Design of VLSI Implementation-Oriented LDPC Codes

Hao Zhong and Tong Zhang
Electrical, Computer and Systems Engineering Department
Rensselaer Polytechnic Institute, Troy, NY, USA
zhongha@rpi.edu, tzhang@ecse.rpi.edu

***Abstract*— Recently, low-density parity-check (LDPC) codes have attracted much attention because of their excellent error-correcting performance and highly parallelizable decoding scheme. However, the effective VLSI implementation of an LDPC decoder remains a big challenge and is a crucial issue in determining how well we can exploit the benefits of the LDPC codes in the real applications. In this paper, following the joint code and decoder design philosophy, we propose a semi-random design scheme to construct the LDPC codes that not only exhibit very good error-correcting performance but also effectively fit to partially parallel VLSI decoder implementations. The corresponding partially parallel decoder has a very regular structure and simple control mechanism. Our computer simulations show that such LDPC codes achieve very good performance comparable to their counterparts constructed in a fully random scheme, which however have little chance of fitting to partially parallel decoder implementations.**

I. Introduction

Low-density parity-check (LDPC) codes have become a topic of great current interest because of their excellent error-correcting performance and highly parallelizable decoding scheme. The past a few years experienced significant improvement on the design and analysis of the LDPC codes with near Shannon-limit performance. However, the realization of an LDPC decoder VLSI implementation still remains a big challenge and is a crucial issue determining how well we can exploit the unmatched merits of the LDPC codes in the real application.

Most LDPC codes are constructed by first choosing the required code block length and the node degree distributions, then pseudo-randomly generating a parity-check matrix (or the corresponding bipartite graph) under certain constraints. Gallager [1] constructed the (j, k)-regular LDPC code ensemble based on the (j, k)-regular matrices with rows divided into j submatrices, the first composed by k copies of the identity matrix and with subsequent submatrices being random column permutations of the first. Luby et al. [2] proposed LDPC codes with irregular node degree distributions which outperform regular ones. They introduced tools based on linear programming for designing irregular code ensembles for which the maximally allowed crossover probability of the binary symmetric channel is optimized. Richardson and Urbanke [3] extended the work of Luby et al. to any binary input memoryless channel and to soft decision message passing decoding. They determined the capacity of message passing decoders applied to LDPC code ensembles by a method called density evolution. Recently, Kou and Lin [4] developed a finite geometric approach to the construction of LDPC codes which can lead to very good iterative decoding performances in moderate block length.

Unfortunately, all these code construction methods gave little consideration on efficient decoder VLSI implementation. The conventional *code-to-decoder* design, *first construct a code with good error-correcting performance, then develop the decoder VLSI implementation for that specific code*, makes the efficient LDPC decoder VLSI implementation a nearly impossible task. In this paper, we propose a design scheme to construct the LDPC codes that not only exhibit very good error-correcting performance but also effectively fit to efficient partially parallel decoder VLSI implementations. The underlying philosophy, originally proposed in [5] [6], is to jointly consider the code design and decoder implementation. When we prepared the final version of this paper, we found that the proposed code construction strategy is very similar to a recently proposed IIP code construction scheme [7] in the matrix permutation and expansion, which can date back to SFT approach in [8].

The remainder of this paper is organized as follows. We briefly discuss some LDPC decoder implementation issues in Section II. In Section III, we present our proposed method to construct partially parallel decoder implementation oriented LDPC codes. Section IV presents the corresponding partially parallel decoder structure. The simulation results are shown in Section V to illustrate the error-correcting performance of such LDPC codes. Section VI provides some concluding remarks.

II. LDPC Decoder Implementation Issues

The main challenge in the LDPC code decoder VLSI implementation is how to effectively manage the message passing during the iterative belief propagation (BP) decoding. Generally, the LDPC decoder implementations fall into two categories:

- *Fully parallel decoders* that directly instantiate the bipartite graph of the LDPC code to the hardware.
- *Partially parallel decoders* that maps a certain number of variable nodes or check nodes to a single hardware unit in time-division multiplexing mode.

