# Tree-Permutation-Matrix Based LDPC Codes

Sheng Jiang, Fanlu Mo, Francis C.M. Lau, *Senior Member, IEEE* and Chiu-W. Sham, *Member, IEEE*

***Abstract*—Low-density parity-check (LDPC) codes are normally categorized into random structure or regular structure. In this paper, we introduce a new type of LDPC codes which is of semi-regular style. The parity-check matrices of the new LDPC code type are composed of sub-matrices termed tree-permutation matrices (TPMs). These TPMs are "semi-regular" and are constructed in a systematic way. Using the $2 \times 2$ identity matrix and anti-diagonal matrix as an example, we illustrate how $2^M \times 2^M$ TPMs are formed. During the formation of the $2^M \times 2^M$ TPMs, we further apply the hill-climbing algorithm to avoid short cycles. Finally, we construct a girth-8 TPM-LDPC code with a base matrix of size $4 \times 24$ and a girth-10 TPM-LDPC code with a base matrix of size $3 \times 10$. We implement the TPM-LDPC decoders on a FPGA and compare the simulation results and decoder complexity with other LDPC codes.**

***Index Terms*—FPGA implementation, low-density parity-check code, tree-permutation matrix**

## I. Introduction

As one of the two known classes of Shannon limit-approaching codes, low-density parity-check (LDPC) code has been widely studied and used [1–4]. Most commonly used LDPC codes are of quasi-cyclic nature. The parity-check matrices of quasi-cyclic LDPC (QC-LDPC) codes use circulant permutation matrices (CPMs) as their sub-matrices [1, 2, 5]. Due to the regular structure of these parity-check matrices, simple and high-throughput encoders/decoders can be implemented [3, 6, 7]. LDPC codes can also be constructed from random matrices or random permutation sub-matrices [8, 9]. While better error performance could be obtained, encoder/decoder complexity and throughput are the major issues. For example, the FPGA simulation results in [9] have shown that random-permutation-matrix-based cyclically-coupled LDPC (RP-CC-LDPC) codes can outperform cyclically-coupled QC-LDPC (CC-QC-LDPC) codes in terms of bit error performance. However, the RP-CC-LDPC decoder requires much more memory storage and does not support parallel decoding compared with the CC-QC-LDPC decoder.

In this paper, we propose a new type of matrices called *tree-permutation-matrices* (TPMs) and apply them in the construction of LDPC codes. TPMs are not as regular as CPMs but also not as random as random-permutation matrices. Thus we describe TPMs as *semi-regular* matrices. Like QC-LDPC codes, TPM-based LDPC codes allow decoding operations to be conducted in parallel and can achieve a high decoding throughput. Moreover, there are more choices of TPMs compared with CPMs for a given matrix size. Hence, a larger girth and/or a fewer number of short cycles can potentially be achieved for TPM-based LDPC codes.

In Sect. II, we will introduce a way to construct TPMs, representations and characteristics of TPMs, and simple algorithms for computing the products and transpose of TPMs. In Sect. III, we define TPM-LDPC codes and show the sufficient condition for cycles to exist. After that we propose an efficient way to construct a high-girth TPM-LDPC code systematically. Then we explain how to avoid the RAM access conflicts in the decoder design. Finally in Sect. IV, we show the simulation results of our proposed TPM-LDPC codes and compare the results and decoder complexity with other regular and irregular LDPC codes.

## II. Tree Permutation Matrices

We first make use of $2 \times 2$ permutation matrices to illustrate how to construct TPMs. There are only two different $2 \times 2$ permutation matrices — the identity $2 \times 2$ matrix (denoted by $\mathbf{I}_{2\times2}$) and the anti-diagonal $2 \times 2$ matrix (denoted by $\tilde{\mathbf{I}}_{2\times2}$). To construct TPMs, we refer to Fig. 1 and start with a matrix that contains only a single element "1" as shown at the top.

