3, pp. 1196–1198, 2005.

[8] S. V. Fedorenko, "Correction to "A simple algorithm for decoding Reed–Solomon codes and its relation to the Welch–Berlekamp algorithm"," *IEEE Transactions on Information Theory*, vol. 52, no. 3, p. 1278, 2006.

[9] L. R. Welch and E. R. Berlekamp, "Error correction for algebraic block codes," US patent 4633470, September 1983.

[10] J. Justesen, "On the complexity of decoding Reed–Solomon codes," *IEEE Transactions on Information Theory*, vol. 22, no. 2, pp. 237–238, 1976.

[11] J. von zur Gathen and J. Gerhard, "Arithmetic and factorization of polynomials over $\mathbb{F}_2$," Tech. Rep. tr-rsfb-96-018, University of Paderborn, Paderborn, Germany, 1996, http://www-math.uni-paderborn.de/~aggathen/Publications/gatger96a.ps.

[12] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, Cambridge, UK, 2nd edition, 2003.

[13] D. G. Cantor, "On arithmetical algorithms over finite fields," *Journal of Combinatorial Theory, Series A*, vol. 50, no. 2, pp. 285–300, 1989.

[14] S. Khodadad, *Fast rational function reconstruction*, M.S. thesis, Simon Fraser University, Burnaby, BC, Canada, 2005.

[15] Y. Wang and X. Zhu, "A fast algorithm for the Fourier transform over finite fields and its VLSI implementation," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 3, pp. 572–577, 1988.

[16] N. Chen and Z. Yan, "Reduced-complexity cyclotomic FFT and its application in Reed–Solomon decoding," in *Proceedings of the IEEE Workshop on Signal Processing Systems (SIPS '07)*, pp. 657–662, Shanghai, China, October 2007.

[17] S. Khodadad and M. Monagan, "Fast rational function reconstruction," in *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC '06)*, pp. 184–190, ACM Press, Genoa, Italy, July 2006.

[18] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*, John Wiley & Sons, Hoboken, NJ, USA, 2005.

[19] J. J. Komo and L. L. Joiner, "Adaptive Reed–Solomon decoding using Gao's algorithm," in *Proceedings of the IEEE Military Communications Conference (MILCOM '02)*, vol. 2, pp. 1340–1343, Anaheim, Calif, USA, October 2002.

[20] E. Berlekamp, G. Seroussi, and P. Tong, "A hypersystolic Reed–Solomon decoder," in *Reed–Solomon Codes and Their Applications*, S. B. Wicker and V. K. Bhargava, Eds., pp. 205–241, IEEE Press, New York, NY, USA, 1994.

[21] D. Mandelbaum, "On decoding of Reed–Solomon codes," *IEEE Transactions on Information Theory*, vol. 17, no. 6, pp. 707–712, 1971.

[22] A. E. Heydtmann and J. M. Jensen, "On the equivalence of the Berlekamp–Massey and the Euclidean algorithms for decoding," *IEEE Transactions on Information Theory*, vol. 46, no. 7, pp. 2614–2624, 2000.

[23] Z. Yan and D. V. Sarwate, "New systolic architectures for inversion and division in GF(2^m)," *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1514–1519, 2003.

[24] T. Park, "Design of the (248, 216) Reed–Solomon decoder with erasure correction for Blu-ray disc," *IEEE Transactions on Consumer Electronics*, vol. 51, no. 3, pp. 872–878, 2005.

[25] "Telemetry Channel Coding," CCSDS Std. 101.0-B-6, October 2002.

[26] B. Sunar and Ç.K. Koç, "Mastrovito multiplier for all trinomials," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 522–527, 1999.

[27] A. Mahboob and N. Ikram, "Lookup table based multiplication technique for GF(2^m) with cryptographic significance," *IEE Proceedings: Communications*, vol. 152, no. 6, pp. 965–974, 2005.