AGGREGATED CIRCULANT MATRIX BASED LDPC CODES

Yuming Zhu, Chaitali Chakrabarti

Department of Electrical Engineering

Arizona State University, Tempe, AZ 85287

Email: {yuming,chaitali}@asu.edu

ABSTRACT

This paper presents a variation of circulant matrix based LDPC codes which allows more than one circulant identity matrix in a submatrix of the parity check matrix. The aggregated LDPC supports higher decoding throughput with small increase in datapath complexity. The construction algorithm, bit error rate (BER) performance, information update rule and the architecture for high decoding throughput are also presented.

1. INTRODUCTION

Low-density parity-check (LDPC) codes, which were first introduced by Gallager [1] in 1962, have recently attracted tremendous research interest because of their excellent error correction performance. LDPC codes have been adopted in many standards such as DVB-S2, 10GBase-T, 802.16e (WiMax) and 802.11n. However, designing an LDPC code that has superior performance and can be mapped efficiently into hardware, is still a challenge.

LDPC codes have a large degree of freedom in both code and decoder design. The datapath of the decoder is generally simple, and the operations can be easily parallelized. However, because of the interconnection complexity, the fully parallel LDPC decoder is huge for large block sizes [2]. The partial parallel decoder which makes use of small block matrices with ordered structure is highly preferred. Several LDPC codes with ordered structures based on algebraic constructions have been proposed [3, 4]. These codes make use of algebraic properties that achieve good bit error rate (BER) performance.

Among the partial parallel decoding methods, the layered approach in which the processing is scheduled in the order of block rows [3, 5] is preferable. However, in the parallel processing of block rows, there exists data dependencies between two consecutive block rows if there are non-zero submatrices at the same block column of these two block rows. One way to remove the data dependencies is to reorder the block rows of the parity check matrix [6]. But this kind of reordering become very hard when the code rate is high. This is because for high code rates, even though the number of block rows reduces, a large number of non-zero submatrices has to be kept to maintain the BER performance.

In this paper we derive a variation of the circulant matrix based LDPC codes [4] which allow more than one circulant identity matrix within a submatrix (we will only consider at most two in this paper). We refer to this as the aggregated circulant matrix (ACM) based LDPC code. We show how ACM based LDPC code helps remove the data dependencies between consecutive block rows, and makes it possible to achieve higher throughput. The aggregation algorithm is shown in Section 2. Performance simulation shows that the BER performance of this code under the layered belief propagation (BP) algorithm is better than the non-ACM version. In Section 3, a modified bit update algorithm that supports parallel processing of block rows is presented along with the corresponding architecture. The modified update algorithm can be applied to the decoding of ACM LDPC code for various soft decision algorithms such as BP, min-sum, etc. A decoder for the rate 4/5 code that is pipelined to 3 stages was synthesized using Synopsys Design Compiler with TSMC 90 nm technology. It achieves a throughput of 930 Mbps for 20 decoding iterations. The paper is concluded in Section 4.

This research was supported by an NSF-ITR 0325761.

2. AGGREGATED CIRCULANT MATRIX (ACM) BASED LDPC CODES

2.1. Circulant Matrix based LDPC codes

A fully parallel implementation of the LDPC code is highly complex. For block size of 1024, the fully parallel implementation has utilization of 50% and has 1.75 million gates [2]. Thus partially parallel implementations which use submatrices with ordered structure are preferred [3, 5, 7, 8]. The submatrix is usually a circulant matrix which is a cyclic shifted version of the identity matrix.

In this paper, we follow the notation in [4]. Let the dimension of the block parity check matrix H_b be $m_b \times n_b$; each submatrix is $Z \times Z$, where Z is a prime number. The shifted value for the submatrix (i, j) is $P_{i,j} = b^{i-1}a^{j-1}$ where a and b have multiplicative order of n_b and m_b respectively, in Galois field GF(Z). The block matrix H_b can be made up as $H_b(i, j) = I_{b^{i-1}a^{j-1}}$, where I_x is the circulant identity matrix with shift value of x. For simplicity, we refer to I_x as 'I' in the rest of the paper. We denote the all-zero submatrix by 'O' and the submatrix with two circulant identity matrices aggregated by 'II' (we will show how to determine the shift values in the next subsection).