1) To construct Layer-1 TPMs, we replace the '1' at the top with either $\tilde{\mathbf{I}}_{2\times2}$ ($a_{1,0} = 1$) or $\mathbf{I}_{2\times2}$ ($a_{1,0} = 0$). There are only two possibilities and hence $N_1 = 2$ possible Layer-1 TPMs, the size of which are $2 \times 2$.
2) To construct Layer-2 TPMs, the 1's in each Layer-1 TPM are replaced with either $\tilde{\mathbf{I}}_{2\times2}$ or $\mathbf{I}_{2\times2}$. For each Layer-1 TPM, there are $2^2 = 4$ possible choices ($a_{2,0}a_{2,1} = 11, 10, 01, 00$). Hence the total number of Layer-2 TPMs equals $N_2 = 2^2 \times N_1 = 2^3 = 8$. The size of each Layer-2 TPM is $2^2 \times 2^2$.
3) To construct Layer-3 TPMs, the 1's in each Layer-2 TPM are replaced with either $\tilde{\mathbf{I}}_{2\times2}$ or $\mathbf{I}_{2\times2}$ again. For each Layer-2 TPM, there are $2^{2^2} = 16$ possible choices. Hence the total number of Layer-3 TPMs equals $N_3 = 2^{2^2} \times N_2 = 2^7 = 128$. The size of each Layer-3 TPM is $2^3 \times 2^3$.
4) TPMs in subsequent layers are generated in a similar manner. It can be easily shown that at Layer $M$, there are $2^{2^M-1}$ different TPMs, the size of which are $2^M \times 2^M$. The tree expansion is illustrated in Fig. 2. We also denote $P_2^M$ as a Layer-$M$ TPM matrix formed by $\tilde{\mathbf{I}}_{2\times2}$ and $\mathbf{I}_{2\times2}$.

In our study, we consider only TPMs formed by $\tilde{\mathbf{I}}_{2\times2}$ and $\mathbf{I}_{2\times2}$. In general, TPMs can be constructed by $Z \times Z$ permutation matrices. In Fig. 3, we further show the expansion tree if every '1' in the previous layer is replaced by a $Z \times Z$

The work described in this paper was supported by a grant from the RGC of the Hong Kong SAR, China (Project No. PolyU 521809).

Sheng Jiang, Fanlu Mo and Francis C.M. Lau are with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong. Chiu-W. Sham is with the Department of Computer Science, The University of Auckland, New Zealand. (Emails: sheng.jiang@connect.polyu.hk, fanlu.mo@connect.polyu.hk, francis-cm.lau@polyu.edu.hk, b.sham@auckland.ac.nz)

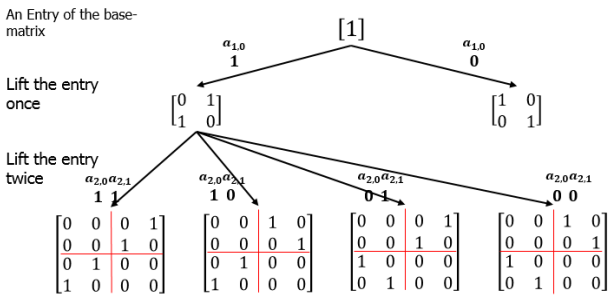


Fig. 1. Forming tree-permutation matrices by replacing each '1' in the upper layer with a $2 \times 2$ permutation matrix.

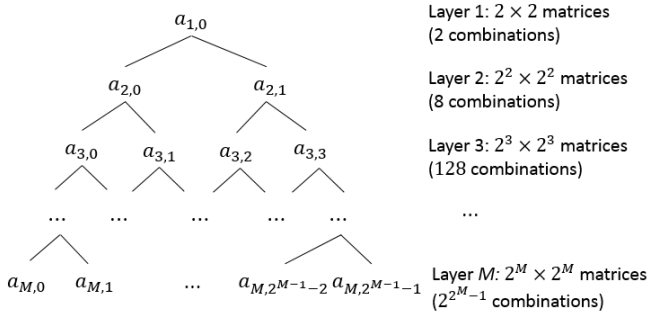


Fig. 2. The full tree representation of $P_2^M$.

permutation matrix. At Layer $M$, each TPM will have a size of $Z^M \times Z^M$.

