

High-Throughput LDPC Decoders

Mohammad M. Mansour, *Member, IEEE*, and Naresh R. Shanbhag, *Senior Member, IEEE*

Abstract—A high-throughput memory-efficient decoder architecture for low-density parity-check (LDPC) codes is proposed based on a novel turbo decoding algorithm. The architecture benefits from various optimizations performed at three levels of abstraction in system design—namely LDPC code design, decoding algorithm, and decoder architecture. First, the interconnect complexity problem of current decoder implementations is mitigated by designing architecture-aware LDPC codes having embedded structural regularity features that result in a regular and scalable message-transport network with reduced control overhead. Second, the memory overhead problem in current day decoders is reduced by more than 75% by employing a new turbo decoding algorithm for LDPC codes that removes the multiple check-to-bit message update bottleneck of the current algorithm. A new merged-schedule merge-passing algorithm is also proposed that reduces the memory overhead of the current algorithm for low to moderate-throughput decoders. Moreover, a parallel soft-input-soft-output (SISO) message update mechanism is proposed that implements the recursions of the Balh–Cocke–Jelinek–Raviv (BCJR) algorithm in terms of simple “max-quartet” operations that do not require lookup-tables and incur negligible loss in performance compared to the ideal case. Finally, an efficient programmable architecture coupled with a scalable and dynamic transport network for storing and routing messages is proposed, and a full-decoder architecture is presented. Simulations demonstrate that the proposed architecture attains a throughput of 1.92 Gb/s for a frame length of 2304 bits, and achieves savings of 89.13% and 69.83% in power consumption and silicon area over state-of-the-art, with a reduction of 60.5% in interconnect length.

Index Terms—Low-density parity-check (LDPC) codes, Ramanujan graphs, soft-input soft-output (SISO) decoder, turbo decoding algorithm, VLSI decoder architectures.

I. 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 which were first discovered by Gallager in 1961 [2]. Recent breakthroughs to within 0.0045 dB of AWGN-channel capacity were achieved with the introduction of irregular LDPC codes in [3], [4] putting LDPC codes on 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 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. The iterative decoding process of both codes consists of two main steps [5]: 1) computing independent messages proportional to the *a posteriori* probability distributions of the code bits and 2) communicating the messages. The complexity incurred in both steps depends on how messages are communicated with respect to the process of computing the messages. In turbo codes, the communication mechanism is defined by a simple pseudorandom message interleaver that is external to the process of computing the messages, while the computational complexity is proportional to the code length and memory. On the other hand, the communication mechanism for LDPC codes is defined by a pseudorandom bipartite graph and is internal with respect to message computation (i.e., an internal interleaver), while the computational complexity is very low (\sim order of the logarithm of the code length). In both cases, the performance depends on the properties of the interleaver: the scrambling of channel burst errors for turbo codes, and the extremal properties (such as girth and expansion coefficient) of the bipartite graph defining the code for LDPC codes.

However, unlike turbo codes, the lack of any structural regularity in the message communication mechanism of LDPC codes has far reaching consequences on the decoder implementation such as: 1) complex interconnects that limit the amount of inherent parallelism that can be exploited efficiently in a parallel decoder implementation and 2) stringent memory requirements in a serial decoder implementation that limits the applicability of LDPC codes in latency and power-sensitive applications. The dependence of the decoder architecture on the code properties is depicted in the traditional LDPC design flowgraph in Fig. 1(a), where the dependence of the architecture on the code structure is depicted by a dashed line. All current decoder architectures [6]–[11] apply optimizations at one system abstraction level, and either retain the disadvantages of or introduce further disadvantages at other levels, while the limited code-design optimizations introduced in [12] require significant overhead in terms of network control and source synchronization.

This paper attempts to break the architectural dependence on the code properties by performing optimizations at three levels of abstraction—code-design level, (decoder) algorithmic

Manuscript received September 10, 2002; revised February 3, 2003. This work was supported by the National Science Foundation under Grant CCR 99-79381 and Grant CCR 00-73490.

M. M. Mansour was with the University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA. He is now with the Department of Electrical and Computer Engineering, American University of Beirut, Beirut, 1107 2020 Lebanon (e-mail: mmansour@aub.edu.lb).