In a fully parallel decoder, each individual variable node or check node is physically implemented as a node decoding unit, and all the units are connected through an interconnection network reflecting the bipartite graph connectivity. It is clear that such fully parallel decoders can achieve very high decoding throughput, e.g., Howland and Blanksby [9] [10] have implemented a 1Gbps decoder for 1024-bit, rate 1/2 LDPC code. However, the fully parallel decoders suffers from high

implementation complexity, especially the prohibitive routing wire overhead with too many global long routing wires. Moreover, the random-like connection between the variable nodes and check nodes may cause routing congestion. Hence, the fully parallel decoder is only a feasible option for LDPC codes with short code length, i.e., few thousand bits.

Partially parallel LDPC decoder [6] [11] [12] targets on appropriate trade-offs between hardware complexity and decoding speed. By using time-division multiplex mapping in which a certain number of variable nodes or check nodes are mapped to a single decoding unit, partially parallel decoder trades the decoding speed for reduced wire and logic implementation complexity. In this work, we are interested in the LDPC codes suitable for partially parallel decoder implementations.

As mentioned in the above, the construction of the LDPC codes typically relies on random construction that results in *unstructured* bipartite graph topology. However, the partially parallel decoder implementation demands relatively *structured* LDPC code bipartite graph topology that could lead to the efficient realization of partially parallel message passing. Thus the partially parallel decoder implementation is not trivial. In this paper, we propose an approach to construct a class of LDPC codes that have relatively structured bipartite graph topologies that lead to very good performance and directly fit to partially parallel decoder implementations.

III. LDPC Code Design

In this section we present our proposed LDPC code construction approach. To construct an LDPC code with an $M \times N$ parity check matrix, provided that $M = p \cdot M_s$ and $N = p \cdot N_s$, the proposed construction approach performs the following three steps:

1) Use a heuristic algorithm to construct a group of *base* parity check matrices with the size of $M_s \times N_s$.
2) Randomly *expand* each base matrix into a $p \cdot M_s \times p \cdot N_s$ parity check matrix.
3) Apply a *cycle effect* criterion to select an LDPC code from the developed code group.

A. Base Matrices Construction

Generally speaking, there are two factors that mainly determine the error-correcting performance of an LDPC code: the effectiveness of the iterative belief propagation algorithm and the minimum distance of the LDPC code. While the LDPC code has a reasonably large code length (larger than 1K), its minimum distance *almost for sure* will be sufficiently good. So the error-correcting performance is mainly determined by the effectiveness of the iterative belief propagation algorithm. Therefore, we only need to concentrate on how to construct an LDPC code which can be effectively decoded under the iterative BP algorithm.

It is well known that when the bipartite graph of the LDPC code is cycle-free, the belief propagation algorithm becomes a recursive algorithm that always converges to the true maximizing a posteriori (MAP) algorithm after a finite number of messages have been passed. However, all the good codes inevitably have cycles in their bipartite graphs. Fortunately, the BP algorithm performs remarkably well when the bipartite graph does not contain too many small cycles.

Hence, in the construction of the base matrices, we try to avoid small cycles as much as possible. We adopt the heuristic approach called bit-filling proposed by Campello and Modha [13] to construct a certain number of $M_s \times N_s$ base matrices whose corresponding bipartite graphs have large girths[1]. The basic idea behind this strategy is to insert 1's into the corresponding matrix one by one while maintaining the prescribed girth constraint. The construction starts with insisting on a large girth constraint, until it cannot add more bits without violating the girth constraint. Then the girth constraint is decreased and the process continues. The algorithm terminates when all the needed bits (edges) are inserted into the matrix (graph) or the girth constraint falls below a specified minimum value. To make the algorithm applicable to an LDPC code with an arbitrary weight (or node degree) distribution, we redefine the problem and revise the algorithm as follows.