Referring to Fig. 2, a Layer-$M$ TPM formed by expanding $2 \times 2$ permutation matrices repeatedly can be defined by the "tree" vector

$$\mathbf{V} = (a_{1,0}, a_{2,0}, a_{2,1}, a_{3,0}, a_{3,1}, a_{3,2}, a_{3,3}, \ldots, a_{M,0}, a_{M,1}, \ldots, a_{M,2^{2^M-1}-1}) \quad (1)$$

in which each element is either a "1" or "0". When the element assumes the value "1" and "0", it represents an expansion with $\tilde{\mathbf{I}}_{2\times2}$ and $\mathbf{I}_{2\times2}$. respectively. It can be readily shown that

- *a $2^M \times 2^M$ identity matrix will be associated with the all-zero tree vector with $2^M - 1$ elements; and*
- *a $2^M \times 2^M$ TPM has a fixed column[1] if and only if at least one branch transversing from the top to the bottom of the tree (in Fig. 2) assumes all "0" values, e.g., the branch $a_{1,0} = a_{2,0} = a_{3,0} = \cdots = a_{M,0} = 0$.*

[1]A permutation matrix has a fixed column (or row) if and only if it overlaps with the identity matrix in at least one column (or row) [10].

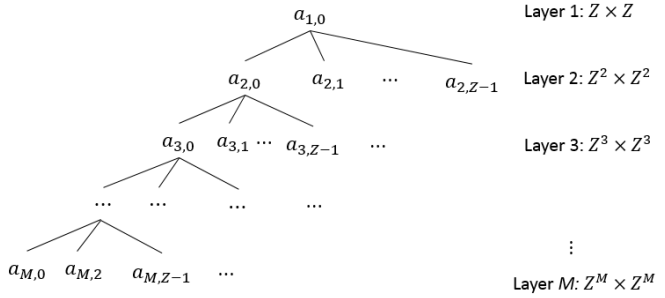


Fig. 3. The general tree representation of a tree permutation matrix.

**Algorithm 1** Multiplication of two $P_2^M$ TPMs

1: $\mathbf{P}_M[0] \leftarrow \mathbf{P}_A[0] \oplus \mathbf{P}_B[0], \mathbf{V}[0] \leftarrow 0$
2: **for** $node = 1; node < 2^M - 2; node++$ **do**
3: $\quad parent \leftarrow (node - 1)/2$
4: $\quad$ **if** $node$ is odd **then**
5: $\quad\quad \mathbf{V}[node] \leftarrow \mathbf{V}[parent] \times 2 + \mathbf{P}_A[parent]$
6: $\quad$ **else**
7: $\quad\quad \mathbf{V}[node] \leftarrow \mathbf{V}[parent] \times 2 - \mathbf{P}_A[parent]$
8: $\quad \mathbf{P}_M[node] \leftarrow \mathbf{P}_A[node] \oplus \mathbf{P}_B[node + \mathbf{V}[node]]$

### A. Multiplication of TPMs

It can be easily verified that TPMs formed by $2 \times 2$ permutation matrices are closed under multiplication. One way is to consider all possible multiplications and then build a look-up table to simplify the computation. The table can be large depending on the value of $M$. On the other hand, each TPM is represented by a tree vector. To find the product of two TPMs, we can make use of their corresponding tree vectors, perform appropriate and simple module-2 additions, and arrive at a new tree vector that represents the TPMs' product. The procedures to compute the product of two TPMs with size $2^M \times 2^M$ are shown in Algorithm 1. In Algorithm 1, $\mathbf{P}_A$ and $\mathbf{P}_B$ denote the tree vectors of the two TPMs to be multiplied; $\mathbf{V}$ is a vector used to locate the corresponding elements in the two tree vectors; and $\mathbf{P}_M$ represents the tree vector of the product of the two TPMs.

