Partially Parallel Low-Complexity Chase Decoding of Reed-Solomon Codes

Jiwei Liang [†], Lijia Yang [‡], Li Chen [§]
[†]School of System Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China
[‡]School of Electronics and Communication Technology, Sun Yat-sen University, Shenzhen 518107, China
[§]School of Electronics and Information Technology, Sun Yat-sen University, Guangzhou 510006, China
Email: liangjw59@mail2.sysu.edu.cn, yanglj39@mail2.sysu.edu.cn, chenli55@mail.sysu.edu.cn

2022 IEEE Globecom Workshops (GC Wkshps) | 978-1-6654-5975-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/GCWkshps56602.2022.10008502

***Abstract*—This paper proposes the partially parallel low-complexity Chase (PPLCC) decoding for Reed-Solomon (RS) codes. With the formulated test-vectors, the Kötter's interpolation based Chase decoding events are processed in a partially parallel manner, maintaining both low decoding complexity and latency. The decoding will be terminated once a codeword candidate that satisfies the maximum-likelihood (ML) criterion is found. Furthermore, a skipping rule is introduced to reduce the decoding complexity by assessing the Hamming distance between an estimated codeword and the test-vector. Simulation results show that the proposed PPLCC decoding achieves an improved tradeoff between decoding complexity and latency over several benchmark decoding schemes.**

***Index Terms*—low-complexity Chase decoding, partially parallel decoding, Reed-Solomon codes**

I. Introduction

Reed-Solomon (RS) codes are widely used in communication and storage systems. Conventionally, they are decoded by the efficient Berlekamp-Massey (BM) algorithm [1]. The Guruswami-Sudan (GS) [2] algorithm and its soft-decision variant, the Kötter-Vardy (KV) [3] algorithm, can correct errors beyond the half distance bound by formulating the decoding as a curve-fitting problem. They yield a better performance but a higher decoding complexity. Such interpolation based decoding algorithms can be facilitated by various methods. E.g., the re-encoding transform [4] and the progressive interpolation [5] can reduce the interpolation computation and adapt the decoding complexity to the quality of received information, respectively. By identifying η unreliable received symbols, the low-complexity Chase (LCC) decoding [6] formulates 2^η test-vectors and yields a low interpolation complexity by exploiting the similarity among all test-vectors. Several variants of the LCC decoding have been proposed, including the hardware implementation friendly backward-forward LCC (BF-LCC) decoding [7] and the progressive LCC (PLCC) decoding [8].

In the LCC decoding, the 2^η test-vectors can be decoded in a fully parallel manner resulting in a low latency but a decoding complexity that has accommodated the redundant computation. Note that in the Chase decoding paradigm, only one out of all test-vectors can produce (by decoding) the intended codeword (or message). The PLCC decoding processes the test-vectors serially and terminates the decoding once the estimated codeword that satisfies the maximum likelihood (ML) criterion [9] is found. It can adapt the decoding complexity to the quality of received information. That says if the received information is less corrupted, the intended codeword can be found earlier, saving the unneccessary Chase decoding efforts. But its worst-case complexity remains the same as the LCC decoding. However, in practice, the decoding hardware can often support a certain degree of parallel processing. Hence, it yields room for the design of a partially parallel LCC decoding that can obtain a better tradeoff between the decoding complexity and latency. The prioritized interpolation which performs group-by-group decoding and employs an early termination based on the polynomial degree has been proposed in [10]. Within each group, the BF-LCC decoding is performed. A serial-parallel combined LCC decoding architecture has been proposed in [11]. It adjusts the parallel processing factors within the computation units to maintain the decoding latency at a certain level.

In order to reduce decoding complexity while maintaining a low decoding latency, this paper proposes the partially parallel LCC (PPLCC) decoding. The 2^η test-vectors will be categorized into several groups. Within each group, the test-vectors will be decoded in parallel. The decoding will be terminated once a codeword candidate that satisfies the ML criterion is found. Furthermore, a skipping rule reduces the decoding complexity. By assessing the Hamming distance between the estimated codewords and the test-vector, the redundant Chase decoding efforts are eliminated. The proposed PPLCC decoding is able to maintain both the decoding complexity and latency at a reduced level. Our simulation results show that the proposed decoding scheme achieves an improved tradeoff between decoding complexity and latency over several benchmark decoding schemes for RS codes.

II. Prerequisites

Let $\mathbb{F}_q = \{\sigma_0, \sigma_1, \dots, \sigma_{q-1}\}$ denote the finite field of size q. We consider an (n, k) RS code, where n and k are the length and dimension of the code, respectively. Given a message polynomial

$$f(x) = f_0 + f_1 x + \cdots + f_{k-1} x^{k-1}, \tag{1}$$

where $f_0, f_1, \dots, f_{k-1} \in \mathbb{F}_q$ are the message symbols, the codeword can be generated by

$$\underline{c} = (c_0, c_1, \dots, c_{n-1}) = (f(\alpha_0), f(\alpha_1), \dots, f(\alpha_{n-1})), \tag{2}$$

where $\alpha_0, \alpha_1, ..., \alpha_{n-1}$ are n distinct nonzero elements of $\mathbb{F}_q$.

