# ARCHITECTURE-AWARE LOW-DENSITY PARITY-CHECK CODES

*Mohammad M. Mansour and Naresh Shanbhag*

Coordinated Science Laboratory, ECE Department
University of Illinois at Urbana-Champaign, Urbana, IL
mmansour@uiuc.edu, shanbhag@uiuc.edu

## ABSTRACT

**A high-throughput memory-efficient decoder architecture for architecture-aware low-density parity-check (LDPC) codes is proposed based on a novel turbo-decoding algorithm. The architecture benefits from various optimizations at the code-design, decoding algorithm, and decoder architecture levels. The interconnect complexity and memory overhead problems of current decoder implementations are reduced by designing structured or architecture-aware LDPC codes and employing a new turbo-decoding algorithm. An efficient memory architecture coupled with a scalable and dynamic transport network for storing and routing messages are proposed. Simulations demonstrate that the proposed architecture attains a throughput of 1.92 Gbits/s for a frame length of 2304 bits, and achieves savings of 89.13 % and 62.80 % in power consumption and silicon area over state-of-the-art, with a reduction of 60.5 % in interconnect wires.**

## 1. INTRODUCTION

The phenomenal success of turbo codes [1] powered by the concept of iterative decoding via message-passing has rekindled the interest in low-density parity-check (LDPC) codes first discovered by Gallager in 1961 [2]. Recent breakthroughs were achieved with the introduction of irregular LDPC codes in [3] putting LDPC codes on a par with turbo codes. However, efficient hardware implementation techniques of turbo decoders have given turbo codes a clear advantage over LDPC codes allowing them to occupy mainstream applications ranging from wireless applications to fiber-optics communications. Hence, the quest for more efficient LDPC decoder implementation techniques has become a topic of increasing interest, gradually promoting LDPC codes as serious competitors to turbo codes on both fronts.

The design of LDPC decoder architectures differs from the decoder design for other classes of codes, in particular turbo codes, in that it is intimately related to the structure of the code itself through its parity-check matrix [4]. An LDPC code is defined by a random sparse parity-check matrix $\mathbf{H} = [h_{ij}]_{m \times n}$ and is described by a bipartite graph (see Fig. 1) whose *reduced* adjacency matrix is $\mathbf{H}$. The graph has on one side $m$ *check* nodes $\{c_1, c_2, \cdots, c_m\}$ corresponding to the $m$ rows of $\mathbf{H}$, and on the other $n$ *bit*-nodes $\{b_1, b_2, \cdots, b_n\}$ corresponding to the $n$ columns of $\mathbf{H}$. A bit-node $b_j$ is connected to a check-node $c_i$ if $h_{ij} = 1$. In a *regular* $(c, r)$-LDPC code, bit-nodes have degree $c$ and check-nodes have degree $r$ in the graph, while in an *irregular* $(C, R)$-LDPC code, the bit- and check-node degrees are drawn from the sets $R$ and $C$, respectively.

LDPC codes are decoded iteratively using Gallager's two-phase message-passing (TPMP) algorithm [2] which computes iteratively extrinsic probability values associated with each bit-node using disjoint parity-check equations that the bit participates in [2]. Each iteration consists of two phases of computations in which updates of all bit-nodes are done in **phase 1** by sending messages to neighboring check-nodes, and then updates of all check-nodes are done in **phase 2** by sending messages to neighboring bit-nodes. Updates in each phase are independent and can be parallelized.

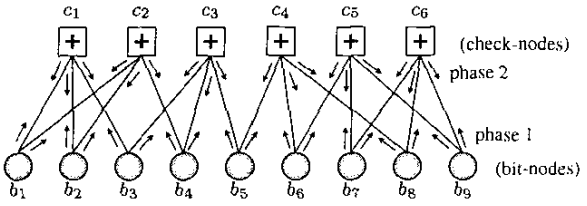



Figure 1: Bipartite graph of a regular (2,3)-LDPC code.