### B. Transpose of a TPM

Similar to multiplication, the transpose operation on a TPM can be conducted effectively based on its tree vector. The transpose of a TPM with size $2^M \times 2^M$ is computed with the method shown in Algorithm 2. In Algorithm 2, $\mathbf{P}$ and $\mathbf{P}^T$ represent the tree vectors of a TPM and its transpose, respectively; and $\mathbf{V}$ is a vector used to locate the corresponding elements in the two tree vectors.

**Algorithm 2** Transpose of a $P_2^M$ TPM

$\mathbf{P}^T[0] \leftarrow \mathbf{P}[0], \mathbf{V}[0] \leftarrow 0$
**for** $node = 1; node < 2^M - 2; node++$ **do**
  $parent \leftarrow (node - 1)/2$
  **if** $node$ is odd **then**
    $\mathbf{V}[node] \leftarrow \mathbf{V}[parent] \times 2 + \mathbf{P}[parent]$
  **else**
    $\mathbf{V}[node] \leftarrow \mathbf{V}[parent] \times 2 - \mathbf{P}[parent]$
  $\mathbf{P}^T[node + \mathbf{V}[node]] \leftarrow \mathbf{P}[node]$

## III. TPM-BASED LDPC CODES

We define a TPM-based parity-check matrix $\mathbf{H}_{\rm TPM}$ as

$$\mathbf{H}_{\rm TPM} = \begin{bmatrix} \mathbf{T}_{0,0} & \mathbf{T}_{0,1} & ... & \mathbf{T}_{0,L-1} \\ \mathbf{T}_{1,0} & \mathbf{T}_{1,1} & ... & \mathbf{T}_{1,L-1} \\ ... & ... & & ... \\ \mathbf{T}_{J-1,0} & \mathbf{T}_{J-1,1} & ... & \mathbf{T}_{J-1,L-1} \end{bmatrix} \quad (2)$$

where each $\mathbf{T}_{i,j}$ indicates a TPM matrix, and $J$ and $L$ correspond to the number of rows and columns of the parity-check base matrix. We further define the corresponding LDPC code of the TPM-based parity-check matrix as a TPM-based LDPC code.

### A. Cycle Evaluation

To evaluate the cycles of a TPM-based LDPC code, we can apply the following theorem [11].

*Theorem 1:* Let $\mathcal{C}$ be a code which can be described by a parity-check matrix $\mathbf{H} = (\mathbf{Q}_{i,j})$, where the $(i, j)$-th entry $\mathbf{Q}_{i,j}$ represents a $q \times q$ permutation matrix. If there exists a cycle of length *2l* which including the indices $i_0, i_1, \ldots, i_{l-1}$ and $j_0, j_1, ..., j_{l-1}$ ($i_s \neq i_{s+1}$, $j_s \neq j_{s+1}$, for all $s \in 0, 1, \ldots, l-1$), then the product of the matrices

$$Q_{i_0,j_0} Q^T_{i_1,j_0} Q_{i_1,j_1} Q^T_{i_1,j_1} \cdots Q_{i_{l-1},j_{l-1}} Q^T_{i_0,j_{l-1}} \quad (3)$$

has a fixed column.

Moreover, in the previous section, we have stated that a TPM has a fixed column if and only if there exists an all-zero branch in its tree representation. Therefore, a cycle exists if the tree vector corresponding to (3) contains an all-zero branch. Furthermore, the tree vector can be readily evaluated based on the tree vectors of the component TPMs in (3) using Algorithms 1 and 2.

### B. Code Construction

Recall that a $2^M \times 2^M$ TPM has $2^{2^M-1}$ different combinations. In the design of a TPM-based parity-check matrix $\mathbf{H}_{\rm TPM}$ shown in (2), there will be a tremendous number of possible combinations to consider if $M$ is large. Fortunately, it can be readily shown that if a $\mathbf{H}_{\rm TPM}$ has already achieved a girth of $g$, expanding each '1' in $\mathbf{H}_{\rm TPM}$ with $\tilde{\mathbf{I}}_{2\times2}$ and $\mathbf{I}_{2\times2}$ will result in a new $\mathbf{H}'_{\rm TPM}$ with girth no less than $g$.