In the interpolation based decoding, message polynomial $f(x)$ can be recovered by finding y-roots of the interpolation polynomial $Q(x,y) = \sum_{a,b} Q_{a,b} x^a y^b$ where $Q_{a,b} \in \mathbb{F}_q$. They can be organized under the (μ,ν)-reverse lexicographic (revlex) order, which is defined as follows. For a bivariate monormial $x^a y^b$, its (μ,ν)-weighted degree is $\deg_{\mu,\nu} x^a y^b = \mu a + \nu b$. Given $x^{a_1}y^{b_1}$ and $x^{a_2}y^{b_2}$, it is claimed $x^{a_1}y^{b_1} < x^{a_2}y^{b_2}$, if $\deg_{\mu,\nu} x^{a_1}y^{b_1} < \deg_{\mu,\nu} x^{a_2}y^{b_2}$, or $\deg_{\mu,\nu} x^{a_1}y^{b_1} = \deg_{\mu,\nu} x^{a_2}y^{b_2}$ and $b_1 < b_2$. If $x^{a'}y^{b'}$ is the leading monomial (LM) of Q with $Q_{a'b'} \neq 0$, the (μ,ν)-weighted degree of Q is $\deg_{\mu,\nu} Q = \deg_{\mu,\nu} x^{a'}y^{b'}$. Given two polynomials Q_1 and Q_2 with LM$(Q_1) = x^{a_1'}y^{b_1'}$ and LM$(Q_2) = x^{a_2'}y^{b_2'}$, respectively, $Q_1 < Q_2$ if LM$(Q_1) <$ LM(Q_2). In the proposed PPLCC decoding, re-encoding transform will be employed to reduce the interpolation complexity. As a result, $\mu = 1$ and $\nu = -1$. Polynomials are organized under the $(1,-1)$-revlex order.

III. The LCC and PLCC Decoding

This section revisits the LCC and the PLCC decoding. It starts with the test-vectors formulation.

A. Test-vectors Formulation

Assume that an RS codeword $\underline{c}$ is transmitted over a memoryless channel and $\underline{r} = (r_0, r_1, \ldots, r_{n-1}) \in \mathbb{R}^n$ is the received vector. The reliability matrix $\Pi \in \mathbb{R}^{q \times n}$ with entries $\pi_{i,j} = \Pr(r_j | c_j = \sigma_i)$ can be obtained, where $0 \le i \le q-1$ and $0 \le j \le n-1$. Let $i_j^{\mathrm{I}} = \arg\max_i \{\pi_{i,j}\}$ and $i_j^{\mathrm{II}} = \arg\max_{i, i \neq i_j^{\mathrm{I}}} \{\pi_{i,j}\}$. The two most likely decisions of codeword symbol c_j are $r_j^{\mathrm{I}} = \sigma_{i_j^{\mathrm{I}}}$ and $r_j^{\mathrm{II}} = \sigma_{i_j^{\mathrm{II}}}$. The following metric is defined to assess the reliability of r_j

$$\gamma_j = \frac{\pi_{i_j^{\mathrm{I}},j}}{\pi_{i_j^{\mathrm{II}},j}}, \tag{3}$$

where $\gamma_j \in [1, \infty)$. r_j is more reliable if γ_j is greater, and vice versa. By sorting γ_j in a descending order, a new symbol index sequence $j_0, j_1, \ldots, j_{n-1}$ is obtained. It indicates $\gamma_{j_0} \ge \gamma_{j_1} \ge \cdots \ge \gamma_{j_{n-1}}$. We define the index set of $n-\eta$ most reliable symbols as $\Theta = \{j_0, j_1, \ldots, j_{n-\eta-1}\}$, and its complementary set will be $\Theta^c = \{j_{n-\eta}, j_{n-\eta+1}, \ldots, j_{n-1}\}$. Therefore, all test-vectors can be written as

$$\underline{r}_u = (r_{j_0}^{(u)}, r_{j_1}^{(u)}, \ldots, r_{j_{n-1}}^{(u)}), \tag{4}$$

where $r_j^{(u)} = r_j^{\mathrm{I}}$ if $j \in \Theta$, and $r_j^{(u)} = r_j^{\mathrm{I}}$ or r_j^{II} if $j \in \Theta^c$, and $u = 0, 1, \ldots, 2^\eta - 1$. Since there are two decisions for each of the η unreliable symbols, 2^η test-vectors can be formulated.

B. Re-encoding Transform

The re-encoding transform can be employed to reduce the interpolation complexity. Let $\eta \le n-k$ so that all test-vectors share at least k common symbols. The index set of the k most reliable symbols is defined as $\Psi = \{j_0, j_1, \ldots, j_{k-1}\}$ and its complementary set will be $\Psi^c = \{j_k, j_{k+1}, \ldots, j_{n-1}\}$. A re-encoding codeword $\underline{h} = (h_0, h_1, \ldots, h_{n-1})$ can be generated by setting $h_j = r_j^{\mathrm{I}}, \forall j \in \Psi$ and determining the remaining symbols of Ψ^c by erasure decoding [4]. Consequently, all test-vectors can be transformed by

$$z_j^{(u)} = r_j^{(u)} - h_j, \forall j. \tag{5}$$

In a reliability sorted order, the transformed test-vectors can be written as

$$\underline{z}_u = (0, 0, \ldots, 0, z_{j_k}^{(u)}, \ldots, z_{j_{n-\eta}}^{(u)}, \ldots, z_{j_{n-1}}^{(u)}). \tag{6}$$