N. R. Shanbhag is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: shanbhag@uiuc.edu).

Digital Object Identifier 10.1109/TVLSI.2003.817545

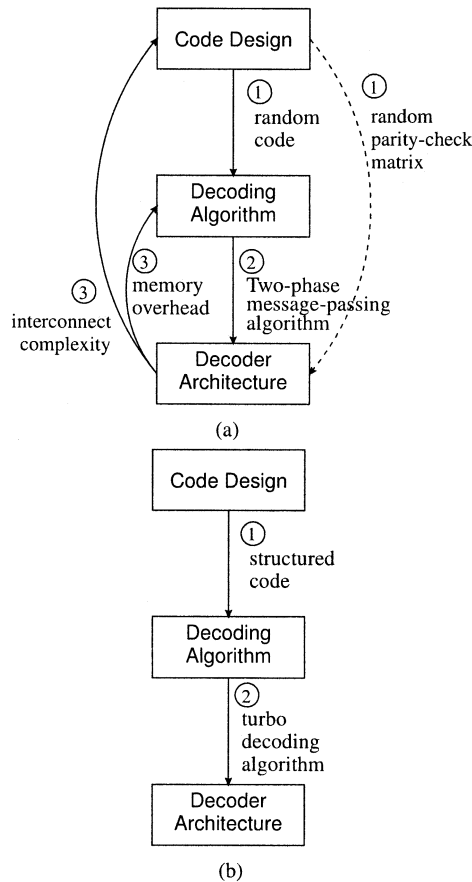


Fig. 1. LDPC decoder design flowgraph: (a) traditional and (b) proposed.

level, and architectural level—according to the proposed design flowgraph shown in Fig. 1(b) where the architecture is decoupled from the code structure. In particular, the contributions of the paper are: the concept of structured or architecture-aware LDPC codes to solve the interconnect problem, a new decoding algorithm based on the turbo-decoding principle for LDPC codes along with an efficient message-update mechanism to solve the memory overhead problem and improve the convergence speed of the decoder, and a new programmable and scalable high-throughput turbo-decoder architecture for LDPC codes that achieves savings of 89.13% and 62.8% over existing architectures for frame lengths of 2304, with a reduction of 60.5% in interconnect length.

First, LDPC codes are essentially random in nature (and hence, require highly irregular hardware interconnectivity), which goes against efficient VLSI implementation paradigms that call for regular and structured design. This immediate difficulty is best addressed by designing good and architecture-aware LDPC codes having regularity features favorable for an efficient decoder implementation. Two classes of high-performance codes based on cyclotomic cosets [13] and Ramanujan graphs [14], [15] targeted for short-to-medium and large code lengths, respectively, are shown to possess these features.

Second, the memory overhead problem in current decoder implementations is addressed by proposing the concept of turbo-decoding of LDPC codes [5], [13]. The overhead

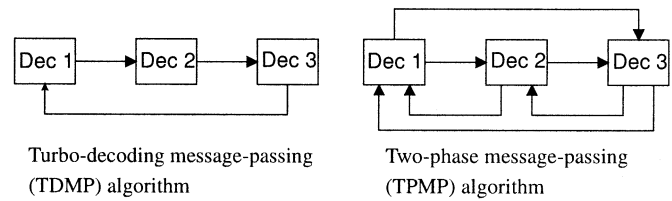


Fig. 2. TDMP algorithm versus the TPMP algorithm.

problem refers to the memory overhead of the two-phase message-passing (TPMP) algorithm [2] commonly employed in current day decoders. Related ideas in the context of optimized two level generalized low-density (GLD) codes were considered in [16]. An LDPC code is viewed as a code concatenated from a number of constituent codes connected through interleavers [5] which are defined by the parity-check matrix of the code. The turbo-decoding message-passing (TDMP) algorithm decodes each of these constituent codes in tandem by passing messages only between adjacent decoders in the pipeline as opposed to all other decoders as is the case with the TPMP algorithm (see Fig. 2). The advantages of the TDMP algorithm are twofold: 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 20%–50% fewer decoding iterations to converge for a given SNR (and hence, higher decoding throughput) compared to the TPMP algorithm.

Moreover, a new parallel soft-input soft-output (SISO) message update mechanism in the form of a message processing unit (MPU) based on the BCJR algorithm [13], [17] is proposed. The MPU implements the recursions of the BCJR algorithm using metric differences based on simple “max” operations that do not require lookup tables and incur negligible performance loss compared to the ideal case. The proposed update equations perform better than the approximations proposed in [18] both in terms of accuracy and hardware complexity, and, contrary to the observations made in [6], it is shown that the SISO MPU is indeed suitable for high-throughput applications.

Finally, at the architectural level, an efficient programmable architecture based on the TDMP algorithm is proposed that handles both regular and irregular LDPC codes. A regular and scalable dynamic message-transport network for routing messages to and from the MPUs is proposed. Among other features, the decoder architecture is flexible enough to handle GLD codes [16], [19] as well by simply modifying the MPUs.

The remainder of the paper is organized as follows. Section II introduces LDPC codes and the standard TPMP algorithm. Section III presents basic and improved LDPC decoder architectures, and identifies their drawbacks in terms of hardware complexity. Architecture-aware LDPC codes are introduced in Section IV. Sections V and VI propose the TDMP algorithm and the SISO MPU for message computation. Section VII presents a programmable TDMP architecture for AA-LDPC codes together with the memory and message-transport network architectures. Finally, Section VIII presents some simulation results, and Section IX concludes the paper.

