Low-Latency Parallel Row-Layered Min-sum MDPC Decoder for McEliece Cryptosystem

Jiaxuan Cai and Xinmiao Zhang
Dept. of Electrical & Computer Engineering, The Ohio State University, Columbus, OH 43210 U.S.A.
{cai.1072, zhang.8952}@osu.edu

***Abstract*—In the latest round of post-quantum cryptography standardization, the McEliece cryptosystem utilizing medium-density parity-check (MDPC) codes remains a candidate. The row-layered Min-sum decoding for MDPC codes has better trade-off on performance and complexity. Previous work adds constraints to the parity-check matrix construction in order to enable efficient parallel decoding. However, the constraints for large parallelism cause undesirable reduction on the number of usable secret keys and hence the previous scheme has limitations in achieving higher speed. This paper proposes two new schemes to substantially reduce the latency of row-layered MDPC decoding. Instead of further increasing the constraints to achieve higher parallelism, multiple identity blocks in the parity-check matrix are processed simultaneously in the first scheme and large blocks of variable width are processed in a hybrid way in the second design. Efficient hardware architectures are also developed for both proposed decoders. For an example code, the two proposed decoders achieve around 40% speedup compared to the best prior effort with less than 10% area overhead.**

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

I. Introduction

Existing cryptography standards are at risk from the imminent quantum computing attacks. The McEliece cryptosystem based on medium-density parity-check (MDPC) codes is a finalist for the post-quantum cryptography standard by the National Institute of Standards and Technology [1], [2]. The parity-check matrices of MDPC codes consist of a few very large random circulant matrices and serve as the secret keys. They have inherently different structures compared to those of low-density parity-check (LDPC) codes used for error correction. Hence, most existing methods for LDPC decoder optimization do not apply to MDPC decoders.

Previous literature on MDPC decoder design mainly considers the simple bit-flipping algorithm and its variations [3]–[10]. However, the Min-sum algorithm [11] can achieve orders of magnitude improvement in decoding failure rate (DFR) when proper scalar values are utilized [12], [13]. This enhances resistance against the reaction-based attacks that exploit decoding failures to recover the secret parity-check matrix [14], [15].

The Min-sum MDPC decoder in [12] achieves parallel processing by dividing the parity-check matrix into segments row-wise and processing the segments simultaneously. However, the random locations of the nonzero entries in the parity-check matrix limit the achievable speedup factor to a single digit and cause large memory overhead. The row-layered scheduling [16] is utilized in the MDPC decoder design in [13] to reduce the latency and memory requirement, which contributes to the majority of the decoder complexity. To effectively increase the speed of decoding with L rows in a layer, this design forces the distances between any two adjacent nonzero entries in a column to be at least L during the construction of the random parity-check matrix. Accordingly, the parity-check matrix is divided into blocks of $L \times L$ identity submatrices, and processing one submatrix in each clock cycle leads to L times speedup compared to a serial decoder. However, for a large L, such as 64, it takes a very long time to find a random secret parity-check matrix satisfying the constraint and the number of usable secret keys becomes insufficient. Besides, larger L leads to higher DFR, which makes the scheme more vulnerable to reaction attacks.

This paper proposes two schemes that achieve higher decoding speed without imposing constraints that undesirably affect the usability of the McEliece cryptosystem or increase its vulnerability to attacks. Our first design simultaneously processes multiple identity blocks within the same row layer in order to improve the parallelism. The memories storing the messages are divided into more banks to support such simultaneous processing and the effective achievable speedup is analyzed. In the second proposed scheme, the constraint on the minimum distance between adjacent nonzero entries, p, is set to be less than L, the number of rows included in a layer. Each row layer is dynamically divided into blocks with L rows but variable columns with a nonzero entry in the top left corner to simplify the computation units and on-the-fly address generation in the decoding process. A hybrid decoding scheduling scheme is also developed to eliminate the DFR degradation caused by having more than one nonzero entry in a column of a layer. Hardware implementation architectures are also developed for each decoder in this paper. For an example MDPC code constrained by $p = 32$, the two proposed decoders achieve around 40% speedup compared to the best prior effort [13] with less than 10% area overhead.

This paper is organized as follows. Section II introduces background information. The two proposed schemes are presented in Sections III and IV. Complexity analyses and conclusions follow in Section V and VI, respectively.

This material is based upon work supported by the National Science Foundation under Award No. 2052641.