C. Common Element Interpolation

For a test-vector $\underline{z}_u$, interpolation is to construct a minimal polynomial $g(x,y)$ that satisfies $g(\alpha_j, z_j^{(u)}) = 0, \forall j$. The above description shows that all test-vectors share $n-\eta$ common interpolation points $(\alpha_j, z_j^{(u)}), \forall j \in \Theta$. With the re-encoding transform, the first k points become $(\alpha_j, 0)$, where $j \in \Psi$. Their interpolation property can be established by polynomial $\prod_{j \in \Psi}(x - \alpha_j)$. Let $\Psi' = \Psi^c \backslash \Theta^c = \{j_k, j_{k+1}, \cdots, j_{n-\eta-1}\}$ and $(\alpha_j, z_j^{(u)})$, where $j \in \Psi'$, are the remaining common points that all test-vectors share. In the common element interpolation, the $n-\eta-k$ points need to be interpolated by Kötter's interpolation [12]. The interpolation polynomial set is initialized as

$$\mathcal{G} = \{g_0(x,y), g_1(x,y)\} = \{1, y\}. \tag{7}$$

Let $i^* = \arg\min_i \{g_i(x,y) | g_i(\alpha_j, z_j^{(u)}) \neq 0\}$, $g^*(x,y) = g_{i^*}(x,y)$ and $\rho(x,y) = \mathcal{G} \backslash g^*(x,y)$. To interpolate $(\alpha_j, z_j^{(u)})$, $g^*(x,y)$ and $\rho(x,y)$ will be updated as

$$\rho'(x,y) = g^*(\alpha_j, z_j^{(u)}) \cdot \rho(x,y) - \rho(\alpha_j, z_j^{(u)}) \cdot g^*(x,y), \tag{8}$$

$$g^{*\prime}(x,y) = g^*(\alpha_j, z_j^{(u)}) \cdot (x - \alpha_j) \cdot g^*(x,y). \tag{9}$$

The above updates will be performed with the $n-\eta-k$ points. At the end, an updated polynomial set $\tilde{\mathcal{G}} = \{g_i(x,y) | g_i(\alpha_j, z_j^{(u)}) = 0, \forall j \in \Psi'\}$ can be obtained, based on which the following uncommon element interpolation is performed.

D. Uncommon Element Interpolation

The uncommon element interpolation is performed for test-vector $\underline{z}_u$ by interpolating η points $(\alpha_j, z_j^{(u)}), \forall j \in \Theta^c$ with the interpolation operations defined by (8) - (9). Let $\mathcal{G}_{\mathfrak{w}}^{(u)}$ denote the uth polynomial set obtained by interpolating $\mathfrak{w}$ points. When $\mathfrak{w} = 0$, $\mathcal{G}_0^{(u)} = \tilde{\mathcal{G}}$ as all test-vectors inherit the common element result. After η points are interpolated, 2^η polynomial sets $\mathcal{G}_\eta^{(u)} = \{g_i^{(u)}(x,y) | g_i^{(u)}(\alpha_j, z_j^{(u)}) = 0, \forall j \in \Psi^c\}$ are obtained. $\mathcal{G}_\eta^{(u)}$ is the interpolation outcome w.r.t. test-vector $\underline{z}_u$. The interpolation polynomial for each test-vector can be chosen by

$$g^{(u)}(x,y) = \min\{g_0^{(u)}(x,y), g_1^{(u)}(x,y)\}. \tag{10}$$

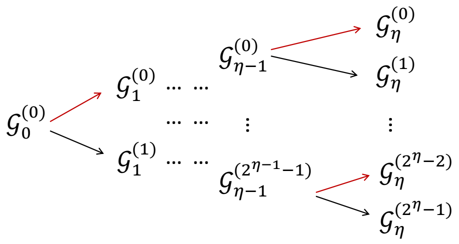
The estimated message polynomial $\hat{f}^{(u)}(x)$ can be recovered by finding y-roots of $g^{(u)}(x,y)$ where $g^{(u)}(x,y) = Q_0^{(u)}(x) + Q_1^{(u)}(x) \cdot y$. Hence $\hat{f}^{(u)}(x)$ can be determined by

$$\hat{f}^{(u)}(x) = -\frac{V(x) \cdot Q_0^{(u)}(x)}{Q_1^{(u)}(x)}, \tag{11}$$

where $V(x) = \prod_{j \in \Psi} (x - \alpha_j)$. Since the re-encoding transform is applied, the error magnitudes in Ψ can be determined by

$$\hat{e}_j^{(u)} = -\hat{c}_j^{(u)} + h_j = -\hat{f}^{(u)}(\alpha_j), \tag{12}$$

where $j \in \Psi$. With the error magnitudes, errors in Ψ can be corrected. By using erasure decoding and inversing re-encoding transform, the remaining symbols of the estimated codeword $\underline{\hat{c}}^{(u)}$ will be obtained. After decoding all 2^η test-vectors, at most 2^η distinct codeword candidates will be obtained. Based on $\underline{r}$, the most likely candidate and its corresponding message will be selected as the decoding outputs.