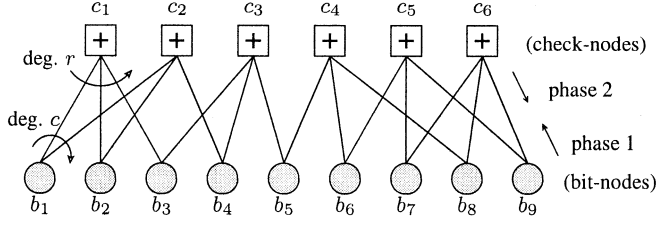


Fig. 3. Bipartite graph of a binary regular (2, 3)-LDPC code of length 9. The graph has $n = 9$ bit nodes and $m = 6$ check nodes. The sum of the values of all bit nodes connected to a check node is equal to zero.

II. BACKGROUND

We start with a formal definition of LDPC codes and the standard iterative decoding algorithm.

A. LDPC Codes

An LDPC code is a linear block code defined by a sparse parity-check matrix $\mathbf{H} = [h_{ij}]_{m \times n}$ and is described by a bipartite or factor graph [20] (see Fig. 3) whose *reduced* adjacency matrix is \mathbf{H} . The graph has on one side m check nodes $\{c_1, c_2, \dots, c_m\}$ corresponding to the m rows of \mathbf{H} , and on the other n bit nodes $\{b_1, b_2, \dots, b_n\}$ corresponding to the n columns of \mathbf{H} . A bit-node b_j is connected to a check node c_i if the entry h_{ij} of \mathbf{H} is 1. The rate of the code is at least $1 - m/n$. In a regular (c, r) -LDPC code, bit nodes have degree c and check nodes have degree r in the graph. The number of edges in the corresponding bipartite graph is the Hamming weight of \mathbf{H} given by $w(\mathbf{H}) = nc = mr$, and the density of \mathbf{H} is defined as $d(\mathbf{H}) = w(\mathbf{H})/mn = c/m = r/n$. The minimum distance of these codes increases linearly with the block length n for a given code rate and node degrees r and c [2].

In an *irregular* (C, R) -LDPC code, the bit-node and checknode degrees are drawn from the sets R and C , respectively. Typically, regular codes are easier to encode and have a simpler decoder architecture than irregular codes, however the latter achieve higher coding gain (e.g., [3], [4] report a rate-0.5 irregular code that has a threshold of 0.0045 dB from AWGN-channel capacity, outperforming the best codes known so far including turbo codes).

B. The Standard Message-Passing Algorithm

LDPC codes are decoded iteratively using the TPMP algorithm proposed by Gallager [2]. The algorithm 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 checknodes, and then updates of all check nodes are done in phase 2 by sending messages to neighboring bit nodes (see Fig. 3). Updates in each phase are independent and can be parallelized.

The TPMP algorithm [2] consists of the following main steps, where λ_j is the intrinsic channel reliability value of the j th bit, $R_{ij}[k]$ is the check-to-bit message from check node i to bit-node j at the k th iteration, $Q_{ij}[k]$ is the bit-to-check message from bit-node i to check node j at the k th iteration, $C[j]$ is the index

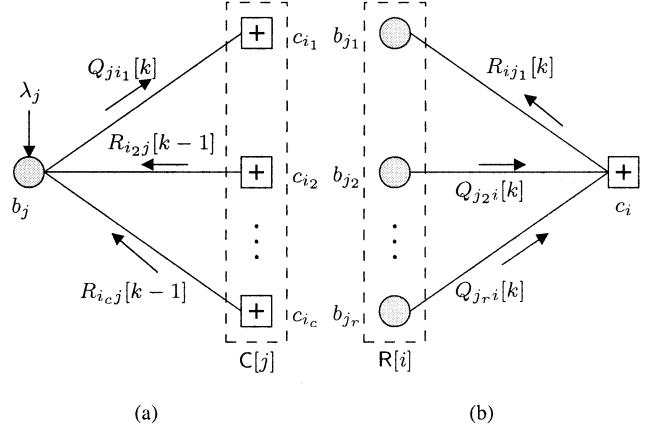


Fig. 4. Message updates on the TPMP algorithm. (a) Bit-to-check update and (b) check-to-bit update.

set of check nodes involving bit-node j , $R[i]$ is the index set of bit nodes involved in check node i :

- **Initialization:** For $k = 0$, the check-to-bit messages $R_{ij}[0]$ from the i th check node to the j th bit node are initialized to zero for all i , with $j \in R[i]$.
- **At iteration k :**
 - **Phase 1:** Compute for each bit node j the message $Q_{ji}[k]$ corresponding to each of its *check node* neighbors i according to [see Fig. 4(a)]

$$Q_{ji}[k] = \lambda_j + \gamma \sum_{i' \in C[j] \setminus \{i\}} R_{i'j}[k-1]. \quad (G1)$$

Note that the check-message $R_{i'j}[k-1]$ from the previous iteration $k-1$ is excluded from the summation in (G1) to eliminate correlation between messages. This is achieved by removing the index i from the index set of the summation using the notation $C[j] \setminus \{i\}$, implementing the so-called *extrinsic* principle.