Hence, we can start our code construction using small-size TPMs and consider small girth first. Then we expand the TPMs and try to achieve a higher girth. Since the expansion will not lead to new cycles whose length is smaller than the current girth, we do not need to consider the previous portion (i.e., original tree vector) of a TPM when searching for a higher girth, but only need to focus on the expanded part (i.e., new elements in the tree vector that define the expansion). Because of this advantage, hill-climbing algorithms [12, 13] can be easily used in searching for high-girth TPM-LDPC codes. In the hill-climbing algorithms, the process is divided into several steps and in each step, each element is optimized with an adaptive cost. For TPMs, this adaptive cost can be simplified because after we have achieved a girth-$2l$ TPM-LDPC code, we do not need to consider cycles shorter than $2l$ after expanding the code.

In order to reduce the computational intensity when searching for high-girth TPM-LDPC codes using hill-climbing algorithms, we first focus on small-size TPMs. When the size of TPMs is small, the conditions of each sub-matrix we need to consider is relatively small. Then we expand the TPMs to try to achieve a larger girth. For example, the construction of a girth-8 TPM-LDPC code with $3 \times 10$ base matrix is presented as follows. We replace each '1' in the $3 \times 10$ base matrix with a randomly chosen $2^3 \times 2^3$ TPM. Note that there are $2^{2^3-1} = 128$ different $2^3 \times 2^3$ TPMs. Then we minimize the number of cycle-4 by varying the TPMs one-by-one. If all cycle-4s are eliminated and a girth-6 TPM-LDPC code is found, we expand the TPMs to the size of $2^4 \times 2^4$ and attempt to minimize the number of cycle-6. At this step, the initial part of the matrix (or tree vector) is fixed and we only need to consider the expanded part. In addition, all the choices of the expanded part will not lead to new length-4 cycles so the calculation only includes the paths of potential length-6 cycle. Then the conditions to traverse will be reduced distinctly. If all cycle-6s cannot be eliminated after a number of iterations, we expand each TPM again to become $2^5 \times 2^5$ and repeat the operation.

When the size of a TPM becomes large, there will be an enormous number of possible combinations after each expansion. It can be very time-consuming to try all the possible combinations. Therefore, we will randomly pick some of these combinations and select the one with the minimum cost. In this way, the TPM-LDPC code can be further "optimized" with the fast hill-climbing algorithm even when the size is large. With this method, we obtain a girth-8 TPM-LDPC code with a base matrix of size $4 \times 24$, and a girth-10 TPM-LDPC code with a base matrix of size $3 \times 10$.

### C. Parallel Decoding and Message Storage

In [9] it has been shown that during the decoding process, conflicts of RAM access for messages will occur when random-permutation matrices are used to replace the circulant permutation matrices in QC-LDPC codes. In the case of TPM-based LDPC codes, such conflicts can be avoided when the messages are properly stored and the number of parallel processors are chosen with care.

For TPM-LDPC codes, a $P_2^M$ TPM has a size $2^M \times 2^M$. It can also be characterized by $2^{M_1}$ smaller TPMs each of size $2^{M_2} \times 2^{M_2}$ where $M_1 + M_2 = M$. For example, each of the $2^3 \times 2^3$ TPMs shown in Fig. 4 can be characterized by (i)

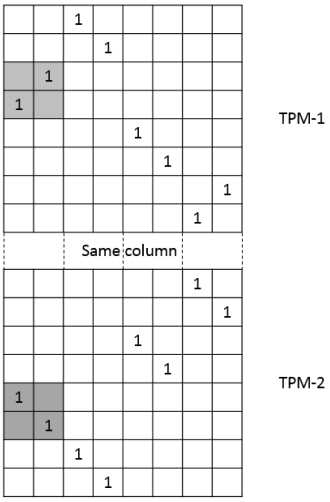


Fig. 4. Illustration of possibility of parallel decoding of TPM-based LDPC code.