$\mathcal{G}_0^{(0)}$
$\mathcal{G}_1^{(0)}$
$\mathcal{G}_1^{(1)}$
$\mathcal{G}_{\eta-1}^{(0)}$
$\mathcal{G}_{\eta-1}^{(2^{\eta-1}-1)}$
$\mathcal{G}_\eta^{(0)}$
$\mathcal{G}_\eta^{(1)}$
$\mathcal{G}_\eta^{(2^\eta-2)}$
$\mathcal{G}_\eta^{(2^\eta-1)}$

Fig. 1. Binary tree representation of the uncommon element interpolation.

The above mentioned processes constitute the LCC decoding [6], where its uncommon element interpolation can be interpreted as a binary tree growing process in a layer-by-layer manner as illustrated in Fig. 1. Since all test-vectors are decoded in parallel, besides low complexity, low decoding latency can also be ensured.

E. The Progressive Variant

As a progressive variant of the above mentioned LCC decoding, the PLCC decoding decodes all test-vectors in a sequential manner, prioritizing to decode the more reliable test-vectors. Given a test-vector $\underline{r}_u = (r_0^{(u)}, r_1^{(u)}, \ldots, r_{n-1}^{(u)})$, its reliability can be determined by

$$\Omega_u = \prod_{j=0}^{n-1} \pi_{i_j^{(u)},j}, \tag{13}$$

where $i_j^{(u)} = \text{index}\{\sigma_i | \sigma_i = r_j^{(u)}\}$. The test-vector with a larger Ω_u is considered to be more reliable. It is more likely to produce the intended message and should be decoded earlier. Since all test-vectors share the common symbols r_j^{I}, where $j \in \Theta$, the reliability function can be simplified into

$$\hat{\Omega}_u = \prod_{j \in \Theta^c} \pi_{i_j^{(u)},j}. \tag{14}$$

Based on $\hat{\Omega}_u$, order all transformed test-vectors yielding $\underline{z}_{v_0}, \underline{z}_{v_1}, \ldots, \underline{z}_{v_{2^\eta-1}}$ where $\hat{\Omega}_{v_0} \geq \hat{\Omega}_{v_1} \geq \cdots \geq \hat{\Omega}_{v_{2^\eta-1}}$. Note that $\underline{z}_{v_0}$ is the hard-decision received codeword. The decoding will process in the above order. If an estimated codeword that satisfies the ML criterion stated as *Lemma 1′* of [9] is found, the decoding will be terminated. More detailed descriptions of the ML criterion can be referred to the Appendix A of [13]. With this ML codeword assessment, the decoding can be terminated earlier, especially when the received vector is not heavily corrupted. As a result, the decoding complexity can be significantly reduced. Unlike the LCC decoding, the PLCC decoding grows the binary tree in a depth-first-search manner [8]. During decoding the test-vectors, the intermediate interpolation information will be stored. The decoding of a test-vector will not start from $\mathcal{G}_0^{(0)}$ but the nearest available node which stores the intermediate interpolation information.

It has been observed that the codeword that satisfies the ML criterion can often be identified by decoding the first few test-vectors, especially when the channel condition is good. Table I illustrates this property of the ML criterion. The $(248, 216)$ RS code, which is often used in optical communication networks, is simulated. These results were obtained by using the PLCC decoding with $\eta = 8$ over the additive white Gaussian noise (AWGN) channel with BPSK modulation. We use N_{ET} to denote the number of test-vectors which have been decoded when an early termination occurs. Table I shows that early termination occurs more frequently as the signal-to-noise ratio (SNR) increases. When the SNR is 6 dB, majority of the Chase decoding outputs produce the ML codeword by only processing the hard-decision received codeword.

TABLE I
PERCENTAGE OF EARLY TERMINATION IN DECODING THE $(248, 216)$ RS CODE WITH $\eta = 8$

SNR (dB)	$N_{\mathrm{ET}} = 1$	$2 \leq N_{\mathrm{ET}} \leq 6$	$N_{\mathrm{ET}} \geq 7$	Failed to Satisfy the ML Criterion
4.5	0.17%	0.01%	0.07%	99.76%
5	10.85%	0.18%	0.12%	88.85%
5.5	69.68%	0.46%	0.10%	29.76%
6	98.85%	0.08%	0.01%	1.06%

IV. The PPLCC Decoding

This section proposes the PPLCC decoding. For the LCC decoding, fully parallel processing of all test-vectors is reliazed to achieve low decoding latency. But the decoding complexity is rather high since all test-vectors should be decoded. In contrast, the PLCC decoding has a lower decoding complexity thanks to its serial decoding manner and the early termination. But its worst-case latency can be much higher than the LCC decoding. In order to achieve an improved tradeoff between the decoding complexity and latency, the PPLCC decoding is proposed. In practice, decoding circuits can often support a certain degree of parallelism, providing the resources to decode a portion of test-vectors in parallel. To exploit this possibility, one straightforward approach is to divide the test-vectors into multiple groups and the test-vectors within each group are decoded in parallel. The test-vectors grouping is described as follows.

A. Test-vectors Grouping

Let P denote the maximum degree of parallelism indicating that at most P test-vectors can be decoded in parallel. The test-vectors are divided into M groups denoted as $\mathfrak{s}_1, \mathfrak{s}_2, \ldots, \mathfrak{s}_M$.

Let $\underline{z}_{\kappa,m}$ denote the mth test-vector of $\mathfrak{s}_\kappa$ and N_κ denote the number of test-vectors in group $\mathfrak{s}_\kappa$, respectively, where $\kappa = 1, 2, \ldots, M$. Hence, $\mathfrak{s}_\kappa = \{\underline{z}_{\kappa,1}, \underline{z}_{\kappa,2}, \ldots, \underline{z}_{\kappa,N_\kappa}\}$. The test-vectors can be grouped with the fixed parallelism or the progressive parallelism.