- **Phase 2:** Compute for each check node i the message $R_{ij}[k]$ corresponding to each of its *bit node* neighbors j according to [see Fig. 4(b)]

$$R_{ij}[k] = \psi^{-1} \left[\sum_{j' \in R[i] \setminus \{j\}} \psi(|Q_{ji'}[k]|) \right] \quad (G2)$$

where $\psi(x) = -(1/2) \log(\tanh(x/2)) = \psi^{-1}(x)$, and δ_{ij} is a sign correction factor that depends on $|R[i]|$ and the sign of the terms $Q_{ji'}[k]$ in (G2). As in (G1), the message $Q_{ji}[k]$ targeted for check node i is left out of the summation in (G2) to implement the extrinsic principle.

- **Final iteration:** Compute for all bit nodes j the posterior reliability quantity given by

$$\Lambda_j = \lambda_j + \sum_{i \in C[j]} R_{ij}[k]. \quad (G3)$$

Hard decisions are then made based on the sign of Λ_j , $j = 1, \dots, n$, and the syndrome of the codeword is checked

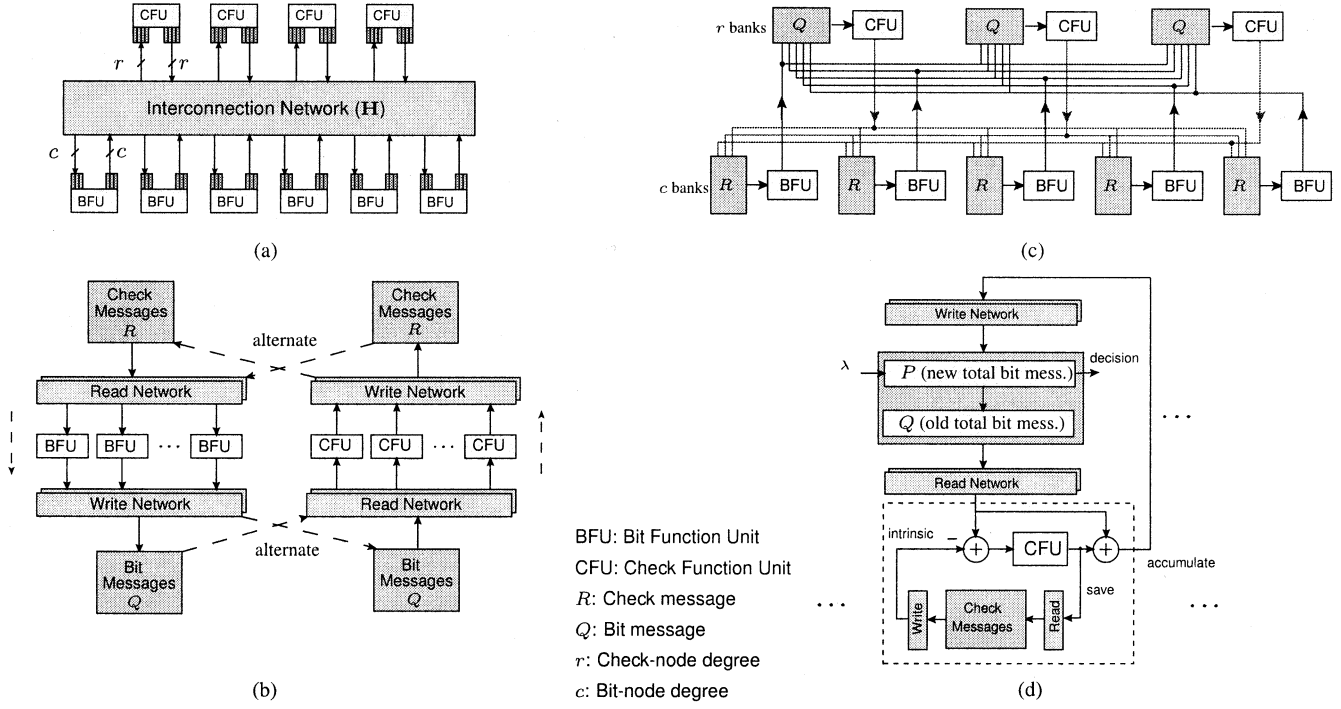


Fig. 5. LDPC decoder architectures: (a) parallel TPMP architecture, (b) serial TPMP architecture, (c) Interconnect-aware TPMP architecture, and (d) memory aware architecture implementing the MSMP algorithm.

for errors. Note that decisions are made using all information known about the bits, and hence all check-to-bit messages are included in the summation of (G3). ■

III. IMPROVED TPMP-BASED LDPC DECODER ARCHITECTURES

In this section, we characterize the hardware complexity of conventional LDPC decoder architectures based on the TPMP algorithm. Their drawbacks in terms of interconnect and memory overhead are identified, motivating the need for interconnect-aware and memory-aware architectures that mitigate these problems.

