Low-Complexity Parallel Min-Sum Medium-Density Parity-Check Decoder for McEliece Cryptosystem

Jiaxuan Cai, *Graduate Student Member, IEEE* and Xinmiao Zhang, *Senior Member, IEEE*

***Abstract*—The McEliece cryptosystem based on medium-density parity-check (MDPC) codes remains a candidate in the fourth round submission of post-quantum cryptography standard. The low-density parity-check (LDPC) decoders used in digital communications have been extensively studied. However, the MDPC codes for the McEliece cryptosystem have much higher column weight and different structure in their parity-check matrices. As a result, simplification techniques for LDPC decoders are not applicable to MDPC decoders. Besides, existing MDPC decoder designs have been focusing on the simplest bit-flipping algorithm, whose performance is inferior compared to that of the Min-sum algorithm. This paper first optimizes the scaled Min-sum algorithm for codes with high column weight to improve the performance with simple scalar multiplications. The overall decoder architecture is re-designed to take into account the sparsity of the parity-check matrix and nontrivial min-sum check node processing. Besides, a flexible message storage scheme is proposed to reduce the worst-case decoding latency of the randomly constructed codes utilized in the McEliece cryptosystem. Then a 2-stage scaling scheme is developed to reduce the long critical path caused by the high column weight and a group size re-balancing scheme is introduced to mitigate the precision loss caused by the 2-stage scaling in parallel decoders. For an example MDPC decoder, the proposed optimized 2-stage scaled Min-sum algorithm leads to orders of magnitude error-correcting performance improvement and 16% higher clock frequency with negligible silicon area overhead compared to unoptimized Min-sum decoders.**

***Index Terms*—McEliece cryptosystem, error-correcting codes, medium-density parity-check codes, Min-sum algorithm, parallel decoder, post-quantum cryptography.**

I. Introduction

SIGNIFICANT progress has been made on quantum processor development recently. To address the imminent need for cryptography schemes resisting quantum computing attacks, the National Institute of Standards and Technology (NIST) called for post-quantum cryptography standardization recently. The fourth round submission has been announced in July 2022, and the McEliece cryptosystem utilizing quasi-cyclic medium-density parity-check (QC-MDPC) codes [1], [2] remains one of the candidates. Although low-density parity-check (LDPC) codes for error correction in digital communication and storage systems have been well-studied, their parity-check matrices consist of an array of many smaller zero and cyclic permutation matrices (CPMs), have low column weight, and are carefully designed to avoid short cycles [3]. On the other hand, the parity-check matrices of the MDPC codes considered for post-quantum cryptography standard have a single row of 2, 3, or 4 randomly-constructed very large circulant matrices of much higher column weight. Due to the high column weight and irregularity, the decoding algorithms extended from those of LDPC codes need to be re-optimized and decoder architectures need to be re-designed.

MDPC codes can be decoded by various algorithms. The bit-flipping (BF) algorithm and its variations are the simplest and have been considered in previous literature. The bits can be flipped according to different strategies [4]. MDPC decoding algorithms with very a small number of decoding iterations have been developed for BIKE [5], which is another candidate of post-quantum cryptography. However, these algorithms have inferior performance compared to those in [4] when more decoding iterations are allowed. The error-correction bounds and error floors for BF algorithms were studied in [6] and [7].

In BF MDPC decoder implementations, the majority of the complexity is contributed by the memories. To reduce the sizes of the memories, the decoder can store only the first column of each circulant in the QC-MDPC parity check matrix. Then cyclical shifting is carried out to derive the other columns. The decoder in [8] considers FPGA implementations. 32 bits in a column are processed in each clock cycle and the cyclically shifted column is stored back to BRAMs using an optimized method. To reduce the decoding latency, the syndromes are updated after every bit flipping instead of once only at the end of each decoding iteration in [9]. Whether to flip a bit is usually decided according to the syndrome weight, whose computation is approximated in [10] to reduce the hardware complexity. Even though the parity check matrices of MDPC codes have higher density than those of LDPC codes, they are still very sparse. The sparsity is further exploited in [11] to reduce the decoding latency and silicon area. Besides, the columns are processed in an out-of-order manner to reduce memory writes.

MDPC decoding algorithms with better error-correcting capabilities not only lower the probability of decryption failure but also help to thwart various attacks, such as the reaction attack [12], [13], that try to recover the secret parity-check matrix by utilizing decoding failures. The Min-sum decoding algorithms [14] have excellent tradeoff on performance and complexity and can achieve substantial coding gain over BF algorithms. The scaled version of the Min-sum algorithm multiplies the sum of the check-to-variable (c2v) messages by a scalar before it is added to the decoder input probability message in the variable node processing step of the decoding. A properly selected scalar leads to further coding gain. The

This material is based upon work supported by the National Science Foundation under Award No. 2052641. The authors are with the Department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH 43210 USA (e-mail: cai.1072@osu.edu; zhang.8952@osu.edu).

scaled Min-sum algorithm has been well-studied for the LDPC codes used in communication and storage systems. However, it needs to be re-investigated and the decoder needs to be re-designed for the MDPC codes due to the much higher column weight. Besides, the check and variable node processing steps in the Min-sum algorithm have much higher complexity compared to their counterparts in BF decoders. As a result, previous methodologies for parallel BF decoding no longer leads to efficient design.

For the first time, this paper investigates the Min-sum decoding algorithm and decoder design for the MDPC codes used in the McEliece cryptosystem. The contributions of this paper include the following: i) The scaled Min-sum decoding algorithm is optimized to improve the error-correcting performance of MDPC codes with low-complexity scalar multipliers; ii) An efficient parallel decoder architecture is developed for the Min-sum algorithm for randomly constructed MDPC codes taking into account both the non-trivial check node processing and the irregular sparse parity-check matrix. To shorten the decoding latency, a flexible message storage scheme is also developed and the decoder can choose from one of the two storage schemes on the fly for a given MDPC code. iii) The high column weight of MDPC codes leads to long critical path in the c2v message accumulation of the variable node processing step. To reduce the critical path, this paper proposes to divide the c2v messages into groups and carry out message scaling in two stages. A systematic procedure is also developed to decide the group size and other parameters of the scaling for a given critical path goal; iv) For parallel decoders, the 2-stage scaling leads to error-correcting performance degradation due to finite precision. A group size re-balancing scheme implemented by simple logic is proposed to evenly distribute the c2v messages into groups to mitigate the precision loss and bridge the decoding performance gap. For an example decoder of MDPC codes with 80-bit security, the proposed optimized 2-stage scaled Min-sum algorithm leads to orders of magnitude error-correcting performance improvement and 16% higher clock frequency with negligible silicon area overhead compared to unoptimized Min-sum decoders.

This paper is organized as follows. Section II introduces the McEliece cryptosystem and baseline Min-sum decoding algorithm. Section III proposes optimized scaled Min-sum algorithm for MDPC decoding. The parallel Min-sum MDPC decoder architecture with flexible message storage scheme is presented in Section IV. The 2-stage scaling for critical path reduction and the group size re-balancing scheme are detailed in Section V and VI, respectively. Section VII presents hardware complexity comparisons. Discussions and conclusions follow in Section VIII and IX, respectively.

II. McEliece Cryptosystem and Min-Sum Decoding

An MDPC code is a linear block code that can be defined by a parity-check matrix $\mathbf{H}$. A vector $\mathbf{x}$ is a codeword iff $\mathbf{x}\mathbf{H}^T$=0. For the McEliece cryptosystem based on MDPC codes, n_0 r-bit vectors, each of which has Hamming weight w, are randomly generated and are used as the private key. They define the first columns of n_0 circulant matrices $\mathbf{H}_i$ $(0 \leq i < n_0)$

TABLE I: QC-MDPC code parameters considered in McEliece cryptosystem [2].

security level (bits)	n_0	n	r	w	t
80	2	9602	4801	45	84
	3	10779	3593	51	53
	4	12316	3079	55	42
128	2	19714	9857	71	134
	3	22299	7433	81	85
	4	27212	6803	85	68
256	2	65542	32771	137	264
	3	67593	22531	155	167
	4	81932	20483	161	137

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

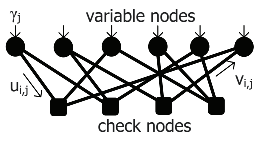


Fig. 1: A toy H matrix and its Tanner graph.

and the parity-check matrix of the MDPC code is $\mathbf{H} = [\mathbf{H}_0|\mathbf{H}_1|\cdots|\mathbf{H}_{n_0-1}]$. If $\mathbf{H}_{n_0-1}$ is not invertible, the corresponding vector needs to be regenerated randomly. The parity check matrix, $\mathbf{H}$, and the generator matrix, $\mathbf{G}$, of a linear block code satisfy $\mathbf{G}\mathbf{H}^T = 0$. Hence, $\mathbf{G}$ can be derived as $[\mathbf{I}|\mathbf{B}^T]$, where $\mathbf{B} = [\mathbf{H}_{n_0-1}^{-1}\mathbf{H}_0|\mathbf{H}_{n_0-1}^{-1}\mathbf{H}_1|\cdots|\mathbf{H}_{n_0-1}^{-1}\mathbf{H}_{n_0-2}]$. Each $\mathbf{H}_{n_0-1}^{-1}\mathbf{H}_i$ $(0 \leq i < n_0 - 1)$ is also circulant, and the first columns of these matrices form the public key of the McEliece cryptosystem.