2.2. ACM based Code Construction Algorithm

In this section, we describe the procedure for constructing ACM based irregular LDPC codes; the procedure can also be applied for regular codes.

Step 1: Determine the dimension of the block parity check matrix (H_b of size $m_b \times n_b$) and determine the degree distribution pair (u, v) such that the bit node degree is in the range $[2, m_b]$. This can be done with the software in [9, 10]. The degree distribution is then quantized such that the bit node density is integer times $\frac{1}{m_b}$, the check node density is integer times $\frac{1}{n_b}$ and the sum of the bit/check node density remains 1. This step is important since the performance of LDPC codes depend (to a large degree) on the variable and check node degree distribution pairs [11].

Step 2: Construct the block parity check matrix with elements of 'I' and 'O'. To reduce the encoding complexity, the resulting block matrix should have an approximately lower triangle structure in the form of $H_b = \begin{bmatrix} A & B & \vdots & T \\ C & D & \vdots & E \end{bmatrix} = \begin{bmatrix} H_{b1} \vdots H_{b2} \end{bmatrix}$, as detailed in [12]. This step is done with an algorithm similar to the progressive edge-growth (PEG) in [13] or block flipping/shifting (RBFS) in [14]. Let Θ be the threshold of bit node degree, i.e., we perform aggregation of the bits with degree no less than Θ. We construct the block matrix as follows:

1. Fill H_{b2} with the lowest degree bit nodes available. The result H_{b2} should be in lower triangle.
2. Fill H_{b1} with the highest degree bit nodes available until all bit nodes been filled. In filling a bit node with degree $\geq \Theta$, first fill the non-zero items pairwise ((i,j) and $(i+\frac{m_b}{2}, j)$), the remaining positions within the column are randomly picked without conflicting the bit and check node degree.

Note that until this point, the algorithm does not require any special steps for the construction of block parity check matrix compared with existing constructions. Thus we can also adopt the intermediate construction from other designs with some modification and proceed with the remaining steps (Step 3 and 4).

Step 3: Perform the "aggregation" on the block matrix obtained from step 2. The non-zero elements in the matrix are filled with 'I', zero elements are filled with 'O'.

1. Choose the 'unprocessed' column with maximum column weight; if there are more than one 'unprocessed' columns with maximum column weight, start from the leftmost one (j). For each $i \leq \lfloor \frac{m_b}{2} \rfloor$, if there exists a column j' such that (i,j), $(i+\frac{m_b}{2}, j)$, (i,j') and $(i+\frac{m_b}{2}, j')$ are all filled with 'I' (the structure is called a cycle-4 'I'-circle), goto 3; otherwise goto 2.
2. If there still exist cycle-4 'I'-circle with two elements in the current column, goto 3; otherwise, mark the current column as 'processed'. If all the columns with degree greater than Θ are marked as 'processed', goto 4; otherwise goto 1.
3. Replace top-left element and bottom-right element of the cycle-4 'I'-circle with II, and the other two elements with O. Goto 1.
4. Exit.

Step 4: The block matrix obtained from step 3 guarantees that there does not exist two 'I' in both (i,j) and $(i+\frac{m_b}{2}, j)$ for $i = 1, \cdots, m_b, j = 1, \cdots, n_b$. In this step we will expand block matrix H_b to the actual parity check matrix H.

- If element (i,j) is 'I', it is replaced with cyclic-shifted identity matrix with shift value equals to $b^{i-1}a^{j-1} (\text{mod } Z)$.
- If element (i,j) is 'II', it is replaced with two cyclic-shifted identity matrix with shift value equals to $b^{i-1}a^{j-1} \pmod p$ and $b^{(i-1\pm\lceil m_b/2 \rceil)}a^{j-1} \pmod p$. The " $\pm$ " sign in the exponent of the second term is determined by the value of i, if $i \leq \lfloor \frac{m_b}{2} \rfloor$, it is " $+$ ", otherwise " $-$ ".
- All the elements 'O' are replaced with all zero matrix.