A. Hardware Complexity of Conventional TPMP Architectures

Direct decoder implementation based on the TPMP algorithm for randomly constructed LDPC codes in the form given in Section II-B presents a number of challenges. The check-to-bit message update equation (G2) is prone to quantization noise since it involves the nonlinear function $\psi(\cdot)$ and its inverse. The function $\psi(x)$ has a wide dynamic range (it rapidly increases for small x) which requires the messages to be represented using a large number of bits to achieve a fine resolution, leading to an increase in memory size and interconnect complexity (e.g., for a regular (3, 6)-LDPC code of length $n = 2048$ with 4-bit messages, an increase of 1 bit increases the memory size and/or interconnect wires by 25%).

Moreover, a parallel decoder implementation [7], [11] exploiting the inherent parallelism of the algorithm such as the one shown Fig. 5(a) is constrained by the complexity of the physical interconnect required to establish the graph connectivity of the code and, hence, does not scale well for moderate ($> 2K$)

to large code lengths. Long on-chip interconnect wires present implementation challenges in terms of placement, routing, and buffer-insertion to achieve timing closure. For example, the average interconnect wire length of the rate-0.5, length $n = 1024$, 4-bit LDPC decoder of [7] is 3 mm using 0.16- μm CMOS technology, and has a chip area of $7.5 \times 7 \text{ mm}^2$ of which only 50% is utilized due to routing congestion. In Fig. 5(a), the bit function units (BFUs) implement (G1) and the check function units (CFUs) implement (G2).

On the other hand, serial architectures [8], [9] in which computations are distributed among a number of function units that communicate through memory instead of a complex interconnect, as shown in Fig. 5(b), require a substantial memory overhead that amounts to four times the Hamming weight of the parity-check matrix. Each function unit requires separate read/write networks with complex control to access memory, and hence it is practical to use only a small number of function units compared to the parallel case, resulting in a low-throughput decoder. A heuristic approach aimed at simplifying implementation complexity was proposed in [9] by relaxing the inequality $i' \neq i$ in (G1). However, this leads to considerable loss in coding gain due to correlation between the messages and premature ending of decoding iterations.

B. Interconnect-Aware Decoder Architectures

An interconnect-aware decoder architecture targeted for moderate- to high-throughput applications was proposed in [13] to solve the interconnect problem of TPMP-based decoders. Fig. 5(c) shows the corresponding architecture. In this approach, structural regularity features are embedded in the parity-check matrix \mathbf{H} that reduce the complexity of the interconnect in implementation. The matrix \mathbf{H} is decomposed

into $S \times S$ submatrices with separate function units allocated to process each column and each row of these submatrices. Each of the function units reads its messages from local memory controlled by a modulo- S counter, and writes its results in remote memory pertaining to other function units using a simple static network with the counters serving to synchronize the memory operations. Compared with the parallel approach, the complexity of the interconnect due to this approach is scaled down by a factor of S . On the other hand, compared with the serial approach, it requires only half the amount of memory, and the switching networks for memory access are eliminated. In addition, two frames are processed simultaneously by the decoder [13] LDPC codes constructed using this approach compare favorably in terms of coding gain with randomly constructed codes having similar complexity for bit error rates (BERs) as low as 10^{-8} .

C. Memory-Aware Decoder Architectures

A merged-schedule message-passing (MSMP) algorithm that addresses the memory overhead problem of LDPC decoder architectures targeted for low- to moderate-throughput applications was proposed in [5]. In the MSMP algorithm, the computations of both phases of the TPMP algorithm are merged into a single phase as described in the pseudocode listed below in Algorithm 1, resulting in a considerable reduction in memory requirements. An MSMP decoder architecture is shown in Fig. 5(d).

Algorithm 1. Merged-Schedule Message-Passing Algorithm

```

for  $j = 1$  to  $n$  do
   $Q_j[0] \leftarrow P_j[0] \leftarrow \lambda_j$ 
for  $i = 1$  to  $m$ ,  $j \in R[i]$  do
   $R_{ij}[0] \leftarrow 0$ 
 $k \leftarrow 1$ 
while  $k < \text{MAXITER}$  do
  for  $i = 1$  to  $m$  do
    for all  $j \in R[i]$  do
       $R_{ij}[k] \leftarrow \psi^{-1}[\sum_{j' \neq j \in R[i]} \psi(|Q_{j'}[k-1] - R_{ij'}[k-1]|)] \cdot \delta_{ij}$  (G4)
       $P_j[k] \rightarrow P_j[k-1] + R_{ij}[k]$ 
    for  $j = 1$  to  $n$  do
       $Q_j[k] \leftarrow P_j[k]$ 
       $P_j[k] \leftarrow \lambda_j$ 
     $k \leftarrow k + 1$ 
end while