II. Background

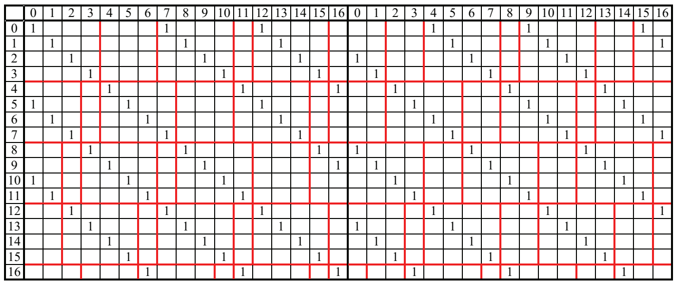
The MDPC code is characterized by its parity-check matrix $\mathbf{H}$. In the McEliece cryptosystem, the secret key comprises n_0 randomly generated vectors, each with r bits and a Hamming weight w. The structure of $\mathbf{H}$ is $[\mathbf{H}_0|\mathbf{H}_1|\cdots|\mathbf{H}_{n_0-1}]$, where $\mathbf{H}_i$ $(0 \leq i < n_0)$ is a circulant matrix whose initial column is the i-th random vector. The last vector needs to be randomly regenerated if $\mathbf{H}_{n_0-1}$ is non-invertible, so that the generator matrix $\mathbf{G}$ can be constructed as $[\mathbf{I}|[\mathbf{H}_{n_0-1}^{-1}\mathbf{H}_0|\mathbf{H}_{n_0-1}^{-1}\mathbf{H}_1|\cdots|\mathbf{H}_{n_0-1}^{-1}\mathbf{H}_{n_0-2}]^T]$. For the post-quantum cryptography standard, n_0 ranges between 2 and 4, r is between 3079 and 32771, and w is between 45 and 161, depending on the target security level.

The encryption process of the McEliece cryptosystem is mainly MDPC encoding. The $(n_0-1)r$-bit plaintext is multiplied with the generator matrix $\mathbf{G}$ to derive the n_0r-bit codeword $\mathbf{c}$, which is then added with a random vector, $\mathbf{e}$, with at most t nonzero bits to generate the ciphertext $\mathbf{x}$. Here t is a constant ranging from 42 to 264 depending on the parameters of the MDPC codes. The decryption process is to carry out MDPC decoding on $\mathbf{x}$. If $\mathbf{e}$ is a correctable error vector, then $\mathbf{c}$ is recovered. Its first $(n_0-1)r$ bits equal the plaintext since $\mathbf{G}$ is systematic.

Algorithm 1 Scaled Min-sum Decoding Algorithm [11]

```
1:  **Input:** $\mathbf{x} = [x_0, x_1, \cdots, x_{n-1}]$
2:  **Initialization:** $u_{i,j} = \gamma_j$
3:  **for** $k = 1$ 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**
```

A Tanner graph can be utilized to represent the $\mathbf{H}$ matrix. In this graph, variable nodes represent columns in the $\mathbf{H}$ matrix, check nodes correspond to rows, and edges between check and variable nodes denote nonzero entries in $\mathbf{H}$. Algorithm 1 [11] shows the Min-sum decoding algorithm. It iteratively updates and exchanges reliability information between connected check and variable nodes in the Tanner graph and makes decisions on received bits. For an input bit x_j $(0 \leq j < n)$, its reliability information, denoted as γ_j, is initially set to $+C$ or $-C$, when x_j is '0' or '1', respectively. C is a constant whose optimal value can be decided from simulations. Throughout this paper, $u_{i,j}$ represents the variable-to-check (v2c) message from variable node j to check node i, while $v_{i,j}$ represents the check-to-variable (c2v) message from check node i to variable node j. The decoding iteration number is denoted by k. $S_v(i)$ ($S_c(j)$) refers to the set of variable (check) nodes connected to check (variable) node i (j). In order to reduce the DFR, the sum of c2v messages is scaled by a factor, α, during the computation of v2c messages and *a posteriori* information, $\tilde{\gamma}_j$. A decoding failure is declared if no valid codeword is found after $I_{\max}$ iterations.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1							1					1									1					1						1	
1		1							1					1									1					1						1
2			1							1					1			1						1					1					
3				1							1					1			1						1					1				
4					1							1					1			1						1					1			
5	1					1							1								1						1					1		
6		1					1							1								1						1					1	
7			1					1							1								1						1					1
8				1					1							1		1						1						1				
9					1					1							1		1						1						1			
10	1					1					1									1						1						1		
11		1					1					1									1						1						1	
12			1					1					1									1						1						1
13				1					1					1				1					1						1					
14					1					1					1				1					1						1				
15						1					1					1				1					1						1			
16							1					1					1				1					1						1		

Fig. 1. Example $\mathbf{H}$ matrix for a toy MDPC code with $(n_0, r, w) = (2, 17, 3)$ and constraint $p = 4$ divided into identity blocks [13].