Unlike the case for turbo codes, the randomness in communicating messages on the bipartite graph in LDPC codes results in complex interconnects for large $n$ in a parallel decoder implementation, and stringent memory requirements in a serial decoder implementation limiting the applicability of LDPC codes in latency and power sensitive applications. All current decoder architectures ( [5, 6]; see references in [7]) address the interconnect and memory problems by applying optimizations at one system abstraction level, and either retain the disadvantages of or introduce further disadvantages at other levels.

This paper attempts to break the architectural dependence on the code properties by jointly performing optimizations at the code-design, (decoder) algorithmic, and architectural levels. First, the interconnect problem stemming from the inherent randomness of LDPC codes is addressed by designing **architecture-aware** LDPC (AA-LDPC) codes having regularity features favorable for an efficient decoder implementation. Second, the memory overhead problem in current decoder implementations employing the TPMP algorithm is addressed by proposing a turbo-decoding message-passing (TDMP) algorithm for LDPC codes [4, 7, 8]. Moreover, a new parallel message update mechanism in the form of a message processing unit (MPU) is proposed that implements the recursions of the well-known BCJR algorithm [9] in differential form based on simple "max" operations that do not require lookup tables and incur negligible performance loss compared to the ideal case. Finally, at the architectural level, an efficient TDMP decoder architecture based on a regular and scalable dynamic message-transport network for routing messages to and from MPU's is proposed. Among other features, the decoder architecture is flexible enough to handle regular and irregular AA-LDPC codes, as well as gener-

This work was supported with funds from NSF under grants CCR 99-79381 and CCR 00-85929.

0-7803-7761-3/03/$17.00 ©2003 IEEE

alized low-density codes by simply modifying the MPU's.

The remainder of the paper is organized as follows. Section 2 introduces the concept of AA-LDPC codes. Section 3 briefly presents the TDMP algorithm, and Section 4 presents the TDMP decoder architecture. Finally, Section 5 presents some simulation results.

## 2. ARCHITECTURE-AWARE LDPC CODES

In this section, we introduce the concept of architecture-aware LDPC codes [7]. An AA-LDPC code is defined by a parity-check block matrix $\mathbf{H}$ having $B$ block rows and $D$ block columns such as the one shown in Fig. 2, where each block is an $S \times S$ binary sub-matrix and $S$ is a code-independent parameter. These submatrices are required to be either all-zero $S \times S$ matrices or *permutation* matrices. An $S \times S$ binary permutation matrix is simply the identity matrix $\mathbf{I}_{S\times S}$ whose rows (or equivalently columns) are randomly permuted. Hence, a regular $(c, r)$-LDPC code would have $r$ permutation matrices per block row and $c$ permutation matrices per block column, while an irregular LDPC code would have these matrices positioned according to the degree distribution of the code. The code length is given by $n = SB$, and the code rate is $R \geq 1 - D/B$. Figure 2 shows a parity-check matrix of a regular $(2, 4)$-LDPC code for $S = 6$ having two and four permutation matrices per block column and row, respectively.

The main advantages of AA-LDPC codes over other LDPC codes are twofold. First, they transform the LDPC decoding problem employing the TPMP algorithm into a turbo decoding problem [4] thus eliminating the storage required to save multiple check-to-bit messages (a savings of 75 %). This follows from the fact that the ones in the rows in each block row of $\mathbf{H}$ do not overlap, and consequently, the block rows can be processed independently by passing messages only between adjacent block rows as opposed to potentially all rows as is the case with the TPMP algorithm. Second, taking the next step further towards an efficient decoder implementation (as compared to [2]), the structure of AA-LDPC codes reduces the complexity of the interconnection network when it comes to forwarding and retrieving messages between the non-zero entries as defined in $\mathbf{H}$. Observe that the ones in the rows of $\mathbf{H}$, absent any structure, would generally have random column indices requiring $r$ $(n{:}1)$-multiplexers to read $r$ messages corresponding to the row, and then $r$ $(1{:}n)$-demultiplexers to write the results back in memory in proper order. This solution quickly becomes impractical for large code lengths $n$, or when multiple rows are accessed in parallel to increase decoding throughput. Moreover, the overhead of the control mechanism of the (de-)multiplexers which keeps track of the column positions of all the ones in the parity-check matrix becomes too complex to implement. On the other hand, with the prescribed structure of AA-LDPC codes, only $rS$ $(J{:}1)$ instead of $rS$ $(n{:}1)$-(de-)multiplexers are needed to access messages from memory, where $J < B - r \ll n$. Further, every $S$ of these (de-)multiplexers can be controlled by a common simple sequencer programmed to trace the corresponding permutations.