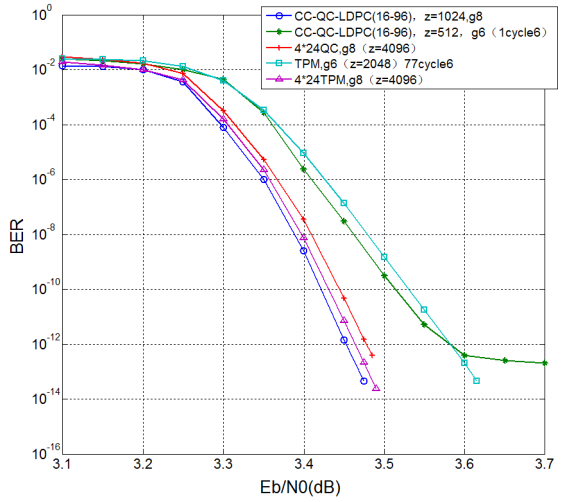


Fig. 5. Bit error rates of different codes.

$2^1(=2)$ smaller $2^2\times 2^2(=4\times 4)$ TPMs, or (ii) $2^2(=4)$ smaller $2^1\times 2^1(=2\times 2)$ TPMs. To avoid conflicts of RAM access, the degree of parallelism $G$ must equal $2^{M_2}$ for some $M_2$. Moreover, the $2^{M_2}$ check-to-variable/variable-to-check ($C\leftrightarrow V$) messages corresponding to each $2^{M_2}\times 2^{M_2}$ small TPM must be assigned to $G=2^{M_2}$ different RAMs. Then, the condition that two $C\leftrightarrow V$ messages in the same RAM are needed at the same time will never happen. From Fig. 4, it can be visualized why parallel decoding can be performed without RAM access conflicts when $G=2$. In fact, it can also be seen that parallel decoding can be performed when $G=4$.

In addition, we need to use a group of RAMs (called *address RAMs*) to store the address information of the RAMs storing $C\leftrightarrow V$ messages. Due to the possibility of parallel decoding, the number and size of such kind of RAMs in a TPM-LDPC decoder are reduced compared with those needed in random-permutation-matrix LDPC decoders [9].

## IV. Simulation Results

We first construct and simulate TPM-LDPC codes with the following parameters.

- $J=4, L=24, M=12$ (sub-matrix size $z\times z=4096\times 4096$), $g=8$, a code rate of $5/6$ and a code length of $98304$
- $J=4, L=24, M=11(z=2048), g=6$, a code rate of $5/6$ and a code length of $49152$

For both cases, we optimize the girth of the LDPC codes using the fast hill-climbing method [13]. The decoder is implemented on an Altera Stratix IV E FPGA. Throughout our simulations, an additive white Gaussian noise (AWGN) channel is assumed. Moreover, 10 belief propagation decoding iterations are used and 4-bit quantization is applied in FPGA decoding. The BER results are plotted in Fig. 5. We also plot the BER of the following codes for comparison.

- Regular $4\times 24$ QC-LDPC codes with $z=4096$, $g=8$ and length $98304$

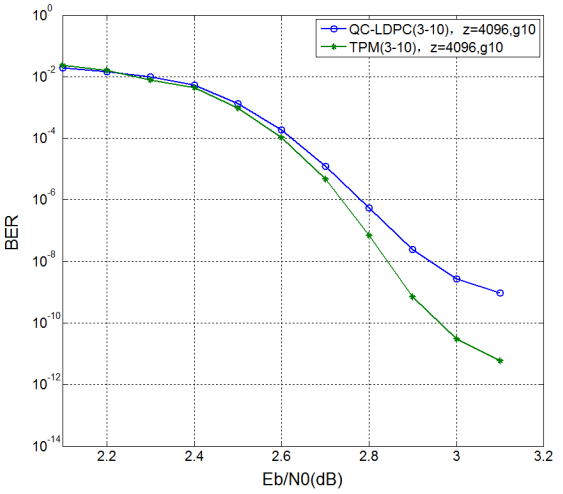


Fig. 6. Bit error rates of QC-LDPC code and TPM-based LDPC code with a $3\times 10$ sub-matrix.