The row-layered decoding scheduling scheme [16] achieves faster convergence and reduces the memory requirement. In this scheme, $\mathbf{H}$ is partitioned into blocks of L rows and each block row is referred to as a layer. Additionally, each column in a layer can have no more than one nonzero entry. While decoding layer f in iteration k, the v2c and c2v messages corresponding to a nonzero entry in $\mathbf{H}$ are represented as $u^{(k,f)}$ and $v^{(k,f)}$, respectively. The row-layered decoding utilizes the most updated c2v messages to generate the v2c messages. It is achieved by updating the *a posterior* information based on the most recent c2v messages as

$$\tilde{\gamma}' = \tilde{\gamma} - \alpha v^{(k-1,f)} + \alpha v^{(k,f)}. \tag{1}$$

Comparing Lines 14 and 16, the v2c message utilizing the latest c2v messages is calculated as

$$u^{(k,f)} = \tilde{\gamma} - \alpha v^{(k-1,f)}. \tag{2}$$

In summary, in row-layered decoding, (1) and (2) replace Lines 16 and 14, respectively, of Algorithm 1.

Since the locations of the nonzero entries of $\mathbf{H}$ are random, dividing $\mathbf{H}$ into blocks of rows and columns in a straightforward manner leads to very limited achievable speedup and high hardware complexity [12]. To enable more efficient parallel processing, it was proposed in [13] to add the constraint that the distance between any pair of adjacent nonzero entries in each of the n_0 r-bit random vectors needs to be at least p. Then $\mathbf{H}$ can be divided dynamically into identity blocks of size $p \times p$. Processing one identity block in each clock cycle effectively achieves p times speedup compared to a serial decoder. A toy example of the $\mathbf{H}$ matrix with $p = 4$ and dynamic division is shown in Fig. 1.

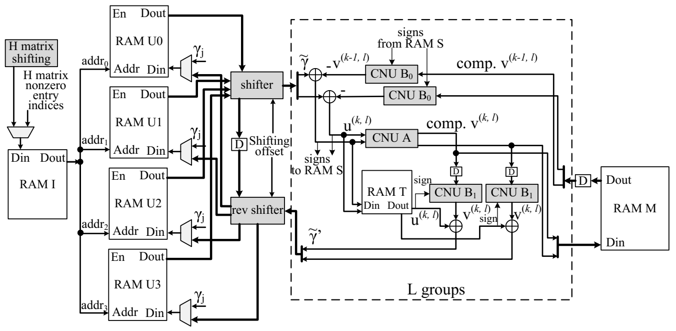


Fig. 2. Top-level architecture of row-layered Min-sum MDPC decoder processing multiple identity blocks simultaneously.

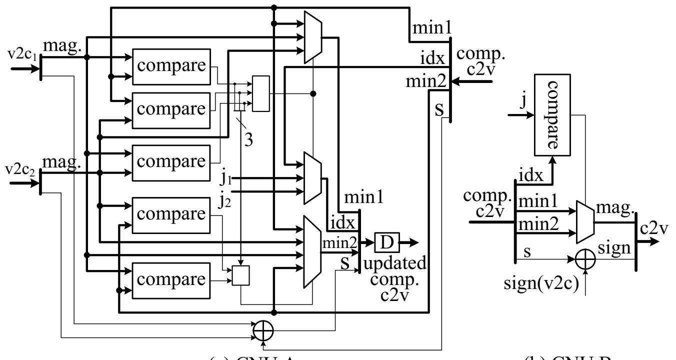

(a) CNU A (b) CNU B

Fig. 3. (a) CNU A capable of processing two input v2c messages to update $min1$, $min2$, idx and s; (b) CNU B for generating c2v messages [17].