The encryption mainly consists of an MDPC encoding process. The plaintext vector of $(n_0 - 1)r$ bits is multiplied with $\mathbf{G}$ to calculate a $n = n_0 r$-bit codeword $\mathbf{c}$. Then a randomly generated n-bit vector with at most t nonzero bits is added to $\mathbf{c}$ to derive the ciphertext $\mathbf{x}$. Hence, the ciphertext is an MDPC codeword corrupted by a random vector of t errors. The decryption is to carry out MDPC decoding on $\mathbf{x}$ to recover the correct codeword $\mathbf{c}$. Since $\mathbf{G}$ is systematic, the first $(n_0 - 1)r$ bits of $\mathbf{c}$ equals the plaintext. Table I lists the parameters of the MDPC codes adopted in the McEliece cryptosystem for the standard.

The $\mathbf{H}$ matrix of an MDPC code can be represented by a bipartite graph, also called the Tanner graph. This graph has two types of nodes: variable nodes and check nodes. Fig. 1 shows a toy $\mathbf{H}$ matrix and the corresponding Tanner graph. Each variable node represents a column of $\mathbf{H}$ and each check node corresponds to a row. If an entry in $\mathbf{H}$ is nonzero, the corresponding variable and check nodes are connected by an edge in the Tanner graph.

MDPC codes can be decoded by extending the algorithms for LDPC decoding. There are a spectrum of LDPC decoding algorithms, ranging from BF, Min-sum, to belief propagation. Compared to the simple BF algorithms, the Min-sum algorithm [14] achieves significant coding gain. Besides, they have much better performance-complexity tradeoff than the complicated belief propagation.

Most previous MDPC decoder designs for the McEliece cryptosystem use the simple BF algorithms. The popular Gal-

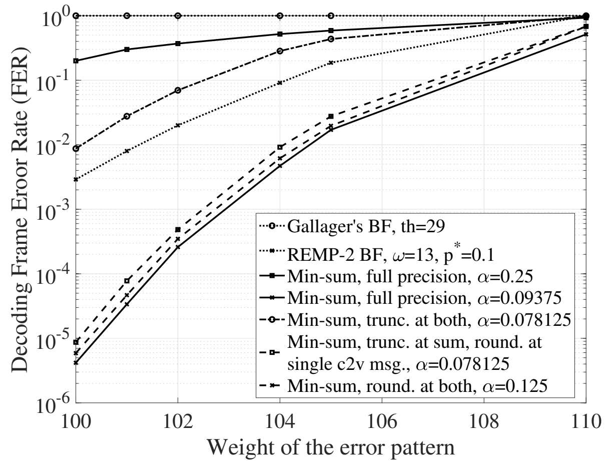


Fig. 2: FERs of Min-sum and BF decoding with $I_{max} = 30$ for an MDPC code with $(n_0, r, w) = (2, 4801, 45)$.

lager' BF algorithm flips a bit when its number of participating unsatisfied parity checks (PUPCs) exceeds a threshold [15]. Other variations on BF algorithms have been proposed in the literature to improve the correction performance for MDPC codes. In particular, the REMP-2 algorithm [4], which utilizes extrinsic messages and introduces random erasures with certain probability to the variable-to-check (v2c) messages, are among the BF algorithms with the best performance. Fig. 2 shows simulation results of Gallager's and REMP-2 BF algorithms for a randomly generated MDPC code with $(n_0, r, w) = (2, 4801, 45)$. The maximum number of decoding iterations is set to $I_{max} = 30$. For the Gallager's algorithm, $th = 29$ is used as the threshold for bit flipping because it leads to the lowest decoding frame error rate (FER) from simulations. For the REMP-2 algorithm, $\omega = 13$ is the weight multiplied to the channel bit and $p^* = 0.1$ is the probability of inserting erasures in the v2c message computation. These are optimized parameters found in [4]. Simulation results for Min-sum algorithms with optimized scalars, which will be detailed in the following sections, are also included in Fig. 2. They can achieve orders of magnitude improvement on the decoding FER compared to BF algorithms.

The Min-sum algorithm iteratively passes multi-bit reliability information between connected check and variable nodes to update the decisions on the input bits as listed in Algorithm 1. For hard-decision input vector $\mathbf{x} = [x_0, x_1, \cdots, x_{n-1}]$, the initial probability information of x_i, denoted by γ_i, is set to $+C$ or $-C$ when x_i is '0' or '1', respectively. Here C is a constant that can be decided from simulations. Its optimal value depends on the number of bits used to represent the reliability messages and the column weight of the code. In Algorithm 1, $u_{i,j}$ denotes the reliability message from variable node j to check node i, and $v_{i,j}$ is the message from check node i to variable node j. $S_c(j)$ ($S_v(i)$) represents the set of check (variable) nodes that are connected to variable (check) node j (i). The sign bit of a message is '0' and '1' when it is positive and negative, respectively. In Algorithm 1, $\oplus$ denotes the XOR operation. If no codeword is found at the end of iteration $I_{\max}$, decoding failure is declared. To improve the error-correcting capability, a scalar, α, is multiplied to the sum of the c2v messages in the variable node processing and *a posteriori* information calculation [14]. It is typically set to 0.5 or 0.25 for LDPC codes with column weight 4 in order to reduce the hardware complexity [16].

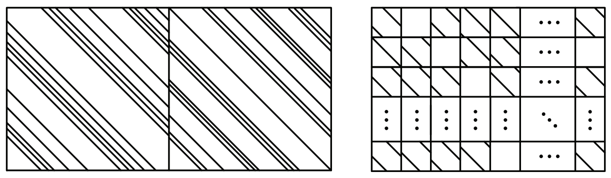


Fig. 3: Example parity check matrices of (a) an MDPC code used in the McEliece cryptosystem; (b) an LDPC code used for error correction in digital communication and storage systems.

Algorithm 1 Scaled Min-sum Decoding Algorithm

```
1: **Input:** $\mathbf{x} = [x_0, x_1, \cdots, x_{n-1}]$
2: **Initialization:** $u_{i,j} = \gamma_j$
3: **for** $iter = 1$ to $I_{\max}$ **do**
4:         Stop if $\mathbf{x}\mathbf{H}^T = 0$

5:         **Check node processing:**
6:         $min1_i = \min_{j \in S_v(i)} |u_{i,j}|$
7:         $idx_i = \arg\min_{j \in S_v(i)} |u_{i,j}|$
8:         $min2_i = \min_{j \in S_v(i), j \neq idx_i} |u_{i,j}|$
9:         $s_i = \oplus_{j \in S_v(i)} \text{sign}(u_{i,j})$
10:         for each  $j \in S_v(i)$
11:                 $|v_{i,j}| = \begin{cases} min1_i & \text{if } j \neq idx_i \\ min2_i & \text{if } j = idx_i \end{cases}$
12:                 $sign(v_{i,j}) = s_i \oplus sign(u_{i,j})$

13:         **Variable node processing:**
14:         $u_{i,j} = \gamma_j + \alpha \sum_{i' \in S_c(j), i' \neq i} v_{i',j}$

15:         **A posteriori info. comp. & tentative decision:**
16:         $\tilde{\gamma}_j = \gamma_j + \alpha \sum_{i \in S_c(j)} v_{i,j}$
17:         $x_j = \text{sign}(\tilde{\gamma}_j)$
18: **end for**
```

Many hardware implementation architectures have been developed for the Min-sum decoders of LDPC codes [17]–[20]. However, the LDPC codes used for error correction in communication and digital storage systems have fundamentally different structure in their parity check matrices compared to the MDPC codes in the McEliece cryptosystem as shown in Fig. 3. The diagonal lines in this figure denote the nonzero entries. A QC-LDPC parity check matrix consists of a large number of smaller CPMs. Hence, efficient parallel decoders can be realized by processing blocks of CPMs simultaneously in each clock cycle. However, such parallel processing is not applicable to MDPC decoders due to the irregular locations of the nonzero entries in the randomly generated $\mathbf{H}$ matrix. The check node processing in the BF decoding algorithms are simple XOR operations. Hence, previous MDPC decoders [8]–[11] process a segment of L consecutive bits in a column of $\mathbf{H}$ each time by using L check node units (CNUs) even though many of them are idling for those '0' entries in each clock cycle. On the other hand, the Min-sum check node processing has much higher complexity as shown in Algorithm 1. Extending the parallel processing scheme of BF decoders to Min-sum decoders would lead to large silicon area and low hardware efficiency. As a result, the parallel processing scheme needs to be re-designed for Min-sum MDPC decoding.