2.3. Design example of ACM LDPC code construction

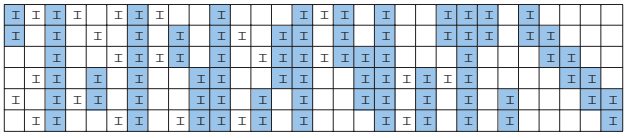
In this section, we outline the construction of an irregular LDPC code with code rate 4/5 in GF(31) for additive white Gaussian noise (AWGN) channel. The study of multiplicative order in GF(31) shows that we can choose $a = 3$, $b = 6$ as the construction element. The parity check matrix then has $n_b = 30$ and $m_b = 6$, and each submatrix is of size 31×31 (which results in a LDPC code with block size of 930). The optimal degree distribution obtained from [10] and our design results are listed in Table 1.

Node Degree	Optimal Distribution	Design Result	
Bit Node	Optimal Bit Dist.	Quant. Dist.	Node #
2	0.42348914827824	0.4	12
3	0.36868120674005	0.4	12
6	0.20782964498171	0.2	6
Check Node	Optimal Check Dist.	Quant. Dist.	Node #
16	1	1	6

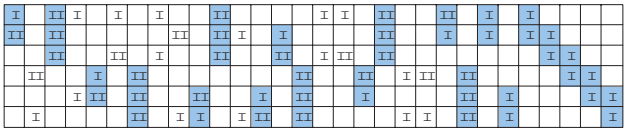
Table 1. Bit/Check node degree design result

From the above results, we construct the block matrix shown in Fig. 1(a). The aggregated version is shown in Fig. 1(b) for threshold $\Theta = 3$. Here "I" represents a circulant matrix (the shift values have not been listed for easier reading), "II" represents two circulant matrices aggregated in one block, and all the empty blocks are zero matrixes.

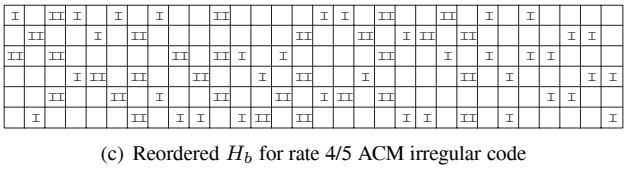
The shaded submatrices in Fig. 1(a) are the submatrices that contain data dependencies under iterative layered decoding. Clearly, some of the dependencies can be removed from block row reordering. Others, such as those in block columns with 6 non-zero entries can not be removed. In contrast, the data dependencies in the aggregated H_b in Fig. 1(b) are all removable. Fig. 1(c) shows how all the data dependencies in 1(b) have been removed with block row reordering in decoding.



(a) H_b for rate 4/5 irregular code before aggregation



(b) H_b for rate 4/5 irregular code after aggregation



(c) Reordered H_b for rate 4/5 ACM irregular code

Fig. 1. An example of aggregation

2.4. Performance of ACM LDPC codes

Fig. 2 shows the BER performance of the irregular 4/5 LDPC code presented in Fig. 1. In the simulation, we assume that the encoded data are modulated with BPSK and transmitted through AWGN channel. The plots shows the resulting BER under 20 iterations of BP decoding. We can see that the ACM LDPC outperform the codes without ACM with around 0.1 dB coding gain at BER=10^{-5}.