III. Multi-Identity Block Parallel Processing

Higher parallelism is needed to further reduce the decoding latency. The parallelism in the design of [13] is determined by the constraint p. It is becoming very difficult to find an $\mathbf{H}$ satisfying a larger p, such as 64. Moreover, the number of secret keys satisfying a larger p constraint reduces dramatically and the DFR may also increase with p. Both of these issues compromise the security of the McEliece cryptosystem. Two new parallel decoders are proposed in this and the next sections to achieve substantially higher processing speed without further increasing p.

Our first proposed decoder utilizes the same constraint and dynamic matrix division as those in [13], but processes an integer $m_1 > 1$ identity blocks in the same layer each time. This effectively increases the processing speed without further increasing p. Besides, since there is also at most one nonzero entry in each column in a layer, our design achieves the same DFR as that of the previous layered decoder with the same layer size.

The block diagram in Fig. 2 presents our first proposed parallel decoder for an example case of $m_1 = 2$. The check node unit (CNU) for computing c2v messages from v2c messages can be found in many literature, such as [17]. The details of the CNU for $m_1 = 2$ are shown in Fig. 3. It is divided into two parts. CNU A is capable of processing $m_1 = 2$ v2c messages from two identity blocks at a time. It iteratively calculates $min1$, $min2$, idx, and s following Lines 6-9 of Algorithm 1. They are stored in RAM M in Fig. 2 as compressed c2v messages. Since the L rows of $\mathbf{H}$ in a layer are processed simultaneously, the compressed messages for L rows are stored at each RAM M address. To shorten the data path, all necessary comparisons are done in parallel in CNU A using 5 comparators. The CNU B in Fig. 3 (b) derives an actual c2v message from the compressed form as listed in Lines 11 and 12 of Algorithm 1. Since our design processes L rows in a layer simultaneously, L copies of CNU A and two groups of CNU B are adopted. The first and second groups, denoted by CNU B_0 and CNU B_1, in Fig. 2, recover the $v^{(k-1,f)}$ and $v^{(k,f)}$, respectively, in (1). Each group has $m_1 L$ copies of the architecture in Fig. 3(b).

TABLE I
Speedup factor compared to serial decoder achieved by the first proposed design for example MDPC codes with $(n_0, r, w) = (2, 4801, 45)$ and $p = 32$.

# of RAM U banks (depth)	m_1		
	2	3	4
2 (151)	33.1	-	-
3 (101)	36.1	38.1	-
4 (76)	44.3	50.5	53.1

In Fig. 2, RAM U is initialized by the channel information and is then overwritten by the most updated *a posteriori* information during the decoding process. To simplify the information storage and facilitate the parallel processing of each layer, the decoder input bits are divided into $L = p$-bit groups and the *a posteriori* information for each group is stored in an address of RAM U. In the dynamic matrix division scheme, an identity block can start from any column of $\mathbf{H}$, and does not necessarily align with the information stored in a single address of RAM U. In the design of [13], RAM U is divided into two banks for the even and odd addresses, and the messages for one identity block can always be extracted from two addresses, one from each memory bank. The shifter in Fig. 2 is used to assemble the messages from different RAM U banks into those for the identity blocks. However, to process $m_1 > 1$ identity blocks at once, RAM U needs to be divided into more banks to reduce the memory access conflicts. On the other hand, shallow memories occupy larger silicon area per bit in many CMOS processes. This is because the sense amplifiers do not scale down much for shallower memories and they occupy a significant portion of the RAM area. The memory access conflict and RAM silicon size need to be jointly considered.

Table I shows the speedup that can be achieved by our first proposed design compared to a serial decoder for example MDPC codes with $(n_0, r, w) = (2, 4801, 45)$ considered for the standard and constraint $p = 32$. The CNUs handling a larger number of inputs may have a longer data path. However, they can be pipelined to achieve the same data path as that of the CNU handling one input each time at the cost of $\lceil r/L \rceil$ extra clock cycles for each decoding iteration. The speedup achievable by our design shown in Table I is derived by counting the number of clock cycles needed for each iteration averaged over 10 randomly generated MDPC codes. Denote the number of RAM U banks by b. If $b > 4$ for the example code considered in Table I, then each bank only has a depth of less than 60. To reduce the size of RAM U, it is divided into at most $b = 4$ banks in our consideration. Since each RAM U bank stores the information of $L = p$ bits in each address, the number of $p \times p$ identity blocks for which the information can be read in each clock cycle does not exceed b. Therefore, having $m_1 > b$ does not bring further speedup.