While the parity-check matrix shown in Fig. 2 has desirable architectural properties, it is not a priori clear whether LDPC codes having such structure would achieve comparable BER performance to randomly constructed codes of similar complexity. In [7, 8, 10], it was shown that indeed AA-LDPC codes based on cyclotomic cosets [8] and Ramanujan graphs [10, 11] have BER performance that compares favorably with randomly generated codes.

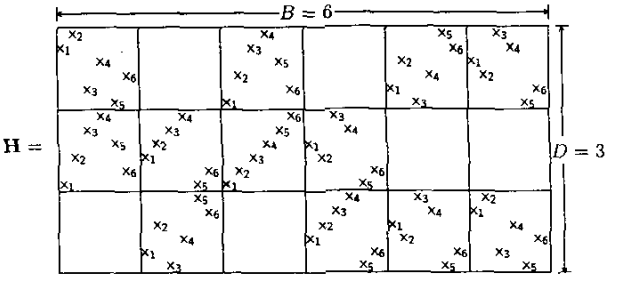



Figure 2: A parity-check matrix of an AA-LDPC code with $S = 6$. The label $\mathsf{x}_i$ denotes a one at column position $i$ within a block.

## 3. TURBO DECODING OF AA-LDPC CODES

In this section, we briefly describe the turbo-decoding message-passing (TDMP) algorithm [4,10] for AA-LDPC codes, and present a reduced-complexity message update mechanism in the form of a soft-input soft-output (SISO) message processing unit (MPU) [7, 8] based on the BCJR algorithm [9]. For more details on the TDMP algorithm and the MPU, the reader is referred to [4,7,8,10]. Assume an AA-LDPC code is defined by a $D \times B$ parity-check block matrix $\mathbf{H}$ having parameter $S$. The $D$ block rows of $\mathbf{H}$ are processed in tandem by $S$ MPU's that operate in parallel. The MPU's together form a SISO decoder which we designate by $\mathbf{D}$. The messages corresponding to the $D$ block rows are stored in $B$ memory modules $\mathbf{M}_1, \cdots, \mathbf{M}_B$, where $\mathbf{M}_i$ has $S$ memory banks and $(c_j - 1)$ words per bank ($c_j \geq 1$ is the number of non-zero sub-matrices in block column $j, j = 1, \cdots, B$).

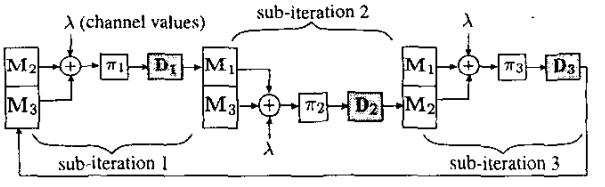



Figure 3: Block diagram of the TDMP algorithm showing the message exchange between the decoders for $D = 3$. $\mathbf{M}_i, \pi_i$, and $\mathbf{D}_i$ are the memory, interleaver, and SISO blocks of the $i$th block row.

The TDMP algorithm is described for simplicity with reference to the matrix shown in Fig. 2 for $D = 3$. The algorithm performs $D = 3$ decoding sub-iterations on the block rows of $\mathbf{H}$, and the operations of the algorithm in each of these sub-iterations are illustrated in the unfolded block diagram shown in Fig. 3. Starting from block row 1, extrinsic reliability values are computed for each bit using SISO decoder $\mathbf{D_1}$ and the input channel reliability values $\lambda_j, j = 1, \cdots, n$, assuming that the bit belongs to the code defined by block row 1. This extrinsic information is fed as *a priori* information through an interleaver ($\pi_1$) to SISO decoder $\mathbf{D_2}$ operating on the second block row. The interleaver can be factored into at most $B$ $S$-to-$S$ independent permuters following the structure of the AA-LDPC code. $\mathbf{D_2}$ in turn updates the extrinsic reliability values assuming that the bits belong to the code defined by block row 2. The process is repeated for the third block row. A single update of messages based on one block row is referred to as a *sub-iteration*, and a round of updates across all the block rows

constitutes a single decoding iteration. In the final iteration, hard decisions are made based on the *posterior* reliability values read (after de-interleaving) from the third SISO decoder.