- $16\times 96$ CC-QC-LDPC code with $z=1024$, and $g=8$ and length $98304$
- $16\times 96$ CC-QC-LDPC code with $z=512$, $g=6$ and length $49152$

Comparing codes with length $98304$, the CC-QC-LDPC code accomplishes the best BER, outperforming our proposed TPM-based LDPC code by about 0.015 dB and QC-LDPC code by about 0.03 dB. Comparing codes with length $49152$, the CC-QC-LDPC code outperforms our proposed TPM-based LDPC code by about 0.02 dB. However, the CC-QC-LDPC code reaches an error floor at around $10^{-13}$ whereas our proposed TPM-based LDPC code does not show an error floor below $10^{-13}$.

Next, we construct and simulate a TPM-based LDPC code with the following parameters:$J=3, L=10, z=4096, g=10$, code rate of $7/10$, and a code length 40960. The BER curve is plotted in Fig. 6. We also show the BER of a $3\times 10$

TABLE I
HARDWARE INFORMATION OF THE DECODER IMPLEMENTATIONS.

| Code | A | B | C | D |
|---|---|---|---|---|
| Parallelism degree | 32 | 32 | 4 | 32 |
| ALUTs | 65,178 | 87,479 | 31,660 | 70,342 |
| Registers | 45,336 | 49,322 | 13,564 | 43,801 |
| Memory bits | 1,796,372 | 2,589,891 | 4,297,289 | 2,395,296 |
| Clock | 100 MHz | 100 MHz | 100 MHz | 100 MHz |
| Throughput | 1.55 Gbps | 1.55 Gbps | 0.182 Gbps | 1.55 Gbps |

regular QC-LDPC code with $z = 4096$ and $g = 10$ in the same figure. Both codes cannot achieve a girth of 12 at $z = 4096$ using the fast hill-climbing algorithm. The results show that the proposed TPM-LDPC code not only outperforms the regular QC-LDPC code, but also shows a lower error floor.

In Table I, we show the hardware complexity of the different decoders. The details of the codes are as follows.

- Code A: $4 \times 24$ regular QC-LDPC code, girth=8, $z = 4096$
- Code B: $4 \times 24$ TPM-LDPC code, girth=8, $z = 4096$
- Code C:$16 \times 96$ RP-CC-LDPC code, $z = 1024$
- Code D:$16 \times 96$ CC-QC-LDPC code, $z = 1024$

In general, the complexity becomes lower when fewer address RAMs are used during the decoding. The decoder complexity of the regular QC-LDPC code (Code A) is the lowest. The CC-QC-LDPC decoder (Code D) is more complex than the regular QC-LDPC decoder, but is slightly simpler than the TPM-LDPC decoder (Code B) and much more simpler the random-permutation-based CC-LDPC (RP-CC-LDPC) decoder (Code C). The decoder complexity of the TPM-LDPC codes is lower than that of the RP-CC-LDPC codes. In addition, the degree of parallelism has a great impact on the throughput of the decoder. The TPM-LDPC code has the same throughput as the regular QC-LDPC code and the CC-QC-LDPC code. Their throughputs are much higher than that of the RP-CC-LDPC code.

## V. CONCLUSION

In this paper, a new type of LDPC code called tree-permutation-matrix (TPM)-LDPC code has been proposed. We introduce the construction and properties of TPMs and we define TPM-LDPC codes. We have also described simple methods for computing the product and transpose of TPMs. According to these basic characteristics of TPMs, we have proposed a systematic way of finding large-girth TPM-LDPC codes based on fast hill-climbing algorithm. Using this method, we obtain a girth-8 TPM-LDPC code with a base matrix of size $4 \times 24$ and a girth-10 TPM-LDPC code with a base matrix of size $3 \times 10$. We have shown how RAM access conflicts can be avoided when designing parallel TPM-LDPC decoders.