1) Fixed Parallelism Grouping: In this case, all groups contain the same number of test-vectors, i.e., $M = 2^\eta/P$, and the degree of parallelism is

$$N_\kappa = P. \tag{15}$$

Note that in this work it is assumed that $P|2^\eta$.

2) Progressive Parallelism Grouping: Alternatively, the groups can contain different number of test-vectors. Let Δ denote the number of test-vectors in $\mathfrak{s}_2$ and $\mathcal{W} = (1+\Delta)/P$ where $0 < \Delta \leq P-1$. N_κ can also be chosen based on the observation of Table I and $M = 2^\eta/P + \lceil 2 - \mathcal{W} \rceil$. Hence, the test-vectors will be grouped in a progressive manner

$$N_\kappa = \begin{cases} 1, & \text{if} \quad \kappa = 1; \\ \Delta, & \text{if} \quad \kappa = 2; \\ (1 + \lfloor \mathcal{W} \rfloor) \cdot P - 1 - \Delta, & \text{if} \quad \kappa = 3; \\ P, & \text{if} \quad 3 < \kappa \leq M. \end{cases} \tag{16}$$

Therefore, the first group only contain the hard-decision received codeword. The rest of the Δ most reliable test-vectors are contained in the second group. Based on the observation of Table I, we can reliaze that if the codeword that satisfies the ML criterion can be found, they are far more likely to be found by decoding the first few most reliable test-vectors. Hence, they should be decoded in the earlier groups. In order to compromise the decoding complexity, Δ should not be too large. For the third group, there are two conditions. If $(1+\Delta)$ is smaller than P, i.e. $\mathcal{W} < 1$, the third group contains $P - 1 - \Delta$ test-vectors. Otherwise, it contains P test-vectors, i.e., $(1 + \lfloor \mathcal{W} \rfloor) \cdot P - 1 - \Delta = P$. Since the decoding outputs produced by the remaining test-vectors rarely satisfy the ML criterion, the degree of parallelism will be improved to reduce the decoding latency. Hence, each group contains P test-vectors for $3 < \kappa \leq M$.

B. The Skipping Rule

A skipping rule is further introduced to reduce the decoding complexity. Based on the Hamming distance between a decoded codeword and the test-vector, the redundant Chase decoding efforts can be eliminated [14]. Let $d_{\mathrm{H}}(\underline{\mathfrak{r}}_1, \underline{\mathfrak{r}}_2)$ denote the Hamming distance between two equal length vectors $\underline{\mathfrak{r}}_1$ and $\underline{\mathfrak{r}}_2$, both of which are defined over $\mathbb{F}_q$. Let τ denote the error-correcting radius of the decoding scheme. Note that in the discussed interpolation based Chase decoding events, the interpolation is conducted with a multiplicity of one over all points. It yields an error-correction capability of $\tau = \lfloor (n-k)/2 \rfloor$.

Lemma 1. Given a decoded codeword $\underline{\hat{c}}^{(u^*)}$ in the decoding output list, where the radius of the Hamming sphere is τ, and a test-vector $\underline{z}_u$, if $d_{\mathrm{H}}(\underline{\hat{c}}^{(u^*)}, \underline{z}_u) \leq \tau$, $\underline{z}_u$ can be skipped.

Proof. Assume that the test-vector $\underline{z}_{u^*}$ produces a codeword $\underline{\hat{c}}^{(u^*)}$. If $\underline{z}_u$ lies within the Hamming sphere of $\underline{\hat{c}}^{(u^*)}$, i.e. $d_{\mathrm{H}}(\underline{\hat{c}}^{(u^*)}, \underline{z}_u) \leq \tau$, it will be decoded as $\underline{\hat{c}}^{(u^*)}$. It indicates that the decoding of $\underline{z}_u$ will produce the same outcome as the decoding of $\underline{z}_{u^*}$. Therefore, there is no need to decode $\underline{z}_u$.□

After decoding $\mathfrak{s}_\kappa$, we will filter the test-vectors in $\mathfrak{s}_{\kappa+1}$ by comparing them with the decoded codewords. This helps reduce the decoding complexity significantly. Note that the skipping rule in *Lemma 1* and that in [14] are different. In our method, once the Hamming distance between $\underline{z}_u$ and any decoded codeword is not greater than τ, it will be determined to be skipped and there is no need to compare it with the whole decoding output list, while the complete comparison between $\underline{z}_u$ and all decoded codewords is done in [14].