The TDMP algorithm has two main advantages over the TPMP algorithm [4, 10]: (1) It eliminates the storage required to save *multiple* check-to-bit messages and replaces them with a *single* message corresponding to the most recent check-message update, and (2) it exhibits a faster convergence behavior requiring between 20 %-50 % fewer decoding iterations to converge for a given SNR (and hence higher decoding throughput) compared to the TPMP algorithm.

### 3.1. Low-Complexity Message Computation

The commonly employed message-update mechanism based on Gallager's equations [2] was shown [8] to be prone to quantization noise resulting in increased decoding latency and switching activity (and hence power consumption) in the decoder. In [8], an alternative approach for computing messages was proposed by using a simplified form of the BCJR algorithm [9] tailored to the syndrome trellis of single parity-check code (see [7, 9] for terminology). In this section, we present a reduced-complexity message update unit based on [9] that eliminates the need for lookup-tables, hence simplifying implementation cost considerably especially in parallel MPU implementations. In the proposed method, messages are updated using a simple "max-quartet" bivariate function $Q(x, y)$ defined as

$$Q(x,y) = \max(x,y) - \max(x+y,0) + \max\Big(\frac{5}{8} - \frac{|x-y|}{4}, 0\Big) - \max\Big(\frac{5}{8} - \frac{|x+y|}{4}, 0\Big), \quad (1)$$

which approximates the key equations of the BCJR algorithm in differential form. In [7], we showed that $Q(x, y)$ is a simpler (implementation-wise) and more accurate approximation of the ideal key equations than other approximations available in literature (e.g., [12]). Figure 4(a) shows a logic circuit implementing (1). In terms of $Q(x, y)$, the key equations of the BCJR algorithm simplify to

$$\Delta\alpha' = Q\big(\Delta\alpha, \lambda\big), \Delta\beta' = Q\big(\Delta\beta, \lambda\big), \Lambda = Q\big(\Delta\alpha, \Delta\beta\big). \quad (2)$$

Figure 4(b) shows a parallel dataflow graph implementing the key equations of the BCJR algorithm using the function $Q(x, y)$.

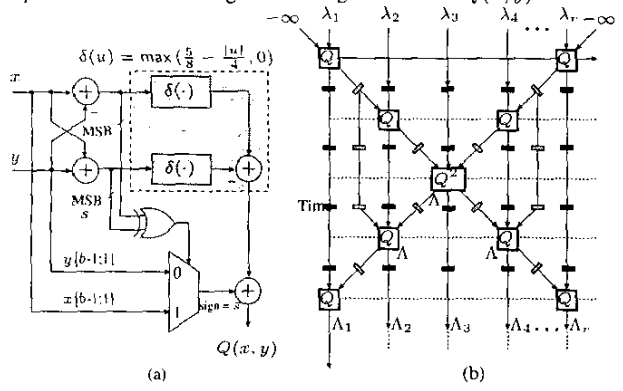



Figure 4: A SISO MPU using the max-quartet function: (a) Logic circuit for $Q(x, y)$, and (b) dataflow graph of the MPU.