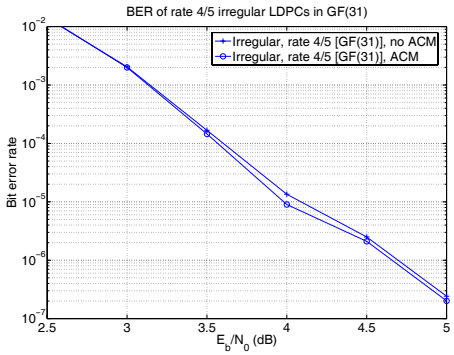


Fig. 2. BER performance of the rate 4/5 irregular LDPC

3. DECODER DESIGN FOR ACM LDPC CODES

3.1. Decoding of LDPC codes

LDPC decoding is done by either the bit flipping hard-decision algorithm [1], or by the soft-decision iterative decoding algorithms such as the belief propagation (BP) algorithm, the min-sum algorithm, the λ-min algorithm [15], etc. All of them involve two kinds of operations: variable (bit) node processing (VP) and check node processing (CP). The operations are summarized below:

$$Initialization: \quad E_{n,m} = 0, L_n = I_n \tag{1}$$

$$VP: \quad L_{n,m} = L_n - E_{n,m} \tag{2}$$

$$CP: \quad E^{New}_{n,m} = f\left(L_{n',m} \,\middle|_{n' \in S \subseteq N(m)}\right) \tag{3}$$

$$Bitupdate: \quad L^{New}_n = L_{n,m} + E^{New}_{n,m} \tag{4}$$

where I_n is the intrinsic information from the received signal, $E^i_{n,m}$ is the extrinsic information from check node m to variable node n, $L_{n,m}$ is the information from variable node n to check node m. $N(m)$ is the set of variable node which is connected with node m in the bipartite graph. Similarly, $M(n)$ is the set of check node which is connected with node n. The different decoding algorithms differ in how the function $f(\cdot)$ is evaluated. The set S is a subset of $N(m)$ when λ-min algorithm is used (S contains the λ items with smallest magnitude), $S = N(m) \setminus \{n\}$ for the BP algorithms. The $f(\cdot)$ function used in BP algorithm is

$$E^{New}_{n,m} = -\prod_{n' \in N(m) \setminus \{n\}} sign(L_{n',m}) \cdot \Psi\Big(\sum_{n' \in N(m) \setminus \{n\}} \Psi(L_{n',m})\Big) \tag{5}$$

3.2. Modified Bit Update algorithm for parallel decoding of ACM LDPC codes

The algorithms for the normal circulant matrix based LDPC codes work fine for the ACM LDPC codes if the rows within a block row are decoded serially or with small degree of parallelism. But for very high throughput applications, such as in DVB-S2 and WiMax, a large parallel factor is necessary. If the rows within a block row have to be processed in parallel to increase the throughput, there arises a problem. For the small block matrixes with two circulant matrix aggregated, there are two set of L^{New}_n values generated simultaneously and only one set should be stored.

For a specific n, let m_1 and m_2 represent the two check nodes connected with n in the current block row. In this block row, E^{New}_{n,m_1} and E^{New}_{n,m_2} are the actual extrinsic information values for n. We rewrite the equations with notation

$$L^{New}_n = S_{(n,m_1,m_2)} + E^{New}_{n,m_1} + E^{New}_{n,m_2} \tag{6}$$

where $S_{(n,m_1,m_2)} = I_n + \sum_{m' \in M(n) \setminus \{m_1,m_2\}} E_{n,m'}$ is the unchanged part between L_n and L^{New}_n in this block row. We define a new variable

$$T_{n,m} \triangleq \begin{cases} L_{n,m} - E_{n,m} + 2 \times E^{New}_{n,m} & ,(m,n) \in II \\ L_{n,m} + E^{New}_{n,m} & ,(m,n) \in I \end{cases} \tag{7}$$

L^{New}_n is now written as

$$L^{New}_n = \begin{cases} \frac{T_{n,m_1} + T_{n,m_2}}{2} & ,(m_1,n),(m_2,n) \in II \\ T_{n,m} & ,(m,n) \in I \end{cases} \tag{8}$$

The result L^{New}_n is identical with the original algorithm as we will show in the following.