Algorithm 1 The PPLCC Decoding Algorithm

```
Input: $\underline{r}$, $\eta$, $P$, $\Pi$, $N_1, N_2, \ldots, N_\kappa$;
Output: $\underline{\hat{c}}$;
1:  Formulate $2^\eta$ test-vectors $\underline{r}_u$ as in (4);
2:  Perform the re-encoding transform to yield $\underline{z}_u$ as in (6);
3:  Calculate $\hat{\Omega}_u$ for each test-vector as in (14);
4:  Re-order $\underline{z}_0, \underline{z}_1, \ldots, \underline{z}_{2^\eta-1}$ as $\underline{z}_{v_0}, \underline{z}_{v_1}, \ldots, \underline{z}_{v_{2^\eta-1}}$, such
    that $\hat{\Omega}_{v_0} \geq \hat{\Omega}_{v_1} \geq \cdots \geq \hat{\Omega}_{v_{2^\eta-1}}$;
5:  Initialize $\mathfrak{s}_1$ as in (15) or (16);
6:  Let $\xi = N_1$;
7:  for $\kappa = 1$ to $M$ do
8:      Decode the test vectors of $\mathfrak{s}_\kappa$;
9:      Update the decoding output list;
10:     if there is a decoded codeword satisfies the ML criterion
11:         Terminate the decoding and output this codeword as
            $\underline{\hat{c}}$;
12:     if $\xi \geq 2^\eta$
13:         Terminate the decoding;
14:     for $m = 1$ to $N_{\kappa+1}$ do
15:         while $\underline{z}_{v_\xi}$ satisfies Lemma 1 and $\xi < 2^\eta$ do
16:             Skip decoding $\underline{z}_{v_\xi}$ and $\xi = \xi + 1$;
17:         end while
18:         if $\xi < 2^\eta$
19:             $\underline{z}_{\kappa+1,m} = \underline{z}_{v_\xi}$ and $\xi = \xi + 1$;
20:     end for
21: end for
22: The most likely codeword candidate will be selected as $\underline{\hat{c}}$.
```

C. The Algorithm

The above description shows that the test-vectors can be grouped and decoded in a partially parallel manner, yielding priority to decode the more reliable test-vectors to achieve early termination. The decoding can also be facilitated by the skipping rule which helps eliminate the redundant decoding computations. First, the 2^η test-vectors $\underline{r}_u$ will be formulated and transformed to $\underline{z}_u$ by the re-encoding approach. By sorting their reliabilities, $\underline{z}_0, \underline{z}_1, \ldots, \underline{z}_{2^\eta-1}$ will be re-ordered as $\underline{z}_{v_0}, \underline{z}_{v_1}, \ldots, \underline{z}_{v_{2^\eta-1}}$. The first group $\mathfrak{s}_1$ will be initialized with fixed or progressive parallelism grouping as in (15) or

(16), respectively. N_1 test-vectors in $\mathfrak{s}_1$ will be decoded. After decoding a group, decoding output list will be updated. The decoded codewords will be validated with the ML criterion to check if an ML codeword has been found. If so, the decoding will be terminated. If not, the decoding will continue to process the next group. Let ξ denote the number of test-vectors which have been decoded (or skipped). The N_2 test-vectors will be filtered from sorted test-vectors $\underline{z}_{v_\xi}, \underline{z}_{v_{\xi+1}}, \ldots, \underline{z}_{v_{2^\eta-1}}$ by the skipping rule. If the Hamming distance between a test-vector $\underline{z}_{v_\xi}$ and any codeword in the decoding output list is not greater than τ, this test-vector will be skipped. Otherwise, it will remain in $\mathfrak{s}_2$ to be decoded. The above decoding procedure will perform for the M groups in a group-by-group manner. Note that the decoding within a group are performed in parallel and the intermediate interpolation information will be stored. This is similar to the PLCC decoding, while interpreted over the binary interpolation tree of Fig. 1, the decoding of a test-vector will start from the nearest available node over the tree. The PPLCC decoding is summarized as in Algorithm 1.

V. Simulation Results

This section shows the decoding complexity, latency and performance of proposed PPLCC decoding. The decoding complexity is measured as the average number of finite field multiplications in a decoding event. The decoding latency is measured as either the average running time required to decode a codeword or the average number of decoded groups when decoding terminates. The decoding performance is measured as the frame error rate (FER). Our results were obtained over the AWGN channel using BPSK modulation. The LCC, the PLCC and the PLCC with the skipping rule (marked as PLCC(SR)) decoding are used as the comparison benchmarks. For the PPLCC decoding, simulation results of both the fixed parallelism grouping (marked as PPLCC (Fixed)) and the progressive parallelism grouping (marked as PPLCC (Progressive)) are shown. For the latter, $\Delta = 7$ when $P = 8$, and $\Delta = 8$ when $P > 8$. Note that PLCC (SR), PPLCC (Fixed) and PPLCC (Progressive) are facilitated by the skipping rule.

A. Decoding Complexity

Table II compares the complexity of related schemes in decoding the $(248, 216)$ RS code with $\eta = 8$ and $P = 16$. Table II shows that the PLCC (SR) decoding is less complex than the PLCC decoding since redundant test-vectors decoding computations have been eliminated with the skipping rule. The PPLCC (Fixed) and the PPLCC (Progressive) decoding achieve a much lower complexity compared with the LCC decoding due to their partially parallel decoding architecture. The PPLCC (Progressive) decoding outperforms the PPLCC (Fixed) decoding since it can further utilize the property of the ML criterion, which was also demonstrated by Table I.

Table III shows the percentage reduction in complexity achieved by the PPLCC (Progressive) decoding over the LCC and the PPLCC (Fixed) decoding with $P = 8, 16, 32$. The complexity reduction percentage is calculated as $((\mathcal{C}_{\text{PPLCC(Prog.)}} - \mathcal{C}_{\text{benchmark}})/\mathcal{C}_{\text{benchmark}}) \times 100\%$, where $\mathcal{C}_{\text{PPLCC(Prog.)}}$ and $\mathcal{C}_{\text{benchmark}}$ denote the decoding complexity of the PPLCC (Progressive) decoding and benchmark schemes, respectively. With different P, the PPLCC (Progressive) decoding is able to reduce the decoding complexity significantly. This advantage becomes more obvious as the SNR increases, where early termination occurs more frequently.