The MDPC codes listed in Table I have column weight 45 or higher instead of 3 or 4 as in the LDPC codes for communications and digital storage. The much higher column weight necessitates the re-optimization of the scaling in the Min-sum algorithm. Besides, the accumulation of a large number of c2v messages in the variable node processing and *a posteriori* information calculation of Algorithm 1 lead to long data paths in feedback loops and limit the achievable clock frequency of the decoder. As a result, the scaling scheme also needs to be reformulated for critical path reduction.

In the following sections, the scaled Min-sum algorithm is first optimized for MDPC codes. Then an efficient parallel decoder architecture is developed. Besides, 2-stage scaling and group size re-balancing schemes are proposed to reduce the critical path with negligible performance degradation and silicon area overhead.

III. Scaled Min-Sum Decoding Algorithm for MDPC Codes

In the Min-sum MDPC decoding for the McEliece cryptosystem, the initial probability information γ_j is set to either $+C$ or $-C$ when the i-th bit at the decoder input is '0' or '1', respectively. The value of C leading to the best decoding performance can be decided from simulations. In the variable node processing and *a posteriori* information calculation steps of Algorithm 1, the sum of the c2v messages are scaled by α before it is added to γ_j. If the magnitudes of the scaled sums are too big, then the initial probabilities and hence the bits at the decoder input do not have much effect on the decoding after the first iteration. On the other hand, if the initial probabilities have much larger magnitudes than the scaled sums, they will prevent erroneous bits from being corrected. Hence, the choice of the scalar α affects the decoding FER significantly. For LDPC codes with column weight 3 or 4, which are typically used in digital communication and storage systems, setting α to 0.5 or 0.25 leads to good error-correcting performance without actually requiring any multiplier in hardware implementation. However, when the column weight is much higher, as in the MDPC codes for the McEliece cryptosystem, these scalar values lead to substantial performance degradation. Fig. 2 shows simulation results for a randomly generated MDPC code with $(n_0, r, w) = (2, 4801, 45)$ and $I_{max} = 30$. In our simulations, each v2c and c2v message is represented by 4-bit integer magnitude and 1-bit sign. In this case, setting C to 9 leads to good performance. Using more bits to represent the messages would lead to lower FER at the cost of higher implementation complexity. From Fig. 2, it can be observed that, the Min-sum decoding with $\alpha = 0.25$ does not achieve better performance than the BF algorithms. A much smaller α is needed to prevent the sums of c2v messages from overly dominating the decoding process.

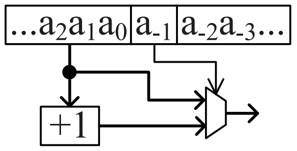


Fig. 4: Hardware implementation architecture for rounding.

The scalar that leads to the lowest FER can be found from simulations. To simplify the scalar multipliers, the scalars considered in our design can be represented by at most two nonzero digits, each of which is either $+1$ or -1. In this case, the scalar multiplication requires at most a single adder/subtractor to implement. The less significant bits of the scalar do not affect the decoding FER much. To reduce the search space, the scalar is limited to 6 digits in the fractional part. From simulations, among the scalars satisfying these conditions, $\alpha = 0.09375$ leads to the lowest FER when full precision is kept on the multiplication results. It is more than three orders of magnitude lower compared to using $\alpha = 0.25$ when there are 101 errors in the decoder input as shown in Fig. 2. When the input has 84 errors as the target correction capability listed in Table I, the gain would be even more significant.

To reduce the hardware complexity of Min-sum decoders, the *a posteriori* information of variable node j is shared to compute each v2c message from variable node j for the next decoding iteration as

$$u_{i,j} = (\gamma_j + \alpha \sum_{i' \in S_c(j)} v_{i',j}) - \alpha v_{i'j}. \quad (1)$$

When $|v_{i,j}|$ is represented as a 4-bit integer, $\lceil \log_2(w(2^4-1)) \rceil$ bits are needed to represent the magnitude of the sum of w c2v messages. If $|u_{i,j}|$ is larger than $2^4 - 1 = 15$, it is saturated to 15.

Scalar multiplication results have fractional parts. Truncation or rounding can be applied to bring them back to integer format. Assume that the scalar multiplication result is $a = \cdots a_2 a_1 a_0 . a_{-1} a_{-2} \cdots$. Truncation is just to discard the fractional bits. Rounding can be implemented by the architecture shown in Fig. 4. To shorten the data path, '1' is pre-added to the integer part of a. Then the result of this addition or the integer part itself is selected to be the rounding result when a_{-1} is '1' or '0', respectively. As the bits of a are generated from the scalar multiplication, they are added with '1'. Hence, such rounding does not add much to the data path of scalar multiplication.

In (1), different combinations of truncation and rounding can be applied to the two scalar multiplications. Simulations have been re-run to find the optimal scalar for each combination and the results are shown in Fig. 2. Carrying out rounding on both scalar multiplication results leads to the best performance. On the other hand, truncating $\alpha v_{i,j}$ substantially degrades the error-correcting performance. This is because that the optimal α for MDPC codes with high column weight is very small and hence α multiplied to a single c2v message

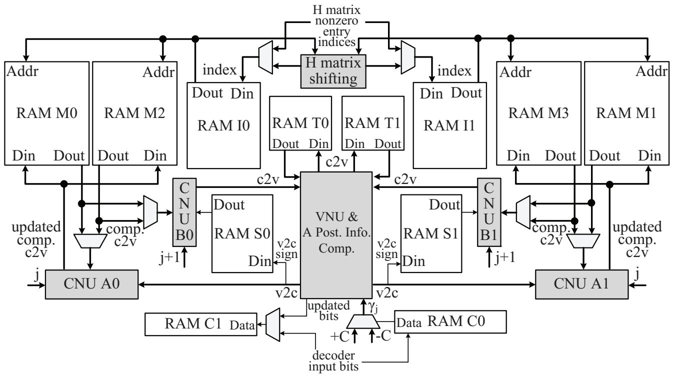
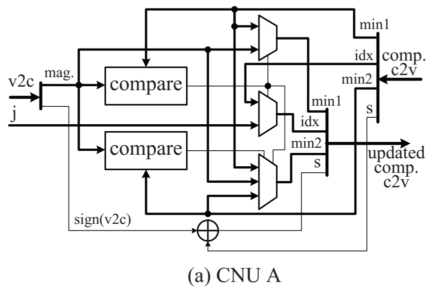


Fig. 5: Top-level architecture of the proposed Min-sum MDPC decoder for the case of 2-parallel processing.



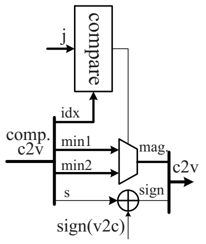


Fig. 6: (a) Architecture of CNU A; (b) architecture of CNU B.

$v_{i,j}$ is also small. As a result, discarding the fractional bits will lead to more substantial precision loss. As shown in Fig. 4, although rounding needs an extra multiplexer and an adder, these units have small area overhead and do not add much to the data path of the multiplier. Hence, rounding is applied after both scalar multiplications in (1) in our design.

IV. Efficient and Low-Latency Parallel Min-Sum MDPC Decoder Architecture

In this section, an efficient parallel Min-sum decoder architecture is first proposed for MDPC codes. Then a flexible message storage scheme is developed to reduce the decoding latency for randomly constructed MDPC codes.