Problem: *Given the weight distribution of the rows and columns, we would like to construct an $m \times n$ parity check matrix H with large girth g(H), where $g(H) \geq \underline{g}$, $\underline{g}$ is the minimum allowable girth value.*

Algorithm 3.1: Bit-filling Algorithm

1) *According to the weight distribution of the columns, generate a sequence $\{a_1, a_2, ..., a_n\}$, where a_j $(1 \leq j \leq n)$ denotes the number of 1's in j-th column of H.*
2) *According to weight distribution of rows, generate a sequence $\{b_1, b_2, ..., b_L\}$, where b_i $(1 \leq i \leq L)$ is the row position of an 1 in H, and L is the total number of ones in H.*
3) *Set $H = \mathbf{O}$ and $g' = \overline{g}$, where $\overline{g}$ is the initial girth constraint.*
4) *For each column j, insert 1's at $H(j, b_i)$ until the weight of column j is equal to a_j. For each insertion of 1, the constraint $g(H) < g'$ should be maintained; b_i is randomly picked and removed from $\{b_1, b_2, ..., b_L\}$ after a successful insertion at $H(j, b_i)$.*
5) *If step 4 can not proceed without violating the constraint, reduce g' by two and go back to step 4.*
6) *Stop when L ones are inserted in H, or $g' < \underline{g}$.*

B. Matrices Expansion

As we mentioned in Section II, our target is to construct LDPC codes that have relatively structured bipartite graph topology that lead to very good performance and directly fit to partially parallel decoder implementations. The following *matrix expansion* plays a key step in reaching this target.

For each $M_s \times N_s$ base matrix generated by the above algorithm, we randomly expand it by a factor p to obtain a $p \cdot M_s \times p \cdot N_s$ matrix, as illustrated in Figure 1. Each 0 in the base matrix is expanded to a $p \times p$ zero sub-matrix $\mathbf{O}$ and each

[1]Girth is the length of the minimum cycle in a graph

1 at the position (u, v) is expanded to a $p \times p$ sub-matrix $\mathbf{T}_{u,v}$ that is obtained by right cyclic shifting a $p \times p$ identity matrix by $k_{u,v}$ columns. Each integer $k_{u,v}$ is generated randomly.

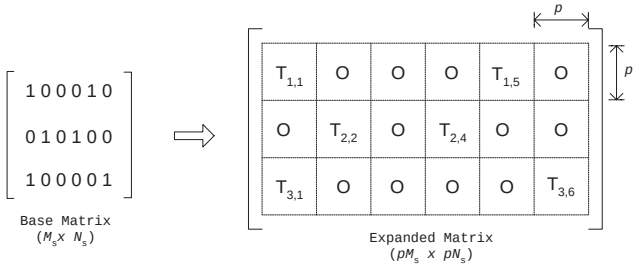

Fig. 1. Matrix expansion.

Notice that the above matrix expansion scheme is beneficial to both error-correcting performance and decoder hardware implementations. The girth of the expanded matrix is always larger than or equal to the girth of the base matrix that has been optimized with its girth. The random cyclic permutation of each identity matrix further increases the value of the average cycle length, which will improve the effectiveness of BP algorithm. As we will see later, such expansion will directly lead to a partially parallel decoder consisting of a structured array of memory and processors. Moreover, the random cyclic permutation of each identity matrix results in very simple control logic for generating the memory access address.

C. Code Selection

The last step is to select the code from the code ensemble constructed above leading to very good error-correcting performance. It is well known that the cycles, particularly short cycles, in the bipartite graph degrade the effectiveness of the iterative BP decoding algorithm and hence the error-correcting capability of the LDPC codes. Therefore, we select the code based on the metric called *cycle effect*, which is also known as loopiness [14]. The cycle effect is defined as:

$$L = \sum_{i=4,6,8,\ldots} N_i \cdot \alpha^i,$$

where N_i is the number of cycles with length i and α is a value chosen for the sum to converge. We may intuitively justify the effectiveness of this selection criterion based on the well-known fact that the LDPC iterative belief propagation decoding algorithm works well if the code graph does not contain too many short cycles.