```

The savings are mainly due to eliminating the storage required to save the multiple bit-to-check messages computed using equation (G1). Since the computations in this equation are additive in nature, the condition $i' \neq i$ can be satisfied by first computing the total sum $\lambda_j + \sum_{i' \in C[j]} R_{i'j}[k-1]$ and then subtracting the intrinsic message $R_{ij}[k-1]$. Thus, only one copy of the total bit-to-check message Q_j is maintained in a buffer Q for each bit holding this total sum [refer to Fig. 5(d)]. Updated check-messages to the bit under consideration are accumulated in a temporary buffer P having size n and initialized with the intrinsic channel reliabilities λ_j . The

buffer P maintains a single running sum of the bit-to-check messages instead of multiple copies pertaining to different check nodes. Unlike [9], a copy of the intrinsic check-to-bit messages $R_{ij}[k-1]$ is saved in a buffer R of size $m \times r$ so that it is subtracted from the cumulative sum in the next iteration to implement the condition $j' \neq j$ in (G2). At the end of each iteration, P and Q are interchanged and P is reloaded with the λ_j 's. The architecture stores $nc + 2n$ messages as opposed to $4nc$, a savings of $(75-50/c)\%$, and hence the name memory-aware decoder architecture. The CFU block implements equation (G4) defined in the pseudo-code below with the ψ -function pre-coded in a lookup table (LUT). Multiple CFU blocks scheduled to operate on distinct bits instead of one as shown in Fig. 5(d) can be used to increase throughput. Extra logic must be added to ensure correct operation in case some conflicts are not removable through scheduling. However, the complexity of the interconnect is still the bottleneck motivating the need for further transformations at the code design and algorithmic levels.

D. Discussion

The architectures presented so far benefit from optimizations performed at one abstraction level and hence do not achieve simultaneously the desired objectives of reduced memory overhead, reduced interconnect complexity, and scalability while maintaining the code performance promised by randomly constructed LDPC codes. In Sections IV–VI, we address these objectives through combined efforts targeting the code structure, decoding algorithm, and architecture.

IV. ARCHITECTURE-AWARE LDPC CODE DESIGN

In this section, we propose a class of architecture-aware LDPC codes that solves the interconnect, memory overhead, and scalability problems associated with LDPC decoders. Motivated by Gallager's original construction of LDPC codes [2], we view the parity-check matrix \mathbf{H} of a regular (c, r) -LDPC code as a concatenation of c parity-check matrices $\mathbf{H}^1, \dots, \mathbf{H}^c$, corresponding to c supercodes $\mathcal{C}^1, \dots, \mathcal{C}^c$. This in turn enables us to decompose the interconnection network defined by the overall matrix \mathbf{H} into c smaller networks or interleavers that connect together adjacent supercodes. This step transforms the LDPC decoding problem into a turbo decoding problem [1], with the codes $\mathcal{C}^1, \dots, \mathcal{C}^c$ acting as constituent codes. Related ideas in the context of GLD codes concatenated from two supercodes were studied in [16]. To enable efficient implementation of the interleavers, we propose an architecture-aware decomposition of the matrices $\mathbf{H}^1, \dots, \mathbf{H}^c$ into null or pseudo-permutation matrices. The major advantages of the architecture-aware decomposition and the associated turbo decoding algorithm as discussed in this section and Section V, are that they reduce the complexity of the interconnect by $\mathcal{O}(n)$, decrease the memory requirements by at least 75% compared to serial TPMP architectures, and improve the decoding throughput due to the faster convergence of the turbo-decoding algorithm compared to the TPMP algorithm.

We start by introducing the concept of trellis representation of parity-check codes and their relation to LDPC codes and the

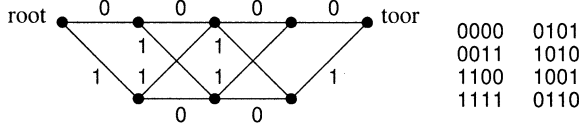


Fig. 6. A Trellis representation of a $(4, 3)$ -SPC code. The 8 codewords shown on the right correspond to all possible eight sequences of edge labels starting from the root and ending in the toor.

TPMP algorithm. The interconnect problem is identified, and then an architecture-aware decomposition of \mathbf{H} is presented to solve this problem.