A. *Efficient parallel Min-sum MDPC decoder architecture*

For MDPC codes, a column in a submatrix of $\mathbf{H}$ is a cyclical shift of the previous column and the locations of the '1's in each column are random. Due to these reasons, MDPC decoders can be implemented more efficiently by processing the $\mathbf{H}$ matrix column by column. Previous implementations of MDPC decoders consider BF algorithms [8]–[11], which are much simpler than the Min-sum algorithm listed in Algorithm 1. For BF algorithms, the v2c messages are single bits. The check node processing consists of XOR operations, and the variable node processing counts the total number of PUPCs, which is utilized to decide whether to flip a bit. For parallel processing, L CNUs can be allocated to process a segment of L consecutive bits of a column of $\mathbf{H}$ in each clock cycle. Even though many of the CNUs are idling since most of the entries of $\mathbf{H}$ are still '0' for MDPC codes, having many copies of them does not bring much area overhead to the overall decoder due to their very simple architecture. However, such a parallel processing scheme can not be extended to Min-sum decoders, which have much more complicated CNUs.

In our proposed Min-sum MDPC decoder, only the nonzero entries of $\mathbf{H}$ are processed in order to reduce the area and increase the hardware utilization efficiency. To reduce the memory requirement, the columns are processed one after another, and the nonzero entry indices for the next column are derived through adding those for the current column by one mod r. The block diagram of the proposed decoder for an example case of $L = 2$-parallel processing is shown in Fig. 5. In the beginning, the indices of the nonzero entries in the first column of each submatrix of $\mathbf{H}$ are split and stored into $L = 2$ blocks of RAM I. During the decoding process, as an index is read out, it is added by one mod r to derive the index of the shifted entry in the next column of H in the "H matrix shifting" block and the result is written back to RAM I. The decoder input bits are written into a pair of RAM C in the beginning. RAM C0 holds the input bits. Depending on whether a bit is '0' or '1', the corresponding probability information is set to $+C$ or $-C$, respectively, through a multiplexer. RAM C1 records the updated bit from Line 17 of Algorithm 1. When the decoding stops. The decoding result is available in RAM C1.

There are two pairs of RAM M. In the first decoding iteration, each row of the RAM M0 and M1 pair stores the temporary $min1$, $min2$, idx and s values corresponding to a row of $\mathbf{H}$. The indices stored in RAM I0 and I1 are used as the addresses to access RAM M0 and M1, and $L = 2$ CNUs are utilized to process the entries read out. A CNU consists of two parts and its architecture is shown in Fig. 6 [16]. Part A updates the temporary $min1$, $min2$, idx, and s values according to Lines 6-9 of Algorithm 1, and the updated results are stored back to RAM M0 and M1. After all the columns of $\mathbf{H}$ are processed, the final $min1$, $min2$, idx, and s for every row of $\mathbf{H}$ are available in RAM M0 and M1. They can be considered as compressed c2v messages. From these values, part B of the CNU derives the c2v messages according to Lines 11-12 of Algorithm 1. The signs of $u_{i,j}$ that are processed by the same CNU A are stored into consecutive addresses of a RAM S block. Each CNU A processes one $u_{i,j}$ in each clock cycle. However, storing the signs of $u_{i,j}$ into memories of 1-bit wide leads to very deep memories considering the large column weight of the code and long codeword length. Instead, our design uses a shift register to collect a number of sign bits, such as 2^e ($e \in Z^+$), and writes them into memories that are 2^e-bit wide. The depth of the memories is reduced accordingly. e can be adjusted to trade off memory width and depth. When the sign bits are needed for Line 12 of Algorithm 1 in CNU B, 2^e of them are read out from a line of RAM S each time and loaded into a shift register, which shifts out individual sign bits.

A 2-parallel variable node unit (VNU) can be implemented by the architecture in Fig. 7. The c2v and v2c message

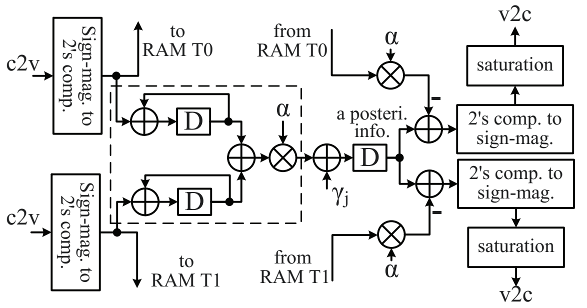


Fig. 7: Architecture of the 2-parallel VNU.

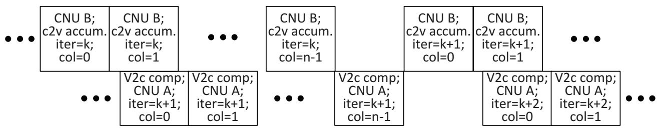

CNU B; c2v accum. iter=k; col=0
CNU B; c2v accum. iter=k; col=1
CNU B; c2v accum. iter=k; col=n-1
CNU B; c2v accum. iter=k+1; col=0
CNU B; c2v accum. iter=k+1; col=1
V2c comp; CNU A; iter=k+1; col=0
V2c comp; CNU A; iter=k+1; col=1
V2c comp; CNU A; iter=k+1; col=n-1
V2c comp; CNU A; iter=k+2; col=0
V2c comp; CNU A; iter=k+2; col=1

Fig. 8: Scheduling of variable and check node processing in the proposed decoder.

magnitudes are represented by d bits. The c2v messages are first converted from sign-magnitude format to 2's complement representation. Then up to $L = 2$ c2v messages are accumulated in each clock cycle. Since it takes multiple clock cycles to calculate the *a posteriori* message, the c2v messages are also stored into RAM T before they are subtracted to derive the v2c messages for the next iteration. The v2c messages are converted back to sign-magnitude representation, and those magnitudes larger than $2^d - 1$ are saturated to this value before they are sent to the CNUs. While the c2v messages from decoding iteration k are being generated, the newly computed v2c messages are sent to the CNUs to carry out decoding iteration $k + 1$. Therefore, the RAM M2, M3 pair and RAM M0, M1 pair are used in a ping-pang manner to store the final $min1$, $min2$, idx, and s for iteration k and those temporary values for iteration $k + 1$. Different from the compressed c2v messages stored in RAM M, a sign bit for decoding iteration k can be overwritten by that for iteration $k+1$ once it is utilized to derive the corresponding c2v message in CNU B. Hence, the decoder only needs one set of RAM S blocks.

Fig. 8 shows the scheduling of the computations in our proposed decoder. The c2v messages generated by CNU Bs for column j are accumulated, scaled, and added to γ_j. Then the sum is subtracted by the scaled individual c2v messages stored in RAM T to compute the v2c messages for column j, which are sent to CNU A to carry out the check node processing for the next iteration. Meanwhile, CNU B generates the c2v messages for column $j + 1$. Although the nonzero indices for two columns of $\mathbf{H}$ are needed at the same time, the indices of column $j+1$ can be derived from those of column j by adding one mod r on the fly. Hence, only the nonzero indices of one column of each sub-matrix of $\mathbf{H}$ are stored in RAM I at any time. From Table I, $n_0 > 1$ for the MDPC codes considered for the standard. Hence, each RAM I block consists of n_0 banks.

The sizes of the RAM blocks utilized in a L-parallel decoder for MDPC codes with code parameters (n_0, r, w) and d-bit

TABLE II: Sizes of the RAM blocks utilized in the proposed L-parallel decoder for (n_0, r, w) MDPC codes with d-bit c2v and v2c message magnitude.

RAM name	# of blocks	size	values stored
RAM I	L	$(\lceil w/L \rceil + \delta) \times (n_0 \lceil \log_2(r/L) \rceil)$	row indices of nonzero entries in one column of the H matrix
RAM C	2	$n \times 1$	bits indicating whether γ_j equals $+C$ or $-C$ / updated hard-decision bits
RAM M	$2L$	$\lceil r/L \rceil \times (2d + 1 + \lceil log_2 n \rceil)$	compressed version of all $v_{i,j}$
RAM S	L	$(n\lceil (w/L + \delta)/2^e \rceil) \times 2^e$	sign bits for all $u_{i,j}$
RAM T	L	$(\lceil w/L \rceil + \delta) \times (d+1)$	buffer for $u_{i,j}$ for column j during variable node processing

c2v and v2c message magnitude are summarized in Table II. If the decoder needs to be reconfigurable to support codes with different parameters, each RAM should be set to the largest possible size. Taking into account the random construction of $\mathbf{H}$, the numbers of nonzero entries in each segment of a column of $\mathbf{H}$ are uneven. Hence, $\lceil w/L \rceil + \delta$ rows are allocated to each of the RAM I and T blocks. The value of δ can be derived from simulations over a large number of random codes. The entries in each memory, except RAM M blocks, are accessed consecutively one after another. The addresses for accessing RAM M blocks are read from RAM I blocks.