We have shown our simulation results of three TPM-LDPC codes with different parameters. We have also compared the BER results and decoder complexity of the proposed TPM-LPDC codes with other LDPC codes under the same code length and code rate. Simulation results have shown that TPM-LDPC codes make use of fewer memory resources and have a higher throughput compared with RP-CC-LDPC codes. Compared with QC-LDPC codes, TPM-LDPC codes require more resources in implementation but can provide a slightly better BER performance under a base-matrix size of $4 \times 24$ and an expansion factor of $z = 4096$.

The target application of the proposed TPM-LDPC codes is optical communication. Thus codes with medium code rates and relatively long lengths are simulated in the paper. In the future, we will look into the TPM-LDPC code performance and decoder complexity for (i) code rate of 0.9 (for data storage applications) and 0.5 (for wireless communications) and also (ii) short code lengths of 4K bits and 1K bits. We plan to publish these new results soon.

## REFERENCES


[1] M. P. Fossorier, "Quasi-Cyclic Low-Density Parity-Check Codes from Circulant Permutation Matrices," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1788–1793, 2004.

[2] Y. Dai, N. Chen, and Z. Yan, "Memory Efficient Decoder Architectures for Quasi-Cyclic LDPC Codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 9, pp. 2898–2911, Oct 2008.

[3] C. C. Cheng, J. D. Yang, H. C. Lee, C. H. Yang, and Y. L. Ueng, "A Fully Parallel LDPC Decoder Architecture Using Probabilistic Min-Sum Algorithm for High-Throughput Applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 9, pp. 2738–2746, Sept 2014.

[4] I. Yoo and I. C. Park, "Low-Power LDPC-CC Decoding Architecture Based on the Integration of Memory Banks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 9, pp. 1057–1061, Sept 2017.

[5] W. M. Tam, F. C. M. Lau, and C. K. Tse, "A class of QC-LDPC codes with low encoding complexity and good error performance," *IEEE Communications Letters*, vol. 14, no. 2, pp. 169–171, February 2010.

[6] C.-W. Sham, X. Chen, F. C. M. Lau, Y. Zhao, and W. M. Tam, "A 2.0 Gb/s Throughput Decoder for QC-LDPC Convolutional Codes," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 60, no. 7, pp. 1857–1869, July 2013.

[7] Q. Lu, J. Fan, C.-W. Sham, W. M. Tam, and F. C. M. Lau, "A 3.0 Gb/s Throughput Hardware-Efficient Decoder for Cyclically-Coupled QC-LDPC Codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 1, pp. 134–145, 2016.

[8] S.-Y. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke, "On the Design of Low-Density Parity-Check Codes Within 0.0045 dB of the Shannon Limit," *IEEE Communications letters*, vol. 5, no. 2, pp. 58–60, 2001.

[9] F. C. M. Lau, F. Mo, W. M. Tam, and C. W. Sham, "Random-permutation-matrix-based cyclically-coupled LDPC codes," in *2017 ICACT*, Feb 2017, pp. 497–500.

[10] R. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[11] R. Smarandache, D. G. M. Mitchell, and D. J. Costello, "Partially quasi-cyclic protograph-based LDPC codes," in *2011 IEEE ICC*, June 2011, pp. 1–5.

[12] Y. Wang, J. S. Yedidia, and S. C. Draper, "Construction of high-girth QC-LDPC codes," in *2008 ISTC*, Sept 2008, pp. 180–185.

[13] F. C. M. Lau and W. M. Tam, "A fast searching method for the construction of QC-LDPC codes with large girth," in *2012 IEEE ISCC*, July 2012, pp. 125–128.