IV. Partially Parallel Decoder Structure

Exploiting the characteristics of the LDPC codes constructed in the above, we develop a partially parallel decoder structure as shown in Figure 2. It consists of an array of node computation units to perform all the node computation in time-division multiplexing mode and an array of memory blocks to store all the decoding message. The message passing that reflects the bipartite graph connectivity is jointly realized by the memory address generation and the interconnection among memory blocks and node computation units. Suppose the base matrix is $M_s \times N_s$ and contains L 1's, and the expansion factor is p. The expanded matrix contains L permutated identity matrices, each one denoted as $T_{u,v}$ as illustrated in Figure 1. We can show that the LDPC code defined by such an expanded matrix exactly fits to the partially parallel decoder as shown in Figure 2. This partially parallel decoder contains M_s check node computation units (CNUs), N_s variable node computation units (VNUs), and $L+N_s$ memory blocks among which L blocks store the iterative decoding messages, each one denoted as DMEM$_{u,v}$, and N_s blocks stores the channel messages, each one denoted as CMEM$_v$.

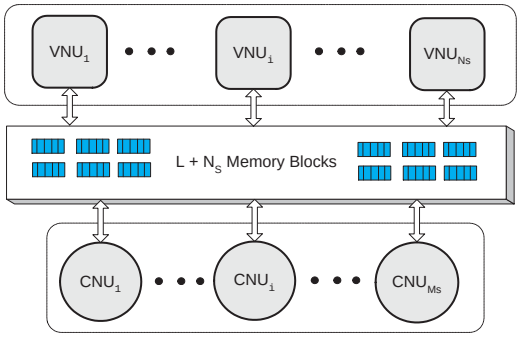

Fig. 2. Partially parallel decoder structure.

Each DMEM$_{u,v}$ connects with CNU$_u$ and VNU$_v$, and stores p decoding messages associated with the p 1's in the permutated matrix $T_{u,v}$. This decoder completes each decoding iteration in $2 \cdot p$ clock cycles. It works in check node processing mode during the 1st p clock cycles, and variable node processing mode during the 2nd p clock cycles. The operations in the two modes are as follows:

- Check Node Processing: CNUs compute check-to-variable messages for all the check nodes in a time division multiplexing fashion. All the DMEMs store the variable-to-check messages at the beginning. In each clock cycle, one variable-to-check message in each DMEM is *converted* to the corresponding check-to-variable message by a *read-computation-write* process. The memory access address of each DMEM$_{u,v}$ is generated by a counter that starts from the block permutation value $k_{u,v}$.
- Variable Node Processing: VNUs calculate extrinsic variable-to-check messages and update the decoding decision of all the variable nodes in a time division multiplexing fashion. All the DMEMs store the check-to-variable messages at the beginning. Similarly, in each clock cycle, one check-to-variable message in each DMEM is converted to a variable-to-check message and the decoding decision of the corresponding is updated. The memory access addresses of all the DMEMs and CMEMs are generated by a counter that starts from 0.

Clearly, the number of node decoding units in this partially parallel decoder is reduced by the expansion factor p compared with its fully parallel counterpart. This partially parallel decoder is well suited for efficient high speed hardware implementation because of the regular structure and simple control logic. Compared with previous work [5] [6] [12], our proposed design scheme supports much more flexible code rate configurations and degree distributions, hence has great potential on achieving very good error-correcting performance.

V. Simulation Results

Applying our proposed design scheme, we construct two $(3, 6)$-regular LDPC codes, denoted as C_1 and C_2, to demonstrate the error-correcting performance. The code lengths of C_1 and C_2 are 4K and 8K, respectively. They have the same expansion factor $p = 64$. The base matrices of C_1 and C_2 are 32×64 and 64×128, respectively. For each code, we randomly generated 500 base matrices optimized with the girth, then randomly expanded each base matrix to one expanded matrix. Finally we selected the one leading to the best cycle effect as the selected parity check matrix. In the simulation, we assumed that both codes were modulated by BPSK and transmitted over AWGN channel.