B. Flexible message storage scheme for decoding latency reduction

To achieve L-parallel processing, the indices of the nonzero entries in each column of the $\mathbf{H}$ matrix need to be split and stored into L blocks of RAM I. The number of clock cycles needed to process a column of $\mathbf{H}$ is the maximum of the number of entries stored in the L memory blocks. The $\mathbf{H}$ matrix for the McEliece cryptosystem is generated randomly. In many cases, the nonzero entries are unevenly distributed. As a result, a larger number of clock cycles are needed for each decoding iteration. To address this issue, our proposed decoder allows two different nonzero entry splitting and message storage schemes. For a given randomly generated $\mathbf{H}$ matrix, the scheme leading to shorter decoding latency is chosen. Both of the schemes are implemented with negligible hardware overhead.

Our first scheme divides the rows of $\mathbf{H}$ into L segments, where segment l ($0 \leq l < L - 1$) consists of $\lceil r/L \rceil$ consecutive rows starting from row $\lceil r/L \rceil l$ and the last segment has the rest rows. The indices of the nonzero entries in segment l of a column are stored in block l of RAM I, and the $\lceil r/L \rceil$ compressed c2v messages of this segment are stored in one RAM M block. The indices are added by one mod r to generate the indices for the next column of H. If the modular sum falls into the range of the next segment, it is stored in the next block of RAM I. Our second scheme puts row i satisfying $i \mod L = l$ in segment l. The nonzero entry indices of $\mathbf{H}$

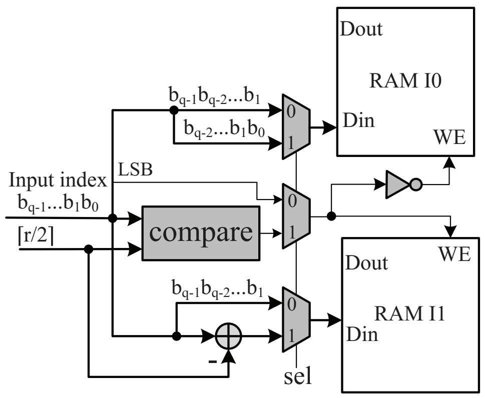


Fig. 9: Modifications to support flexible message storage scheme in a 2-parallel decoder.

in the same segment are stored in the same RAM I block. The compressed c2v messages corresponding to a segment are recorded in a block of RAM M. Different from the first scheme, an index added by one mod r always belongs to the next segment and is written to the next block of RAM I in the derivation of the next column of $\mathbf{H}$.

For a given randomly generated $\mathbf{H}$ matrix, the number of clock cycles needed to process a column equals to the maximum of the number of nonzero entries in the corresponding L segments of bits. These numbers for each of the columns can be derived from simulations and are added up to estimate the latency of each decoding iteration. The storage scheme leading to the shorter latency is chosen. To support the proposed scheme, the 2-parallel decoder architecture in Fig. 5 is modified as shown in Fig. 9 to be able to select one of the two storage schemes on the fly. For the purpose of conciseness, only the modified parts are shown in this figure. The select signal sel equals '1' and '0' when the first and second storage scheme, respectively, are chosen. The index of each nonzero entry originally has $q = \lceil \log_2 r \rceil$ bits. For the first scheme, if the index is lower than $\lceil r/2 \rceil$, it is stored into RAM I0. Otherwise, it is subtracted by $\lceil r/2 \rceil$ and the difference is stored into RAM I1. This subtraction is necessary since the indices stored in RAM I are used as the addresses to access RAM M and the addresses of each RAM M start from zero. For the second scheme, whether the index is even or odd can be told from the least significant bit (LSB). The LSB is eliminated and the higher $q-1$ bits are stored into RAM I0 and I1 also because the addresses for accessing each RAM M block start from zero. Compared to the decoder architecture in Fig. 5, the extra units needed to support the flexible message storage scheme include the subtractor, comparator, and multiplexers shown in gray color in Fig. 9. Besides, the logic in the "H matrix shifting" block of Fig. 5 needs to be modified accordingly. These modifications bring negligible overheads to the overall decoder and can be easily extended to decoders with higher parallelism.

To find the decoding latency reduction that can be achieved by the proposed scheme, simulations have been carried out on 1000 randomly generated MDPC codes with $(n_0, r, w) = (2, 4081, 45)$. The shorter decoding latency of the two message storage schemes is compared to that of the first scheme and

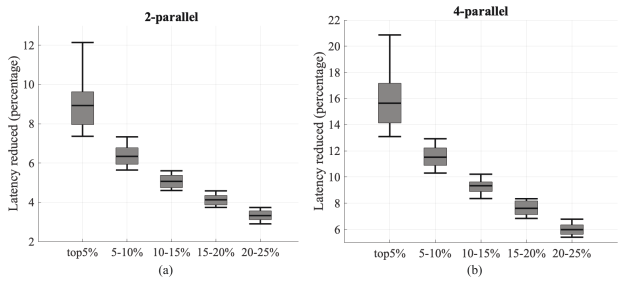


Fig. 10: Decoding latency reduction achieved by the proposed flexible message storage scheme over 1000 randomly generated MDPC codes with $(n_0, r, w) = (2, 4801, 45)$ for (a) 2-parallel decoder; (b) 4-parallel decoder.

the results are shown in Fig. 10. Among the 1000 codes, for 4-parallel design, the codes with top 5% decoding latency reduction have their latency reduced by an average of 15.6% using the proposed scheme. The latency reductions range between 13.1% and 20.9% and the top and bottom edges of the rectangle in Fig. 10 represent the bottom and top quadruples, respectively, of the latency reduction. For the next 5% of the codes with the greatest latency reduction, which is labeled as '5-10%' in the figure, the average latency reduction achieved by the second scheme on average is 11.2%. The achievable saving increases with the parallelism since dividing the rows of $\mathbf{H}$ into more segments leads to larger variation on the number of nonzero entries in each segment.

V. Two-Stage Scaling for Critical Path Reduction

In the VNU architecture shown in Fig. 7, even though the c2v messages are accumulated in L streams, a large number of c2v messages are accumulated in each feedback loop due to the high column weight of the MDPC codes. Hence, the adders in the feedback loops are wide. For example, for the MDPC code with $w = 45$, the δ in Table II for 2-parallel processing is 10 from simulations. Hence $\lceil w/L \rceil + \delta = 33$ c2v messages may be accumulated in a feedback loop. If each c2v message is represented by 1-bit sign and $d = 4$-bit magnitude, the result needs $\lceil \log_2 33 \rceil + 5 = 11$ bits to represent. Since each full adder has 2 levels of logic in the data path, the accumulator has 22 levels of logic in the data path. Comparatively, the 4-bit comparator and 3-input multiplexer in the feedback loop of a CNU shown in Fig. 6(a) can be implemented by 8 levels of 2-input logic. Therefore, the feedback loops in the VNUs have much longer data path than those in the CNUs and they limit the achievable clock frequency of the overall decoder. To shorten the critical path, a 2-stage scaling scheme is proposed in this section. Additionally, a systematic procedure is developed to decide the parameters of the 2-stage scaling for achieving a given critical path goal.

In our 2-stage scaling scheme, the scalar α is decomposed into α_1 and α_2. The stream of c2v messages output from a CNU is divided into groups of g messages. In the case that the number of messages is not a multiple of g, the last group has a smaller number of messages. The messages in each group are accumulated and the result is multiplied by α_1. The

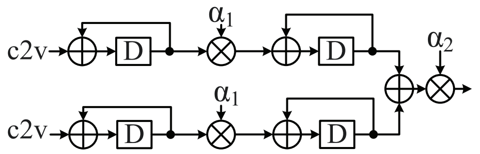


Fig. 11: Architecture implementing 2-stage scaling for 2-parallel decoders.

product is rounded to eliminate the fractional bits. Then the rounded product for each group is accumulated in the second feedback loop. The final accumulated result is scaled by α_2 followed by rounding. For 2-parallel decoding, the architecture implementing this 2-stage scaling is shown in Fig. 11. This architecture replaces the units circled by the dashed block in the VNU architecture in Fig. 7.

Since the sum of each group is scaled down and rounded before it is further accumulated, the widths of the adders in the feedback loops and accordingly the critical path is reduced. Through tuning the group size g, α_1, and α_2, the data paths in the feedback loops can be controlled to achieve a given critical path goal. On the other hand, the rounding after the α_1 multiplication causes additional precision loss and hence performance degradation compared to using a 1-stage scaling with $\alpha = \alpha_1\alpha_2$. To reduce the hardware implementation complexity, every involved scalar should be represented by using no more than 2 nonzero digits. All these issues are jointly considered to develop the following steps for deciding g, α_1, and α_2, given that the critical path does not exceed p full adders.