TABLE II
Complexity comparison in decoding the $(248, 216)$ RS code with $\eta = 8$ and $P = 16$

SNR (dB)	LCC	PLCC	PLCC (SR)	PPLCC (Fixed)	PPLCC (Progressive)
5.7	3.87×10^6	8.03×10^5	2.39×10^5	4.68×10^5	2.65×10^5
5.9	3.72×10^6	2.93×10^5	1.07×10^5	3.19×10^5	1.27×10^5
6.1	3.57×10^6	1.20×10^5	7.85×10^4	2.84×10^5	8.04×10^4
6.3	3.43×10^6	7.66×10^4	7.14×10^4	2.69×10^5	7.21×10^4

TABLE III
Complexity reduction of the PPLCC (Progressive) in decoding the $(248, 216)$ RS code with $\eta = 8$ and $P = 8, 16, 32$

SNR (dB)	Complexity Reduction over the LCC			Complexity Reduction over the PPLCC (Fixed)		
	$P = 8$	$P = 16$	$P = 32$	$P = 8$	$P = 16$	$P = 32$
5.7	-93.04%	-93.15%	-93.13%	-19.34%	-43.29%	-61.11%
5.9	-96.79%	-96.57%	-96.78%	-43.58%	-60.12%	-78.38%
6.1	-97.66%	-97.75%	-97.78%	-51.97%	-71.71%	-84.21%
6.3	-97.91%	-97.90%	-97.88%	-56.05%	-73.22%	-84.86%

B. Decoding Latency

The decoding latency can be measured as the average running time required to decode a codeword. Since the parallel decoding operations are driven by the same clock, we can also use the average number of decoded groups when decoding terminates to assess the latency performance in another perspective. Table IV shows the average running time required to decode a codeword and the average number of decoded groups when decoding terminates of related schemes in decoding the $(248, 216)$ RS code with $\eta = 8$ and $P = 16$. We use t_{rt} and N_{dg} to denote the average running time and the average number of decoded groups, respectively. These results were obtained by implementing the decoding schemes using the C programming language and on the Intel core i7-11700 CPU. Note that for the LCC and the PPLCC decoding schemes, the average running time is estimated by dividing the total running time by the degree of parallelism. Table IV shows that the LCC decoding which decodes all test-vectors in a fully parallel manner achieves the lowest running time and decodes the least groups since all test-vectors are seen as in one group. The PLCC decoding which performs decoding serially yields the highest running time and decodes the largest number of groups. The reduction in the number of decoded groups yielded by the PLCC (SR) decoding over the PLCC decoding shows the number of unnecessary test-vectors reduced by the skipping rule. The PPLCC (Progressive) decoding outperforms the PLCC and the PLCC (SR) decoding. But its latency is slightly higher than the PPLCC (Fixed) decoding since it may decode one or two more groups of test-vectors.

TABLE IV
AVERAGE RUNNING TIME (MS) AND AVERAGE NUMBER OF DECODED GROUPS IN DECODING THE $(248, 216)$ RS CODE WITH $\eta = 8$ AND $P = 16$

SNR (dB)	LCC		PLCC		PLCC (SR)		PPLCC (Fixed)		PPLCC (Progressive)	
	t_{rt}	N_{dg}	t_{rt}	N_{dg}	t_{rt}	N_{dg}	t_{rt}	N_{dg}	t_{rt}	N_{dg}
5.7	8.1	1.0	86.3	46.1	54.6	11.0	11.2	1.9	17.8	2.2
5.9	8.2	1.0	35.0	14.5	20.1	3.0	9.3	1.2	12.3	1.4
6.1	8.2	1.0	13.4	3.9	10.8	1.3	8.2	1.0	9.4	1.1
6.3	7.9	1.0	9.2	1.4	8.9	1.0	7.8	1.0	8.9	1.0

Table V shows the percentage reduction in the number of decoded groups yielded by the PPLCC (Progressive) decoding over the PLCC (SR) and the PPLCC (Fixed) decoding with $P = 8, 16, 32$. The decoded groups reduction percentage is calculated as $((\mathcal{N}_{\mathrm{PPLCC(Prog.)}} - \mathcal{N}_{\mathrm{benchmark}})/\mathcal{N}_{\mathrm{benchmark}}) \times 100\%$, where $\mathcal{N}_{\mathrm{PPLCC(Prog.)}}$ and $\mathcal{N}_{\mathrm{benchmark}}$ denote the decoded groups of the PPLCC (Progressive) decoding and benchmark schemes, respectively. It is obvious that the PPLCC (Progressive) decoding is able to reduce decoded groups in comparison with the PLCC (SR) decoding. The PPLCC (Progressive) decoding decodes a slightly higher number of groups over the PPLCC (Fixed) decoding. However, their differences diminish as the SNR increases since early termination occurs by only processing the hard-decision received codeword.