A. Trellis Representation of Parity-Check Codes

The rows of a parity-check matrix \mathbf{H} correspond to parity-check equations constraining the values of the bits in a codeword. Hence, if the j th row contains r ones at column positions $\{j_1, j_2, \dots, j_r\}$, then this implies that the bits $x_{j_1}, x_{j_2}, \dots, x_{j_r}$, of a codeword \underline{x} must have even parity, or

$$x_{j_1} \oplus x_{j_2} \oplus \dots \oplus x_{j_r} = 0.$$

The set of all codewords of length r having even parity forms a code called the single parity-check (SPC) code of length $n_0 = r$ and dimension $k_0 = r - 1$ denoted as $(r, r - 1)$ -SPC code. The codewords of an $(r, r - 1)$ -SPC code can be represented conveniently using a trellis of length r composed of nodes and labeled edges as shown in Fig. 6. A node in the trellis is a state that represents the parity of the edge labels starting from the leftmost node (called the root) and ending in that node. The nodes in the top row including the root and the rightmost node (or toor) correspond to even parity states, while those in the bottom row correspond to odd parity states. A sequence of edge labels from the root to the toor defines a codeword of length r having even parity. The set of all 2^{r-1} such sequences defines an $(r, r - 1)$ -SPC code.

A parity-check matrix $\mathbf{H}_{m \times n}$ of an LDPC code containing r_i ones per row, $i = 1, \dots, m$, can be represented using m trellises corresponding to $(r_i, r_i - 1)$ -SPC codes. Using this representation, it is possible to describe an LDPC code and the corresponding flow of messages of the TPMP algorithm using an alternative graph based on the bipartite graph of the code. In this graph, a check node of degree r_i is replaced by the trellis of an $(r_i, r_i - 1)$ -SPC code, while the bit nodes are removed. The edges incident on a check node in the bipartite graph are connected to the appropriate sections of the trellis corresponding to that check node, and the edges incident on a bit node are connected directly to each other. If two rows in \mathbf{H} overlap in q column positions, then their trellises are connected by q edges at those overlapping positions. Fig. 7 shows a portion of the bipartite graph of Fig. 3 represented using trellises connected through edges. For a larger random \mathbf{H} such as the one shown in Fig. 8(a), the connections between the trellises quickly become congested, and hence they are represented as a random interleaver as shown in Fig. 8(b).

The bit-to-check and check-to-bit messages associated with the nodes of a bipartite graph are replaced by a single type of message called a SISO message in the trellis flowgraph.

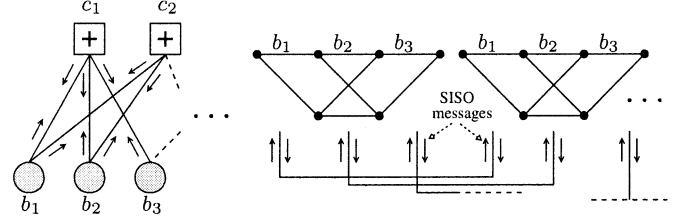


Fig. 7. Trellis representation of a portion of the bipartite graph of Fig. 3.

Section VI describes how these SISO messages are computed using a SISO decoder based on the BCJR algorithm [13] and [17]. This SISO decoder will replace the BFU and CFU blocks of Fig. 5 for message computation in the remainder of the paper.

B. Gallager's Construction of LDPC Codes

According to the definition given in Section II-A, the parity-check matrix of a regular (c, r) -LDPC need not satisfy any properties other than the row and column regularity constraints as shown in Fig. 8(a). We call this construction method the random construction method. The Gallager [2] or *normal* construction method constructs ensembles of regular (c, r) -LDPC codes of length n by defining the parity-check matrix $\mathbf{H}_{m \times n}$ of a code \mathcal{C} as a concatenation of c submatrices, each containing a single 1 in each column as shown in Fig. 8(c) for a regular $(3, 4)$ -LDPC code of length 16. The first of these submatrices \mathbf{H}^1 having size $(n/r) \times n$ defines a supercode \mathcal{C}^1 as the direct sum of n/r parity-check $(r, r - 1)$ -subcodes. Note that \mathcal{C}^1 satisfies a subset of the parity-check equations of \mathcal{C} , and hence \mathcal{C} is a subspace of \mathcal{C}^1 . The other submatrices $\mathbf{H}^2, \dots, \mathbf{H}^c$, are pseudorandom permutations of \mathbf{H}^1 , each of which defines a supercode \mathcal{C}^j on the corresponding subset of the parity-check equations. Hence, \mathcal{C} is simply the intersection of the supercodes $\mathcal{C}^1, \dots, \mathcal{C}^c$.