Step 1: Group Size Decision

The group size, g, in the 2-stage scaling should be as large as possible since adding more c2v messages up before scaling and rounding reduces precision loss and improves the error-correcting performance. To make the width of the adders in the first-stage feedback loops of Fig. 11 at most p-bit, the group size can be chosen as $g = \lfloor (2^{p-1}-1)/(2^4-1) \rfloor$ when each c2v message uses 4 bits to represent the magnitude. For example, if the critical path goal is $p = 9$ full adders, the group size is set to $g = \lfloor (2^8-1)/(2^4-1) \rfloor = 17$.

Step 2: Range of α_1 Determination

The number of bits needed to represent the data in the second-stage feedback loops in Fig. 11 should not exceed p either to achieve a critical path of p full adders. The maximum number of c2v messages sent to a first-stage feedback loop in Fig. 11 is $\lceil w/L \rceil + \delta$. Then the maximum magnitude at the output of a second-stage feedback loop is $\alpha_1(2^4-1)(\lceil w/L \rceil + \delta)$. Therefore, from $\alpha_1(2^4-1)(\lceil w/L \rceil + \delta) \leq 2^{p-1}-1$, the range of α_1 can be determined. For example, in 2-parallel decoding, from the simulation over 1000 randomly generated MDPC codes with $(n_0, r, w) = (2, 4081, 45)$, it was found that $\delta = 10$. To achieve a critical path of $p = 9$ full adders, $\alpha_1 \leq (2^{9-1}-1)/((2^4-1)(\lceil 45/2 \rceil + 10)) = 0.515$ in the 2-parallel decoder.

Step 3: α_1, α_2, and α Selection

To reduce the hardware implementation complexity, our design allows each scalar, including α_1, α_2, and $\alpha = \alpha_1\alpha_2$, to have 6 digits in the fractional part and at most two digits are nonzero. In this case, the corresponding multiplication takes at most one addition/subtraction. Once the range of α_1 is determined from Step 2, all possible values of α_1, α_2, and α satisfying these conditions can be found. For $\alpha_1 \leq 0.515$ considered in the previous example, Table III shows all possible combinations of α_1, α_2 and α satisfying the constraints. Simulations are then carried out to determine the combination of scalars leading to the lowest FER.

Following the above procedure, the scalars leading to

TABLE III: Combinations of $\alpha_1 \leq 0.515$, $\alpha_2 < 1$ and $\alpha = \alpha_1\alpha_2$, each of which has up to 2 nonzero digits in 6-digit representation.

α_1	α_2	α
0.5	0.03125	0.015625
	0.0625	0.03125
	0.09375	0.046875
	0.125	0.0625
	0.15625	0.078125
	0.1875	0.09375
	0.21875	0.109375
	0.25	0.125
	0.28125	0.140625
	0.3125	0.15625
	0.375	0.1875
	0.4375	0.21875
	0.46875	0.234375
	0.5	0.25
	0.53125	0.265625
	0.5625	0.28125
	0.625	0.3125
	0.75	0.375
	0.875	0.4375
	0.9375	0.46875
	0.96875	0.484375
0.46875	0.5	0.234375
0.4375	0.5	0.21875
	0.25	0.109375
0.375	0.75	0.28125
	0.625	0.234375
	0.5	0.1875
	0.375	0.140625
	0.25	0.09375
	0.125	0.046875

α_1	α_0	α
0.3125	0.75	0.234375
	0.5	0.15625
	0.25	0.078125
0.28125	0.5	0.140625
0.25	0.0625	0.015625
	0.125	0.03125
	0.1875	0.046875
	0.25	0.0625
	0.3125	0.078125
	0.375	0.09375
	0.4375	0.109375
	0.5	0.125
	0.5625	0.140625
	0.625	0.15625
	0.75	0.1875
	0.875	0.21875
	0.9375	0.234375
0.21875	0.5	0.109375
0.1875	0.25	0.046875
	0.5	0.09375
	0.75	0.140625
0.15625	0.5	0.078125
0.125	0.125	0.015625
	0.25	0.03125
	0.375	0.046875
0.125	0.5	0.0625
	0.625	0.078125
	0.75	0.09375
	0.875	0.109375
0.09375	0.5	0.046875
0.0625	0.25	0.015625
	0.5	0.03125
	0.75	0.046875
0.03125	0.5	0.015625

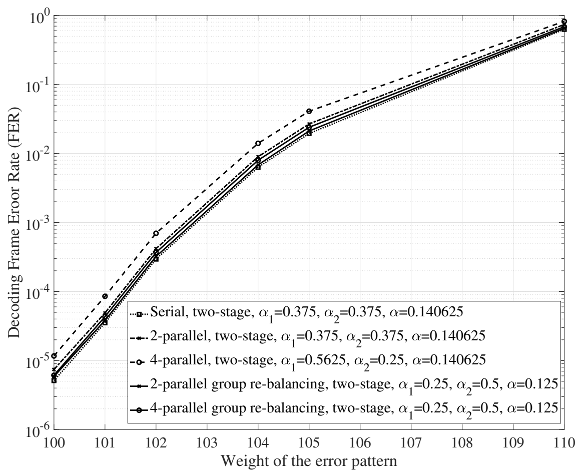


Fig. 12: FERs of two-stage scaling Min-sum decoders achieving 9-full-adder critical path for a random MDPC code with $(n_0, r, w) = (2, 4801, 45)$ and $I_{max} = 30$.

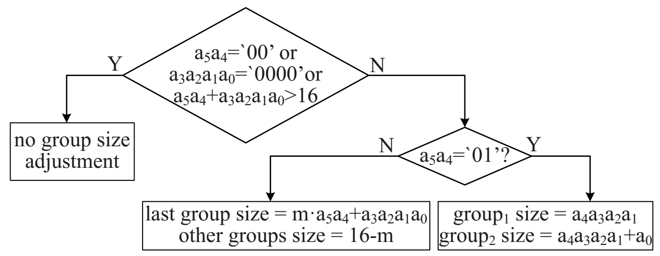


Fig. 13: Flow chart for group size re-balancing when $g = 16$ and the number of c2v messages in a stream to accumulate is at most $3g$.

the lowest FER for the 2-parallel decoder consisting of 9 full adders in the critical path are $\alpha_1 = \alpha_2 = 0.375$, and $\alpha = 0.140625$. This 2-stage scaling reduces the critical path and increases the achievable clock frequency by $(1/9 - 1/11)/(1/11) = 22\%$ compared to the original 1-stage scaling that has 11 full adders in the critical path. For a 4-parallel decoder with the same critical path, the best scalars found from simulations are $\alpha_1 = 0.5625, \alpha_2 = 0.25$, and $\alpha = 0.140625$. The FERs of these two settings are shown in Fig. 12 for a randomly generated MDPC code. The performance degradation of the decoding with 2-stage scaling compared to 1-stage scaling is mainly caused by the precision loss resulted from the α_1 multiplication and rounding. In particular, if the last group of c2v messages has less than g entries, the scaling and rounding on their sum cause more precision loss. This analysis is further verified from the fact that the 4-parallel decoder has higher FER than the 2-parallel decoder as shown in Fig. 12, although both of the settings use the best possible scalars. In the 4-parallel design, since the c2v messages associated with a column of $\mathbf{H}$ are divided into 4 instead of 2 streams, the last group of each stream more likely has a smaller number of entries. Besides, there are 4 such groups that go through scaling and rounding. As a result, more precision is lost.

VI. Group Size Re-balancing

To mitigate the performance loss brought by scaling and rounding the sum of a small number of c2v messages in the 2-stage scaling, a group size re-balancing scheme is proposed in this section. This technique aims to evenly distribute the number of c2v messages to be accumulated in each group as much as possible, thereby reducing the amount of precision loss and narrowing the gap in the decoding performance.

To simplify the group size re-balancing, g is adjusted to the next lower integer that is in the format of a power of 2. For the example MDPC code with $(n_0, r, w) = (2, 4801, 45)$ with 2-parallel decoding and 9 full adders in the critical path, g is adjusted from 17 to 16. For an example case that the number of c2v messages to accumulate in a stream does not exceed $3g = 48$, the group sizes can be re-balanced according to the chart in Fig. 13. A similar procedure can be developed to re-balance the group sizes for other values of g and/or when the number of c2v messages to accumulate in a stream is more than $3g$.