If the *a posteriori* information for all the m_1 identity blocks being processed at a time is located in distinct RAM U banks, then all the information can be read out and the m_1 blocks are processed in one clock cycle. However, it is also possible that some of the *a posteriori* information for the m_1 blocks is located in different addresses of the same RAM U bank and there are memory access conflicts, especially for larger m_1 and/or smaller b. In this case, the CNUs wait for more clock cycles until all the information for the m_1 blocks is read out. Because of the memory access conflicts, the achievable speedup is much lower than $m_1 L$. As shown in Table I, for the same b, the additional speedup achievable by using larger m_1 is decreasing. Besides, larger m_1 substantially increases the number of comparators in CNU A and requires additional copies of CNU B. For the same m_1, having more RAM U banks helps to further increase the speed. However, RAM U should not be divided into too many banks so that the depth of the each bank is very small. In the example decoder architecture illustrated in Fig. 2, four RAM U banks are adopted.

To reduce the memory requirement, only the column index of the '1' in the top left corner of each identity block in the first layer of $\mathbf{H}$ is stored in RAM I in the beginning of the decoding as shown in Fig. 2. During the decoding, the column indices for the next layer are derived from those of the current layer by adding L mod r in the "H matrix shifting" block and the results are written back to RAM I. RAM S is used to store the sign bits of the v2c messages. $u^{(k,l)}$ is buffered in RAM T until it is added up with $\alpha v^{(k,l)}$ to update the *a posteriori* information.

IV. Variable-Width Large Block Parallel Processing

Our first proposed scheme presented in the previous section has the layer size L equal to the constraint p on the $\mathbf{H}$ matrix and tries to process $m_1 p$ nonzero entries from m_1 identity blocks in the L rows of the same layer simultaneously. To circumvent the memory access conflict issue of the first proposed design, this section proposes an alternative design that puts $L > p$ rows of $\mathbf{H}$ in a layer. Each block row is divided into blocks with variable widths that start with a nonzero entry in the top left corner. Many of these larger blocks can be processed in one clock cycle due to the sparsity of the code and hence this second design also achieves effective speedup.

To simplify the generation of the addresses for accessing RAM U and M, p and L are set to be integer powers of 2. To reduce the complexity of generating the locations of the

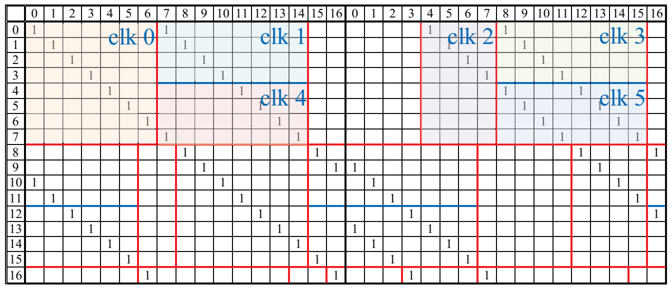


Fig. 4. Example $\mathbf{H}$ matrix divided by the proposed variable-width dynamic division scheme for a toy MDPC code with $(n_0, r, w) = (2, 17, 3)$ and constraint $p = 4$, with blue lines indicating hybrid processing scheme divisions.

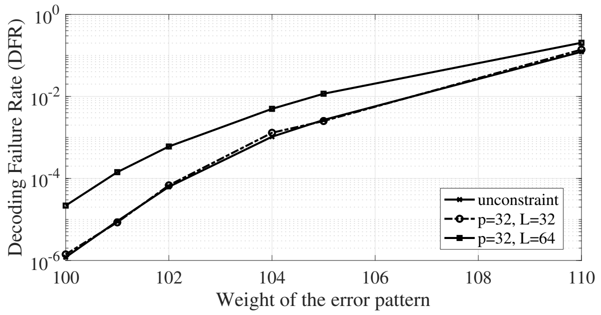


Fig. 5. DFRs of MDPC codes with $(n_0, r, w) = (2, 4801, 45)$ and $I_{\max} = 30$ with and without code construction constraints using different layer size.

nonzero entries of $\mathbf{H}$ on the fly, a variable-width dynamic division scheme for the $\mathbf{H}$ matrix is proposed in our design, as shown in Fig. 4 for a toy MDPC code with $p = 4$ and $L = 2p$. In our scheme, each block starts from a column with a nonzero entry in the first row of the layer, and ends after L columns or before the next column that has a nonzero entry in the first row, whichever comes first. A block may contain up to $m_2 = \lceil L/p \rceil$ diagonal lines, but each of them starts in the first column of the block. In this case, the starting columns of the blocks in the next layer are derived by adding $L \mod r$ to those of the current layer, and the starting rows of the diagonal lines within the block do not change in the corresponding blocks.