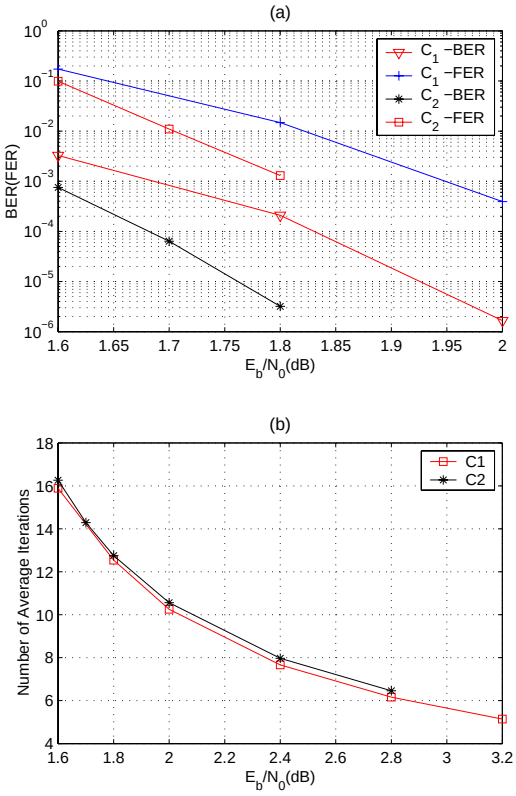

Fig. 3. Simulation results of (a) BER vs. SNR, and (b) average number of decoding iterations vs. SNR, where the maximum iteration number is 20 and each point is obtained with the simulation up to at least 100 frame errors occur.

Figure 3 shows the simulated bit error rate (BER), frame error rate (FER), and the average number of decoding iterations. We note that these codes achieve comparable error-correcting performance to their counterparts constructed in fully random scheme. However the fully randomly constructed codes have little chance of fitting to efficient partially parallel decoder implementations.

VI. Conclusion

In this paper, we present an approach to construct the LDPC codes that not only have very good error-correcting performance but also fit to efficient partially parallel decoder hardware implementations. The code construction scheme and the corresponding partially parallel decoder design have been described in details. The main advantage of this proposed approach that it could realize flexible trade-off between hardware complexity and decoding speed and support any arbitrary node distribution configuration, hence shows great potential on achieving very good error-correcting performance.

Future work is directed to the investigation of appropriate irregular node degree distribution suited to this proposed code design scheme and how to reduce the encoding complexity by exploring the regularity of the code structure.

References

[1] R. G. Gallager, *Low-Density Parity-Check Codes*, M.I.T Press, 1963.

[2] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Transactions on Information Theory*, vol. 47, pp. 585–598, Feb. 2001.

[3] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, pp. 599–618, Feb. 2001.

[4] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Transactions on Information Theory*, vol. 47, pp. 2711–2736, Nov. 2001.

[5] E. Boutillon, J. Castura, and F. R. Kschischang, "Decoder-first code design," in *Proceedings of the 2nd International Symposium on Turbo Codes and Related Topics*, Brest, France, Sept. 2000. available at http://lester.univ-ubs.fr:8080/~boutillon/publications.html, pp. 459–462.

[6] T. Zhang and K. K. Parhi, "VLSI implementation-oriented $(3, k)$-regular low-density parity-check codes," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Sept. 2001. available at http://www.ecse.rpi.edu/homepages/tzhang/, pp. 25–36.

[7] D. E. Hocevar, "Ldpc code construction with flexible hardware implementation," in *IEEE International Conference on Communications*, 2003, pp. 2708 –2712.

[8] T. Fuja D. Sridhara and R.M. Tanner, "Low density parity check codes from permutation matrices," in *Conf. On Info. Sciences and Sys.*, The John Hopkins University, March 2001.

[9] C. Howland and A. Blanksby, "Parallel decoding architectures for low density parity check codes," in *Proc. of 2001 IEEE Int. Symp. on Circuits and Systems*, Sydney, May 2001.

[10] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, March 2002.

[11] E. Yeo, P. Pkzad, N. Nikolic, and V. Anantharam, "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. on Magnetics*, vol. 37, no. 2, pp. 748–755, March 2001.

[12] M. M. Mansour and N. R. Shanbhag, "Low power VLSI decoder architectures for LDPC codes," in *2002 International Low Power Electronics and Design*, 2002, pp. 284–289.

[13] J. Campello and D. S. Modha, "Extended bit-filling and ldpc code design," in *IEEE Global Telecommunications Conference*, 2001, pp. 985–989.

[14] J. Thorpe, "Design of ldpc graphs for hardware implementation," in *IEEE International Symposium on Information Theory*, 2002, pp. 483–483.