TABLE IV: Number of entries to move from each of the first $a_5a_4 = 2$ groups to the last group for size re-balancing in the case of $g = 16$.

$a_3a_2a_1a_0$	$m_2m_1m_0$	$a_3a_2a_1a_0$	$m_2m_1m_0$	$a_3a_2a_1a_0$	$m_2m_1m_0$
0001	101	0110	011	1011	010
0010	101	0111	011	1100	001
0011	100	1000	011	1101	001
0100	100	1001	010	1110	001
0101	100	1010	010	1111	000

In the case that the number of c2v messages in a stream does not exceed 48, it can be represented by a 6-bit integer as $a_5a_4a_3a_2a_1a_0$. Originally, for $g = 16$, $h = a_5a_4$ is the number of groups with $g = 16$ entries and $y = a_3a_2a_1a_0$ is the size of the last group if $y \neq 0$. When $y = 0$, the stream length is a multiple of 16 and every group has 16 entries. Group size adjustment is not necessary. When $h = 0$, there is only one group with less than 16 entries. Group size re-balancing is not needed either. Since 48='11000', a_5a_4 ='01' or '10' for the rest cases. If $y \neq 0$, our design tries to take out the same number of entries from each of the first h groups and put them into the last group. However, if $h + y > 16$, this would make the last group have more than 16 entries even if only one entry is moved from each of the other groups. Our design chooses not to adjust the group size in this case.

If group size re-balancing is needed, and there are only two groups, which happens if a_5a_4 ='01', the stream is evenly divided between the two groups. The size of one group is adjusted to $a_4a_3a_2a_1$, and the size of the other is $a_4a_3a_2a_1 + a_0$. For the remaining cases, a_5a_4 ='10'. Assume that m entries are moved from each of the first $h = a_5a_4$ groups to the last group. Then the size of the last group becomes $m \cdot a_5a_4 + a_3a_2a_1a_0$, while the first h groups are reduced to $16-m$ entries each. The goal of the group size re-balancing is to distribute the c2v messages among the groups as evenly as possible. Hence, m should be chosen to minimize the difference between $m \cdot a_5a_4 + a_3a_2a_1a_0$ and $16 - m$. For a_5a_4='10'=2, such values of $m = m_2m_1m_0$ corresponding to different $a_3a_2a_1a_0$ are listed in Table IV. Utilizing K-map, simplified logic formulas can be derived for $m_2m_1m_0$.

Simulation results of MDPC decoding using the the proposed group size re-balancing scheme are also included in Fig. 12. It can be observed that the group size re-balancing effectively reduces the performance gap between the decoding with single-stage scaling and that with two-stage scaling and parallel processing. Besides, it can be implemented by simple logic following the chart in Fig. 13. After the accumulation of the c2v messages of each group, the result is sent to the α_1 multiplier, while the feedback loop starts to accumulate the c2v messages for the next group. Therefore, having groups of adjusted sizes does not bring any latency penalty.

VII. Hardware Complexity Comparisons

The proposed flexible message storage scheme only requires a comparator, an adder, a few extra multiplexers, and simple change in the control logic. It brings negligible area overhead

TABLE V: Synthesis results of the VNU with 1-stage and 2-stage scaling using GlobalFoundries 22FDX process.

	timing constraint (ps)	area (μm^2)
VNU with 1-stage scaling	242	467
VNU with 2-stage scaling	208	604

TABLE VI: 2-parallel Min-sum and BF decoder area and latency comparisons for an MDPC code with $(n_0, r, w) = (2, 4801, 45)$.

	REMP-2 BF [4]	Min-sum (1-stg. scal.)	Min-sum (2-stg. scal. proposed)
RAM I (bits)	1716	1716	1716
RAM C (bits)	19204	19204	19204
RAM M (bits)	163268	220892	220892
RAM S (bits)	633732	633732	633732
RAM T (bits)	132	330	330
Total memory (bits)	818052	875874	875874
Logic Area (μm^2)	299	577	718
Total Area (# of NAND2) (normalized)	1228573 (1)	1316696 (1.0717)	1317401 (1.0723)
Avg. # of iter.	4.41	4.66	4.68
# of clks /iter. (worst case)	316866	316866	316866
Max. clock freq. (GHz)	6.024	4.132	4.808
Latency (ms) (normalized)	0.232 (1)	0.357 (1.539)	0.308 (1.328)

while achieving non-trivial latency reduction for a significant portion of MDPC codes as shown in Fig. 10. The group size re-balancing scheme can be also implemented by simple logic with negligible area overhead, such as according to Fig. 13 for an example setting. It mitigates the error-correcting performance loss brought by the 2-stage scaling without increasing the number of clock cycles needed for the decoding.

The 2-stage scaling scheme only affects the VNUs. To further evaluate the data path reduction achievable by this scheme, the VNU architecture in Fig. 7 for implementing the original 1-stage scaling and that with the units inside the dashed block replaced by the architecture in Fig. 11 for implementing the proposed 2-stage scaling are synthesized using GlobalFoundries 22FDX process. The synthesis results are listed in Table V. Each design is synthesized using different timing constraints. For the VNU with 1-stage scaling, the area increases substantially when the timing constraint becomes shorter than $242ps$. On the other hand, the area of the VNU with 2-stage scaling does not increase significantly until the timing constraint becomes shorter than $208ps$. Hence the 2-stage scaling increases the achievable clock frequency by (1/208-1/242)/(1/242)=16%. This percentage is less than the (1/9-1/11)/(1/11)=22.2% estimated from the critical path in architectural level since the setup time and propagation delay of the registers also contribute to the achievable clock period.

The overall complexities of 2-parallel Min-sum decoders are compared in Table VI. The logic areas listed in this table are contributed by VNUs, CNUs, and other control logic. Hence they are larger than those in Table V, which are for VNUs only. It can be assumed that storing a bit in memory takes the area of 1.5 NAND2 gates [9]. On the other hand, the area of one NAND2 gate in the GlobalFoundries 22FDX process is $0.2\mu m^2$. From this assumption, the size of the proposed Min-sum MDPC decoder in terms of equivalent NAND2 gates is listed in Table VI. It can be seen that our new design has negligible area overhead compared to the design with traditional 1-stage scaling. This is because that the memories contribute to the majority of the complexity and they are the same in both designs. The average number of decoding iterations is collected from simulations over 10,000 samples with $t = 84$ errors at decoder input, which is the error number targeted by the standard. From the average number of decoding iterations, number of clock cycles for each decoding iteration, and the maximum achievable clock frequency for each decoder listed in Table VI, it can be calculated that the proposed decoder with 2-stage scaling achieves (0.357-0.308)/(0.357)=14% latency reduction. This is lower than the 16% achievable clock frequency increase because that the proposed design requires slightly more decoding iterations.

For comparison, the complexity of the REMP-2 decoder, which is among the best-performing BF decoders for MDPC codes [4], is also included in Table VI. It can be implemented following similar overall architecture and scheduling scheme as shown in Fig. 5 and Fig. 8, respectively. Since each v2c message in the REMP-2 decoder can only have three possible values: ‘0’, ‘+1’, and ‘-1’, the CNU is simplified to count whether the number of ‘0’ v2c messages is 0, 1, or at least 2. Accordingly, the size of RAM M is reduced. However, the sign bit of every v2c message still needs to be stored in RAM S. As it can be observed from Table II, this RAM dominates the memory complexity due to the high column weight of the code. As a result, the total memory size of the REMP-2 decoder is not reduced much and the overall complexity is only 7% less compared to Min-sum decoders for the MDPC code with $(n_0, r, w) = (2, 4801, 45)$ as shown in Table VI. Since the c2v messages in the REMP-2 BF decoder are either ‘0’, ‘+1’, or ‘-1’ instead of 5-bit as in the proposed Min-sum decoder, their accumulation in the VNU has shorter data path. Hence, the achievable clock frequency of the REMP-2 decoder is higher. Although the proposed 2-stage scaled Min-sum decoder has (0.308-0.232)/0.232=32.8% longer latency, its decoding FER is several orders of magnitude lower compared to the REMP-2 decoder as shown in Fig. 2.

VIII. Discussions

For codes with higher column weight w, the critical path of the original scaled Min-sum decoder is even longer. The parameters of the 2-stage scaling can be tuned to achieve a given critical path goal. Accordingly, the proposed design can achieve even more significant improvement on the clock frequency.