The row-layered scheme [16] originally allows only one nonzero entry in each column of the layer and the formula for updating the v2c messages in (2) is developed for this case. When there is more than one nonzero entry in a column, the v2c messages for the second and later entries are not based on the most updated information from every check node. As a result, it causes degradation on the DFR. For LDPC codes, the degradation is very small [18], [19]. However, it becomes much more significant for MDPC codes due to their relative higher density. Fig. 5 shows the DFRs of a randomly generated MDPC code with $(n_0, r, w) = (2, 4801, 45)$ and maximum iteration number $I_{max} = 30$. As shown in [13], adding constraint that is small or moderate, such as $p = 32$, in the code construction does not lead to noticeable performance loss. However, when using $L > p$, the DFR is increased significantly. Simulations have also been run for several other

TABLE II
SPEEDUP FACTORS COMPARED TO SERIAL DECODER ACHIEVED BY THE SECOND PROPOSED DESIGN FOR EXAMPLE MDPC CODES WITH $(n_0, r, w) = (2, 4801, 45)$ AND VARIOUS p AND L.

p \ L	32	64	128
4	24.7	39.6	58.6
8	26.3	41.9	60.0
16	27.9	43.9	63.4
32	-	46.2	65.5

randomly generated codes. Although the performance loss is not always as big as that shown in Fig. 5, some of them are still noticeable.

To eliminate the performance degradation, a hybrid processing scheme is proposed in our design. Each block with multiple nonzero entries in the same column is divided horizontally into segments, each with no more than one nonzero entry in a column, as shown by the blue lines in Fig. 4. Each segment is processed in a separate clock cycle. In the processing of a layer, if a block is divided into more than one segment, the later segments are processed after the last block in the layer is gone over, so that the most updated c2v messages are always incorporated. The clock cycles in which the blocks/segments of the first layer are processed are labeled in Fig. 4. Since each updated message is generated using exactly the same formula in (2), our hybrid processing scheme does not lead to any DFR degradation.

Although the blocks with more than one nonzero entry in a column need more clock cycles to process, they contribute to a small percentage of all the blocks. Many blocks have only one nonzero entry in each column and more than p nonzero entries are processed each time. As a result, our proposed scheme can achieve more than p times speedup compared to a serial design as listed in Table II. For various combinations of p and L, 10 randomly generated MDPC codes with $(n_0, r, w) = (2, 4801, 45)$ were examined, and their average speedup factors are shown in the table. From the data in Table II, larger values of L and p result in higher speedup. However, very large L should be avoided since it will lead to shallow memories for storing the compressed c2v, *a posteriori*, and other messages. Although larger p only leads to small improvement in speed, the largest p that does not undesirably affect the security level or key search time should be utilized.

The overall decoder architecture for implementing the hybrid processing scheme with variable-width blocks is very similar to that shown in Fig. 2 with the following three differences: i) the locations of the nonzero entries of $\mathbf{H}$ are generated differently as described in the second paragraph of this section; ii) a block in this scheme may include up to m_2 diagonal lines. However, as it can be observed from Fig. 4, each of the first p rows always has one nonzero entry. Hence, the first p copies of CNUs only need to handle one input each, while the others have m_2 inputs each; iii) m_2 shifters and reverse shifters are needed to align the messages for the up to m_2 diagonal lines in each block.

TABLE III
THE NUMBER AND SIZES OF THE RAM BANKS UTILIZED IN THE PROPOSED DECODERS AND THAT FROM [13] FOR (n_0, r, w) MDPC CODES WITH CONSTRAINT p, q-BIT C2V AND V2C MESSAGE MAGNITUDE, AND a-BIT *a posteriori* INFORMATION MAGNITUDE.