## 4. DECODER ARCHITECTURE

A decoder architecture implementing the TDMP algorithm for regular and irregular AA-LDPC codes is shown in Fig. 5. The architecture is composed of $B$ memory modules for storing messages, $S$ MPU's that operate in parallel, and read/write networks for transporting messages between memory and these MPU's. The decoder completes a single decoding iteration by performing a round of $D$ updates across the block rows of $\mathbf{H}$. An update corresponding to a single block row $i$ constitutes a sub-iteration which involves the following steps:

1- The read-network (shown in Fig. 6) performs $c$ read operations from memory, where $c$ is the maximum bit-node degree of the block row, using $r$ block selectors that select the appropriate messages corresponding to all other block rows (than $i$) from memory. These messages are then accumulated on top of earlier messages read from previous block rows (including the intrinsic channel reliabilities of the bits) using a row of $r$ column accumulators. Finally, after $c$ accumulation steps, the resulting messages are independently permuted to match the structure of the permutation matrices of block row $i$ using a row of $r$ permuters. The network then forwards $r$ messages to each of the $S$ MPU's.

2- The MPU's update the messages in parallel as described in Section 3.1.

3- The updated messages are routed by the write-network to the appropriate memory modules scheduled to receive messages related to block row $i$. The write network, shown in Fig. 7, performs the reverse operations of the read network, excluding message accumulation.

4- The messages are written in the designated memory modules and the address counters are updated.

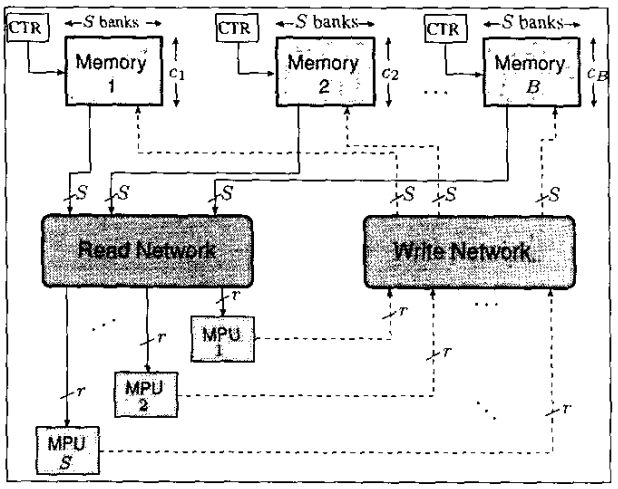



Figure 5: TDMP decoder architecture.

## 5. SIMULATION RESULTS

A TDMP decoder architecture for an AA-LDPC code of length 2304 having a five bit datapath was synthesized using $0.18\,\mu$m.

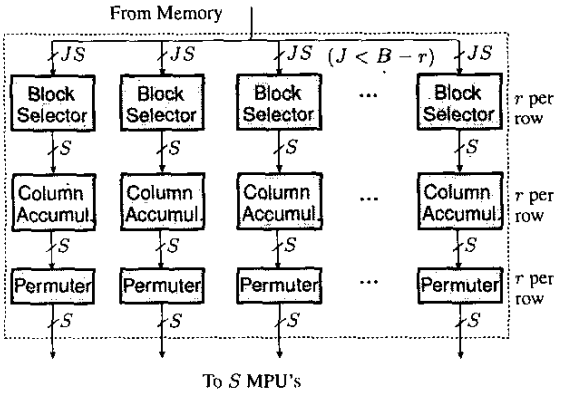



Figure 6: Read-network architecture of the TDMP algorithm.

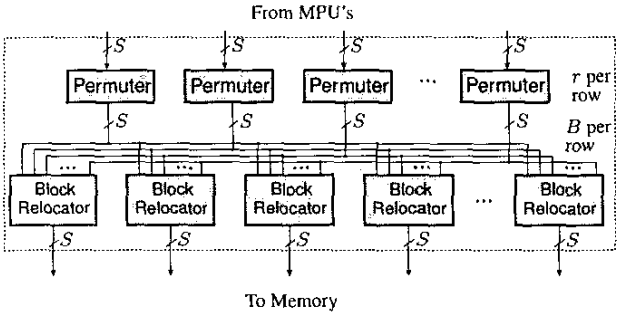
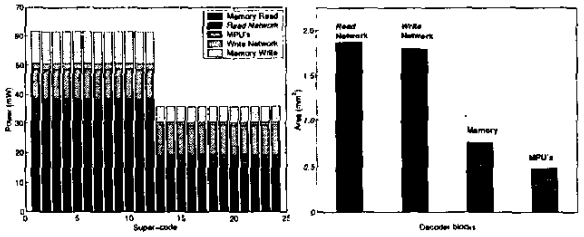



Figure 7: Write-network architecture of the TDMP algorithm.