In the proposed flexible message storage scheme, two different schemes are implemented and the decoder can choose to use one of them on the fly for a given MDPC code. More storage schemes can be utilized to further reduce the decoding latency for a larger percentage of random MDPC codes. However, the additional improvement would be less significant due to the random locations of the nonzero entries in the parity-check matrix.

Due to the irregularity of the parity-check matrix, increasing the parallelism of the proposed design would lead to less significant increase in the decoding throughput. However, the proposed design processes only the nonzero entries in the $\mathbf{H}$ matrix. Even with low parallelism, it is more effective that prior designs that process a large number of consecutive bits in a column of $\mathbf{H}$ in each clock cycle. This is because that, although the parity-check matrix of an MDPC code has higher density than that of an LDPC code, it is still very sparse. As a result, a segment of consecutive bits in a column of $\mathbf{H}$ most likely only has one nonzero bit. For example, it was found in [11] that, for the code with $(n_0, r, w) = (2, 4801, 45)$, dividing a column of $\mathbf{H}$ into 32-bit segments will lead to 39 segments with nonzero bits. Hence, 39 clock cycles are needed to process a column in this 32-parallel design. On the other hand, our proposed 2-parallel design requires at most $\lceil w/2 \rceil + \delta = 33$ clock cycles to process a column of $\mathbf{H}$.

IX. Conclusions

For the first time, this paper investigates scaled Min-sum decoding for MDPC codes with high-column weight used in the McEliece cryptosystem. The scalar is optimized to improve the error-correcting performance and a 2-stage scaling scheme is developed to reduce the critical path in the decoder hardware implementation. An efficient parallel Min-sum MDPC decoder is also proposed by jointly considering the non-trivial check node processing and the irregular but sparse parity check matrix. A group-size re-balancing scheme is developed to mitigate the performance degradation caused by 2-stage scaling in parallel decoders. Besides, a flexible message storage scheme is proposed to reduce the decoding latency for randomly constructed MDPC codes. Future work will address reducing the memory requirement of the decoder and achieving efficient design with higher parallelism.

References

[1] D. J. Bernstein, *et al.* “Classic McEliece: conservative code-based cryptography,” available at https://classic.mceliece.org/nist.html.

[2] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. L. M. Barreto, “MDPC-McEliece: new McEliece variants from moderate density parity-check codes,” *Proc. IEEE Intl. Symp. on Info. Theory*, pp. 2069-2073, Oct. 2013.

[3] S. Lin and D. J. Costello, *Error Control Coding*, Pearson, 2004.

[4] H. Bartz and G. Liva, “On decoding schemes for the MDPC-McEliece cryptosystem,” *Proc. of Intl. ITG Conference on Systems, Communications, and Coding*, pp. 1-6, Mar. 2019.

[5] T. B. Paiva and R. Terada, “Faster constant-time decoder for MDPC codes and applications to BIKE KEM,” *IACR Trans. on Cryptographic Hardware and Embedded Syst.*, vol. 2022, no. 4, pp. 110-134, Aug. 2022.

[6] P. Santini, M. Battaglioni, M. Baldi, and F. Chiaraluce, “Analysis of the error correction capability of LDPC and MDPC codes under parallel bit-flipping decoding and application to cryptography,” *IEEE Trans. on Commun.*, vol. 68, no. 8, pp. 4648-4660, Aug. 2020.

[7] S. Arpin, *et al.* “A study of error floor behavior in QC-MDPC codes,” *Proc. Intl. Conf. on Post-Quantum Cryptography*, pp. 89-103, Sep. 2022.

[8] I. V. Maurich and T. Güneysu, “Lightweight code-based cryptography: QC-MDPC McEliece encryption on reconfigurable devices,” *Proc. IEEE Design, Automation & Test in Europe Conference & Exhibition*, pp. 1-6, Mar. 2014.

[9] I. V. Maurich, T. Oder, and T. Güneysu, “Implementing QC-MDPC McEliece encryption,” *ACM Trans. on Embedded Computing Syst.*, vol. 14, no. 3, pp. 1-27, Apr. 2015.

[10] J. Hu and R. Cheung, “Area-time efficient computation of Niederreiter encryption on QC-MDPC codes for embedded hardware,” *IEEE Trans. on Computers*, vol. 66, no. 8, pp. 1313-1325, Aug. 2017.

[11] Z. Xie and X. Zhang, “Sparsity-aware medium-density parity-check decoder for McEliece cryptosystems," *IEEE Trans. on Circuits and Syst.-II*, minor revision, 2022.

[12] Q. Guo, T. Johansson, and P. Stankovski, “A key recovery attack on MDPC with CCA security using decoding errors," *Proc. ASIACRYPT*, pp. 789815, Dec. 2016.

[13] P. Santini, M. Battaglioni, F. Chiaraluce, and M. Baldi, “Analysis of reaction and timing attacks against cryptosystems based on sparse parity-check codes," *Proc. Code-Based Cryptography: 7th Intl. Workshop*. pp. 115-136, Jul. 2019.

[14] J. Chen and M. Fossorier, “Density evolution for two improved BP-based decoding algorithms of LDPC codes," *IEEE Commun. Letters*, vol. 6, no. 5, pp. 208-210, May 2002.

[15] R. Gallager, *Low-Density Parity-Check Codes*, MIT Press, 1963.

[16] X. Zhang, *VLSI Architectures for Modern Error Correcting Codes*, CRC Press, Jul. 2015.

[17] M. Mansour and N. Shanbhag, “A 640-Mb/s 2048-bit programmable LDPC decoder chip,” *IEEE Journ. of Solid-State Circuits*, vol. 41, no. 3, pp. 684-698, Mar. 2006.

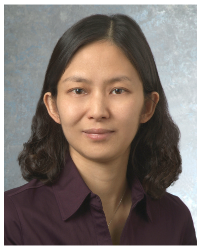
[18] L. Liu and C.-J. Shi, “Sliced message passing: high throughput overlapped decoding of high-rate low-density parity-check Codes,” *IEEE Trans. on Circuits and Syst.-I*, vol. 55, no. 11, pp. 3697-3710, Dec. 2008.

[19] Y.-L. Ueng, *et al.* “An efficient multi-standard LDPC decoder design using hardware-friendly shuffled decoding,” *IEEE Trans. on Circuits and Syst.-I* vol. 60, no. 3, pp. 743-756, Mar. 2013.

[20] X. Zhang and A. Bazarsky, “Perfect column-layered two-bit message-passing LDPC decoder and architectures,” *Proc. IEEE Intl. Symp. Circuits and Syst.*, pp. 1-5, May 2018.

Jiaxuan Cai (Graduate Student Member, IEEE) received his B.Eng. (Hons) degree in Computer Science & Technology from the College of Computer Science and the Hongshen Honors School of Chongqing University, China, in 2022. He is currently pursuing the Ph.D. degree with the department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH, USA.

His current research interest includes high-performance very-large-scale-integration (VLSI) architectures for information theory and its applications, specifically, the error-correcting codes in post-quantum cryptography.



Xinmiao Zhang (Senior Member, IEEE) received her Ph.D. degree in Electrical Engineering from the University of Minnesota. She is currently a Professor at the Ohio State University. She was a Senior Technologist at Western Digital/ SanDisk 2013-2017. Prior to that, she was a Timothy E. and Allison L. Schroeder Associate Professor at Case Western Reserve University. Prof. Zhangs research spans the areas of VLSI architecture design, digital storage and communications, cryptography, security, and signal processing.

Prof. Zhang is a recipient of the NSF CAREER Award 2009, the College of Engineering Lumley Research Award at The Ohio State University 2022, the Best Paper Award at ACM Great Lakes Symposium on VLSI 2004, and Best Paper Award at International SanDisk Technology Conference 2016. She authored VLSI Architectures for Modern Error-Correcting Codes (CRC Press, 2015), and co-edited Wireless Security and Cryptography: Specifications and Implementations (CRC Press, 2007). Prof. Zhang was elected the Vice President-Technical Activities of the IEEE Circuits and Systems Society (CASS) 2022-2023 and served on the Board of Governors of CASS 2019-2021. She was also the Chair (2021-2022) and a Vice-Chair (2017-2020) of the Data Storage Technical Committee (DSTC) of the IEEE Communications Society. She served on the technical program and organization committees of many conferences, including ISCAS, ICC, GLOBECOM, SiPS, GlobalSIP, MWSCAS, and GLSVLSI. She has been an associate editor for the IEEE Transactions on Circuits and Systems-I 2010-2019 and IEEE Open Journal of Circuits and Systems since 2019.