$$\begin{aligned} L^{New}_n &= \frac{L_{n,m_1} - E_{n,m_1} + L_{n,m_2} - E_{n,m_2}}{2} + E^{New}_{n,m_1} + E^{New}_{n,m_2} \\ &= \frac{S_{(n,m_1,m_2)} + E_{n,m_2} - E_{n,m_1}}{2} \\ &\quad + \frac{S_{(n,m_1,m_2)} + E_{n,m_1} - E_{n,m_2}}{2} + E^{New}_{n,m_1} + E^{New}_{n,m_2} \\ &= S_{(n,m_1,m_2)} + E^{New}_{n,m_1} + E^{New}_{n,m_2} \end{aligned} \tag{9}$$

We can see that the modified bit update algorithm doesn't affect $f(\cdot)$ at all. It can be applied to all the decoding algorithms (BP, min-sum and λ-min) to improve the throughput.

3.3. Decoder Architecture for ACM LDPC codes

Fig. 3 shows the top level architecture of the ACM LDPC decoder.

There are two set of memories in the decoder: extrinsic memory which stores the extrinsic information ($E_{n,m}$ for the current iteration) and variable node memory which stores the L_n for the current iteration. Variable node unit (VNU) is a pool of variable processing (VP) units; each VP is a parallel adder which compute $L_n - E_{n,m}$. Check node unit (CNU) is a pool of check processing (CP) units which perform the function described in eqn. (3). The bit update combiner performs the function described in eqn. (7). Because the memory organization in variable node memory is column based, and the scheduling of the decoding within an iteration is check node based, the shuffling network and the inverse shuffling network are required to take care of the spacing and time shuffling. To further increase the throughput, a bypass MUX is added to the input of the shuffling network. If the L^{New}_n data are used immediately after the storage, we can directly send them to the shuffling network by the bypass MUX and reduce one clock cycle in the loop.

The throughput of the proposed ACM LDPC decoder is doubled compared to the non-ACM counterpart when there exist data

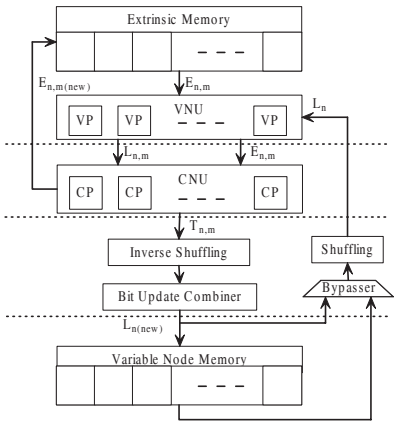


Fig. 3. LDPC decoder architecture

dependencies between consecutive block rows. The throughput of the decoder under layered decoding is: (CodeRate* ParallelFactor* ClockRate) /[(1-CodeRate) * CyclePerBlockRow*Iter]. The synthesis result of the proposed decoder in TSMC 0.90 nm technology shows that the decoder with 3 pipeline stages works with a clock rate=300MHz. The number of cycles per block row is 2 (the non-ACM counterpart requires 4). So the actual throughput of the decoder for 20 iterations is 930 Mb/s. However, the pure throughput value is not a good metric in evaluating the architecture, because the value varies a lot based on the choice of code rate, parallel factor and number of iterations. Instead the ClockRate/CyclePerBlockRow ratio (CCR) is a better metric. The proposed decoder architecture for ACM has a CCR of 150 MHz.

On the flip side, the parallel decoding of ACM LDPC codes results in higher datapath complexity. More specifically we need one more addition because of Eqn. (7) and one addition and one 1-bit right shift operation (whose complexity can be ignored) as shown in Eqn. (8). Table 2 shows the logic complexity in datapath of the parallel decoder. While the complexity of the datapath increased by 20%, the achievable throughput is doubled. In any case, the data path has been demonstrated to occupy a small percentage of the overall decoder and thus this penalty is compensated for.