TABLE V
DECODED GROUPS REDUCTION OF THE PPLCC (PROGRESSIVE) IN DECODING THE $(248, 216)$ RS CODE WITH $\eta = 8$ AND $P = 8, 16, 32$

SNR (dB)	Decoded Groups Reduction over the PLCC (SR)			Decoded Groups Reduction over the PPLCC (Fixed)		
	$P = 8$	$P = 16$	$P = 32$	$P = 8$	$P = 16$	$P = 32$
5.7	-74.30%	-86.92%	-83.29%	16.94%	23.81%	27.78%
5.9	-51.55%	-54.32%	-58.45%	7.14%	15.78%	11.97%
6.1	-17.58%	-20.35%	-21.80%	4.25%	2.58%	2.43%
6.3	-1.58%	-1.28%	-1.35%	0.38%	0.71%	0.67%

C. Decoding Performance

Fig. 2 shows FER performance of the $(248, 216)$ RS code. The Chase decoding schemes with $\eta = 8$ outperform the BM decoding. They yield the same performance since their error correcting capabilities are mainly determined by η. Revisiting Tables II - V, the PPLCC (Progressive) decoding achieves the best tradeoff between decoding complexity and latency over other benchmark schemes. It maintains the same LCC decoding performance.

VI. CONCLUSION

This paper has proposed the PPLCC decoding for RS codes. The Kötter's interpolation based Chase decoding events process the formulated test-vectors in a partially parallel manner. The decoding will be terminated once a codeword candidate that satisfies the ML criterion is found. Furthermore,

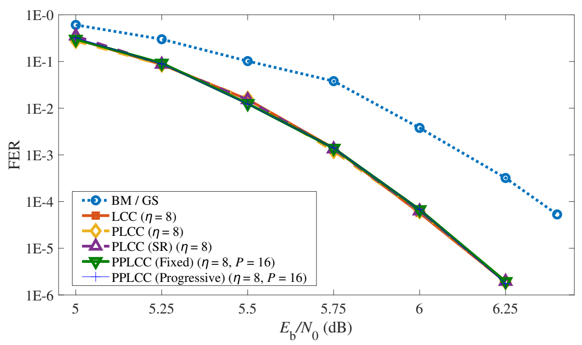


Fig. 2. Decoding performance of the $(248, 216)$ RS code.

a skipping rule has been introduced to reduce the decoding complexity by assessing the Hamming distance between a decoded codeword and the test-vector. Our simulation results have shown that the proposed PPLCC decoding scheme achieves an improved tradeoff between decoding complexity and latency over several benchmark Chase decoding schemes.

ACKNOWLEDGEMENT

This work is sponsored by National Natural Science Foundation of China (NSFC) with project ID 62071498.

REFERENCES

[1] J. Massey, "Shift register synthesis and BCH decoding," *IEEE Trans. Inform. Theory*, vol. 15, no. 1, pp. 122–127, Jan. 1969.

[2] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and algebraic-geometric codes," *IEEE Trans. Inform. Theory*, vol. 45, no. 6, pp. 1757–1767, Sept. 1999.

[3] R. Kötter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Inform. Theory*, vol. 49, no. 11, pp. 2809–2825, Nov. 2003.

[4] R. Kötter *et al.*, "The re-encoding transformation in algebraic list-decoding of Reed-Solomon codes," *IEEE Trans. Inform. Theory*, vol. 57, no. 2, pp. 633–647, Feb. 2011.

[5] L. Chen *et al.*, "Progressive algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Commun.*, vol. 61, no. 2, pp. 433–442, Feb. 2013.

[6] J. Bellorado and A. Kavcic, "Low-complexity soft-decoding algorithms for Reed-Solomon codes - part I: an algebraic soft-in hard-out Chase decoder," *IEEE Trans. Inform. Theory*, vol. 56, no. 3, pp. 945–959, Mar. 2010.

[7] J. Zhu *et al.*, "Backward interpolation architecture for algebraic soft-decision Reed-Solomon decoding," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 17, no. 11, pp. 1602–1615, Nov. 2009.

[8] J. Zhao *et al.*, "Progressive algebraic Chase decoding algorithms for Reed-Solomon codes," *IET Commun.*, vol. 10, no. 12, pp. 1416–1427, 2016.

[9] T. Kaneko *et al.*, "An efficient maximum-likelihood-decoding algorithm for linear block codes with algebraic decoder," *IEEE Trans. Inform. Theory*, vol. 40, no. 2, pp. 320–327, Mar. 1994.

[10] X. Zhang *et al.*, "Modified low-complexity Chase soft-decision decoder of Reed–Solomon codes," *J. Sig. Proc. Syst.*, vol. 66, no. 1, pp. 3-13, Jan. 2012.

[11] W. Zhang *et al.*, "Low-power high-efficiency architecture for low-complexity Chase soft-decision Reed–Solomon decoding," *IET Commun.*, vol. 6, no. 17, pp. 3046–3052, 2012.

[12] R. Kötter, *On algebraic decoding of algebraic-geometric and cyclic codes*. PhD Thesis, Univ. Linköping, Linköping, Sweeden, 1996.

[13] J. Xing *et al.*, "Progressive algebraic soft-decision decoding of Reed-Solomon codes using module minimization," *IEEE Trans. Commun.*, vol. 67, no. 11, pp. 7379–7391, Nov. 2019.

[14] H. Lee and Y. Chen, "Efficient Chase-2 decoding algorithm for linear block codes," *Electron. Lett.*, vol. 55, no. 17, pp. 936–938, 2019.