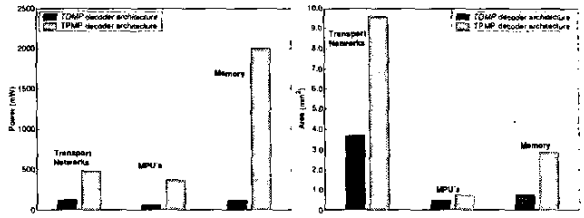
1.8 V CMOS technology, and power and area characterization were done using the Synopsis tool suite. The decoder attains a throughput of 1.92 Gbits/s at 200 MHz. The average power consumption is 1,179 mW, and the decoder area is 4.92 $mm^2$. Figure 8 shows the average power and area distribution of the main blocks of the decoder. As expected, memory reads consume most of the power (50.42 %) followed by the MPU's (20.73 %). The remaining blocks of the architecture consume 28.85 % of the power. The networks (including the peripheral interconnect) occupy 3.67 $mm^2$ or 74.6 % of the area of the decoder. The optimized MPU's occupy only an area of 0.48 $mm^2$ (9.88 %). More importantly, the reduced memory requirement of the TDMP algorithm is evident in that memory occupies only 0.76 $mm^2$ or 16.06 % of the area.

The TDMP decoder architecture was also compared with a decoder architecture implementing the TPMP algorithm and Gallager's update equations having the same throughput. Figure 9(a) compares the power consumed by the transport networks, MPU's (check and bit function units for the TPMP decoder), and memory. The figure demonstrates the power efficiency of the TDMP decoder compared to the TPMP decoder, resulting in a savings of 74.28 %, 83.06 %, and 93.81 % in power consumption in the networks, MPU's, and memory modules, respectively. The overall savings in power consumption is 89.13 %. In terms of area, Fig. 9(b) similarly shows significant reduction in the area of the networks (61.76 %), MPU's (33.33 %), and memory (73.68 %). respectively. The overall area savings is 62.80 %. Finally, the TDMP decoder requires 60.5 % less interconnect wires (including both networks and peripherals) than the TPMP decoder.




(a) (b)

Figure 8: (a) Average power dissipation, and (b) area distribution of the main blocks of the TDMP decoder.




(a) (b)

Figure 9: Comparison of (a) power consumption, and (b) area between the TDMP and the TPMP architectures.

## 6. REFERENCES


[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," in *IEEE Int. Conf. on Communications*, 1993, pp. 1064–1070.

[2] R. G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, MA, 1963.

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

[4] Mohammad M. Mansour and Naresh R. Shanbhag, "Turbo decoder architectures for low-density parity-check codes," in *Proc. IEEE GLOBECOM, Taipei, Taiwan*, Nov. 2002.

[5] Flarion Technologies, "Vector-LDPC$^{TM}$core solutions," Feb. 2002.

[6] 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, pp. 742–745.

[7] Mohammad M. Mansour and Naresh R. Shanbhag, "High-throughput memory efficient decoder architectures for LDPC codes," *submitted to IEEE Transactions on VLSI Systems*, 2002.

[8] Mohammad M. Mansour and Naresh R. Shanbhag, "Low-power VLSI decoder architectures for LDPC codes," in *ISLPED 2002, Monterey, CA*, Aug. 2002, pp. 284–289.

[9] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. on I.T.*, pp. 284–287, Mar. 1974.

[10] Mohammad M. Mansour and Naresh R. Shanbhag, "Memory-efficient turbo decoder architectures for LDPC codes," in *SiPS 2002, San Diego, CA*, Oct. 2002.

[11] Mohammad M. Mansour and Naresh R. Shanbhag, "Construction of LDPC codes from Ramanujan graphs," in *Conference on Info. Sciences and Systems 2002, Princeton University*, Mar. 2002.

[12] X. Y. Hu et al., "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *GLOBECOM 2001*, 2001, vol. 2, pp. 1036–1036E.