Arch	16-Adder/Xor Tree	Adder	LUT	MUX
Non-ACM	31	1488	992	-
ACM	31	1860	992	186

Table 2. Data path complexity

Recently, we became aware of the work on geometric LDPC construction [16] which also builds the submatrices with multiple circulant matrices. Our work differs from [16] in that we perform aggregation based on algebraic construction, and that we propose an explicit construction algorithm to achieve high throughput. In addition, the update rule that we proposed is also applicable to any parallel decoding including the code in [16]. The method in [17] uses the construction in [16] but constraints the shift values in the aggregated submatrix. It requires extra buffers and/or partitioned memory banks as well as complex control logic. In comparison, our approach achieves high throughput with only a slight increase in the complexity of the datapath.

4. CONCLUSION

In this paper we presented ACM LDPC codes which allows more than one circulant identity matrix in one submatrix. The software simulation shows that the ACM LDPC provides comparable or even better BER performance compared with its non-ACM counterpart. The construction algorithm, information update rule and decoder architecture which supports the parallel decoding of these codes for high throughput applications are also presented.

5. REFERENCES

[1] R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Inform. Theory*, vol. IT-8, no. 1, pp. 21–28, Jan. 1962.

[2] C. Howland; A. Blanksby, "A 220 mW 1 Gb/s 1024-bit rate-1/2 low density parity check code decoder," in *IEEE Conf. on Custom Integrated Circuits (CICC)*, May 2001, pp. 293–296.

[3] M.M. Mansour; N.R. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. VLSI*, vol. 11, no. 6, pp. 976–996, 2003.

[4] R.M. Tanner; D. Sridhara; A. Sridharan; T.E. Fuja; D.J Costello, "LDPC Block and Convolutional Codes Based on Circulant Matrices," *IEEE Trans. Inform. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

[5] D.E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *IEEE Workshop on Signal Processing Systems (SIPS)*, 2004, pp. 107–112.

[6] "IEEE 802.16e Draft 7 (P802.16e/D7)," available at: http://www.ieee802.org/16/tge/schedule.html, April 2005.

[7] Tong Zhang; K.K. Parhi, "VLSI implementation-oriented (3, k)-regular low-density parity-check codes," in *IEEE Workshop on Signal Processing Systems (SIPS)*, 2001.

[8] E. Liao; E. Yeo; B. Nikolic, "Low-Density Parity-Check Code Construction for Hardware Implementation," in *Proc. of IEEE Int. Conf. on Commun. (ICC)*, 2004.

[9] S.Y. Chung, "Irregular LDPC design program," Available at: http://lids.mit.edu/ sychung/gaopt.html.

[10] R. Urbanke, "LDPC optimization program," Available at: http://lthcwww.epfl.ch/research/ldpcopt/.

[11] T.J. Richardson; M.A. Shokrollahi; R.L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. on Inform. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[12] T.J. Richardson; R.L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. on Inform. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[13] X.Y Hu; E. Eleftheriou; D.M. Arnold, "Progressive edge-growth Tanner graphs," in *IEEE Global Telecom. Conf. (GLOBECOM)*, Nov. 2001, vol. 2, pp. 995–1001.

[14] H. Zhong; T. Zhang, "Block-LDPC: A Practical LDPC Coding System Design Approach," *IEEE Trans. Circuits and Systems I*, vol. 52, no. 4, pp. 766–775, April 2005.

[15] F. Guilloud; E. Boutillon; J.-L. Danger, "λ-Min Decoding Algorithm of Regular and Irregular LDPC Codes," in *3rd Int. Symp. on Turbo Codes & related topics*, Sept. 2003.

[16] L. Chen; J. Xu; I. Djurdjevic; S. Lin, "Near-Shannon-limit quasi-cyclic low-density parity-check codes," *IEEE Trans. on Commun.*, vol. 52, no. 7, pp. 1038–1042, April 2004.

[17] Z. Wang; Q. Jia;, "Low Complexity, High Speed Decoder Architecture for Quasi-Cyclic LDPC Codes," in *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, May 2005, pp. 5786 – 5789.