RAM name	[13]	First design (this paper)	Second design (this paper)
RAM I	1 bank, $n_0w \times \lceil \log_2(n_0r) \rceil$	1 bank, $\lceil n_0w/m_1 \rceil \times m_1 \lceil \log_2(n_0r) \rceil$	1 bank, $n_0w \times (\lceil \log_2(n_0r) \rceil + (m_2 - 1)(\lceil log_2 p \rceil + 1))$
RAM M	1 bank, $\lceil r/p \rceil \times p(2q + 1 + \lceil log_2(n_0r) \rceil)$	1 bank, $\lceil r/p \rceil \times p(2q + 1 + \lceil log_2(n_0r) \rceil)$	1 bank, $\lceil r/m_2p \rceil \times m_2p(2q + 1 + \lceil log_2(n_0r) \rceil)$
RAM S	1 bank, $n_0w \lceil r/p \rceil \times p$	1 bank, $\lceil n_0w/m_1 \rceil \lceil r/p \rceil \times m_1p$	1 bank, $n_0w \lceil r/m_2p \rceil \times m_2p$
RAM U	2 banks, $n_0 \lfloor r/2p \rfloor \times p(a + 1)$	b banks, $n_0 \lfloor r/bp \rfloor \times p(a + 1)$	2 banks, $n_0 \lfloor r/2m_2p \rfloor \times m_2p(a + 1)$
RAM T	p banks, $n_0w \times (a + 1)$	p banks, $m_1n_0w \times (a + 1)$	p banks, $m_2n_0w \times (a + 1)$

V. HARDWARE COMPLEXITY COMPARISONS

This section analyzes the hardware complexities of the two proposed parallel MDPC decoders and compares them with the best previous parallel decoder that processes a single identity block of $p \times p$ per clock cycle [13].

Memories are the primary contributors to the silicon area of the overall MDPC decoder. Table III provides a summary of the sizes for all RAMs involved in the decoders for (n_0, r, w) MDPC code. The RAM I of the second design is larger compared to that in [13] because it stores both the starting column index of the first diagonal and starting row indices of the other diagonals in the same block. Both the first and second designs utilize the same number of RAM T banks as in [13] but require the size of each RAM T bank to be m_1 and m_2 times of that in [13], respectively. This is because the two designs need to process at most m_1 or m_2 nonzero entries instead of a single entry at a time. Although the two proposed designs may have different depths, widths, or number of banks for the other RAMs, the overall sizes of those RAMs are almost the same as those in [13].

To better show the relative sizes of the memories, they are listed in Table IV for an example MDPC code with $(n_0, r, w) = (2, 4801, 45)$ and $p = 32$. It is assumed that $q = 4$ bits are used to represent the magnitude of each c2v and v2c message, and each *a posteriori* message is represented by 11 bits. When the CNUs need to handle more than two inputs, their complexities become much higher. Hence m_1 and m_2 are set to 2 in our analyses and comparisons. To avoid using very shallow memories, RAM U is divided into $b = 4$ banks in the first decoder. The proposed designs are compared with the parallel decoder that processes a single identity block of 32×32 per clock cycle [13]. Although the overall size of RAM U is the same in terms of the number of bits stored, the one in the first proposed design has 4 instead of 2 banks and hence the depth is halved. Compared to the design in [13], the first

TABLE IV
COMPLEXITY COMPARISONS FOR THE PROPOSED DECODERS WITH $m_1 = m_2 = 2$ AND THAT FROM [13] FOR AN MDPC CODE WITH $(n_0, r, w) = (2, 4801, 45)$ AND $p = 32$.

	[13]	First design ($m_1 = 2$)	Second design ($m_2 = 2$)
RAM I (bits)	1260	1260	1800
RAM M (bits)	111136	111136	111872
RAM S (bits)	434880	434880	437760
RAM U (bits)	105600	105600	105600
RAM T (bits)	31680	63360	63360
Total memory (bits) (normalized)	684556 (1)	716236 (1.05)	720392 (1.05)
Logic (# of XORs)	15342	22054	36678
Total Area (# of XORs) (normalized)	528759 (1)	559231 (1.06)	576972 (1.09)
Speedup compared to serial design (normalized)	32 (1)	44.3 (1.38)	46.2 (1.44)

proposed design has twice the size in RAM T because up to $m_1 = 2$ instead of a single v2c message is processed by each CNU in each clock cycle. The second proposed design has larger RAM I because it needs to record the information for up to $m_2 = 2$ diagonal lines for each block in the $\mathbf{H}$ matrix. Since the layer size is $L = m_2 p = 2p$, its depth of RAM M is a half compared to that of the first design. Similarly, it needs RAM U of half depth and RAM T twice in size compared to the design in [13]. Overall, the memory sizes of both proposed designs are only 5% bigger than that of [13].

The first proposed design employs $p = 32$ copies of the CNU A in Fig. 3(a) and $2m_1 p = 128$ copies of the CNU B in Fig. 3(b). As mentioned previously, only the $(m_2 - 1)p = 32$ of the $L = m_2 p = 64$ CNUs utilized in the second proposed decoder need to handle two inputs at a time. Hence, it uses $p = 32$ copies of the CNU A in Fig. 3(a), $(m_2 - 1)p = 32$ copies of CNU A handling single input [17], and $2p + 2(m_2 - 1)m_2 p = 192$ copies of CNU B. The number of adders in the overall decoder architecture shown in Fig. 2 equals the number of copies of CNU B. The complexities of the CNUs, adders, shifters, and multiplexors in Fig. 2 are analyzed and listed as the logic complexity in Table IV. Assuming that the area for storing one bit in memory equals that of 0.75 XOR gates [10], the total area in terms of the number of XOR gates needed for each design is also computed and listed. The two proposed designs only incur less than 10% area overhead.

Although the data path of the CNU A handling two inputs is longer than that of the CNU A with one input used in [13], the data path for adding the *a posterior* information is much longer. Hence the two proposed designs achieve the same critical path as the design in [13]. As a result, the achievable speedup is decided by the number of clock cycles needed for the decoder. Compared to the design in [13], our two proposed designs can achieve around 40% speedup.

VI. CONCLUSIONS

This paper tackles the limitation on the achievable parallelism in previous row-layered MDPC decoders for the McEliece cryptosystem. Without increasing the constraint on code construction, two schemes are proposed in this work to process more entries of the parity check matrix at a time. The first design processes multiple identity blocks concurrently and the second utilizes larger blocks with variable width and a hybrid processing scheme. Both of them achieve substantial improvement on the decoding speed with small hardware overheads and no DFR degradation. Future work will try to further reduce the decoding latency and memory requirement.

REFERENCES

[1] D. J. Bernstein, *et al.*, "Classic McEliece: conservative code-based cryptography," NIST, Nov. 2017.

[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] H. Bartz and G. Liva, "On decoding schemes for the MDPC-McEliece cryptosystem," *Proc. of Intl. ITG Conf. on Syst., Commun., and Coding*, pp. 1-6, Mar. 2019.

[4] 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.

[5] 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.

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

[7] I. V. Maurich and T. Güneysu, "Lightweight code-based cryptography: QC-MDPC McEliece encryption on reconfigurable devices," *Proc. IEEE Design, Autom. & Test in Europe Conf. & Exhib.*, pp. 1-6, Mar. 2014.

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

[9] 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.

[10] Z. Xie and X. Zhang, "Sparsity-aware medium-density parity-check decoder for McEliece cryptosystems," *IEEE Trans. on Circuits and Syst.-II*, vol. 70, no. 9, pp. 3343-3347, Sep. 2023.

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

[12] J. Cai and X. Zhang, "Low-complexity parallel Min-sum medium-density parity-check decoder for McEliece cryptosystem," *IEEE Trans. on Circuits and Syst.-I*, Sep. 2023.

[13] J. Cai and X. Zhang, "Highly efficient parallel row-layered Min-Sum MDPC decoder for McEliece cryptosystem," *IEEE Trans. on Circuits and Syst.-I*, under review, 2023.

[14] Q. Guo, T. Johansson, and P. Stankovski, "A key recovery attack on MDPC with CCA security using decoding errors," *Proc. ASIACRYPT*, pp. 789-815, Dec. 2016.

[15] 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 Cryptogr.: 7th Intl. Workshop*. pp. 115-136, Jul. 2019.

[16] 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.

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

[18] X. Zhang and Y. Tai, "High-speed multi-block-row layered decoding for Quasi-cyclic LDPC codes," *Proc. IEEE Global Conf. on Signal and Info. Processing*, pp. 11-14, Dec 2014.

[19] Y. Sun, G. Wang and J. Cavallaro, "Multi-layer parallel decoding algorithm and VLSI architecture for quasi-cyclic LDPC codes," *Proc. IEEE Intl. Symp. Circuits and Syst.*, pp. 1776-1779, May 2011.