When the TPMP algorithm is applied to the random matrix \mathbf{H}_R , the message flowgraph takes the form shown in Fig. 8(b) where the two-state trellises represent the rows of \mathbf{H}_R as even parity-checks, and the random interleaver represents the flow of messages between the trellises according to the columns of \mathbf{H}_R . This random interleaver creates the interconnect complexity bottleneck referred to earlier. With the normal matrix \mathbf{H}_N in Fig. 8(c), however, the interleaver can be factored into c constituent interleavers according to $\mathbf{H}_N^1, \dots, \mathbf{H}_N^c$. The corresponding flow of messages can be modeled naturally as shown in Fig. 8(d) where the short two state trellises represent SPC subcodes. The analogy with turbo codes is depicted in Fig. 8(e) where the long trellises represent convolutional codes with constraint length 2. The rows corresponding to each supercode in \mathbf{H}_N do not overlap, and hence the constituent SPC subcodes can be decoded in parallel without exchanging information. But this decomposition of the parity-check matrix again presents a problem when it comes to forwarding and retrieving messages between these decoders and memory. The reason is that the ones in any row have random column indices requiring r $(n : 1)$ -multiplexers to read r messages corresponding to a row, and then r $(1 : n)$ -demultiplexers to write the results back in memory in proper order. This solution

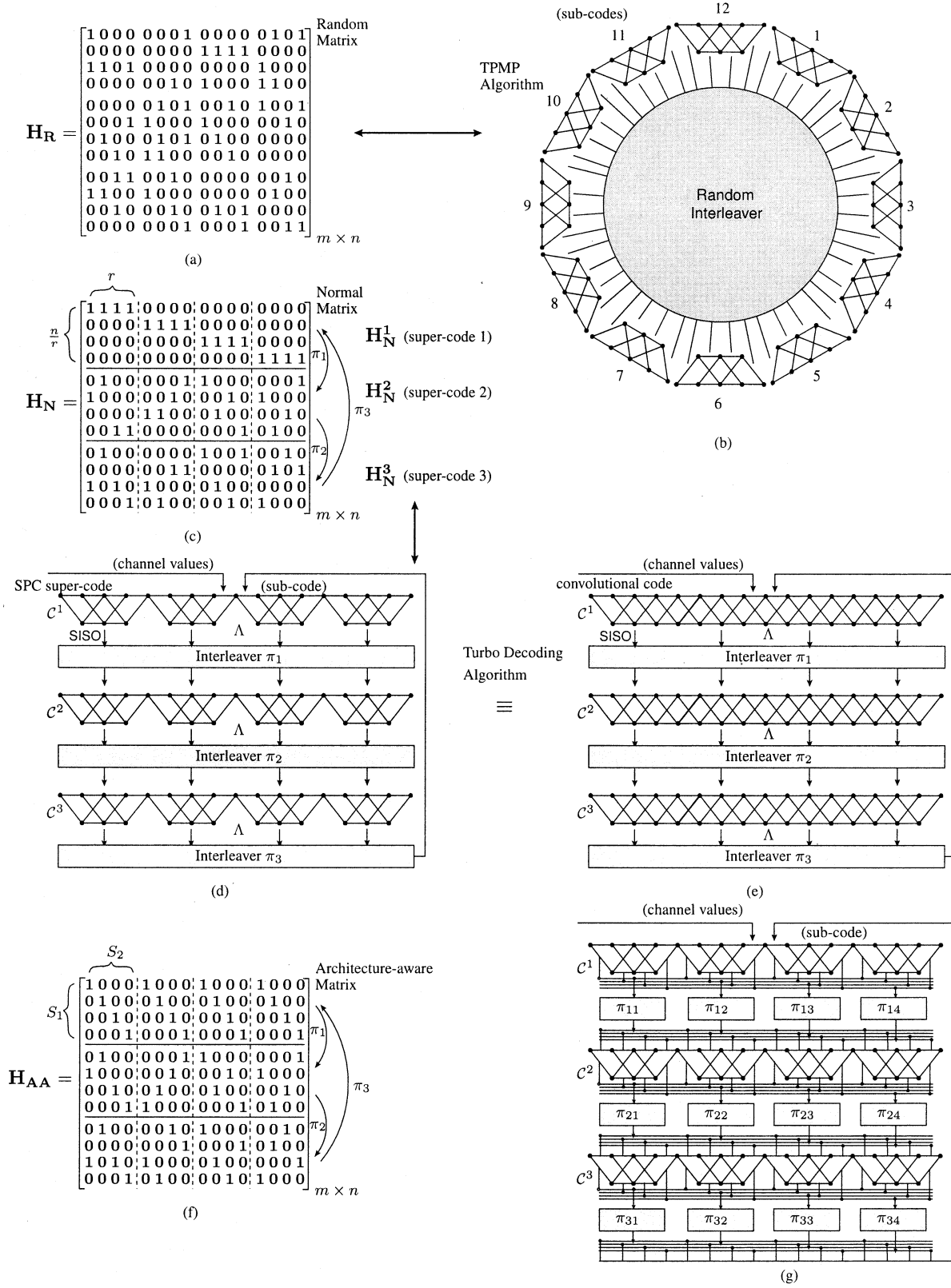


Fig. 8. Analogy between LDPC and turbo codes: (a) random parity-check matrix of an LDPC code, (b) message flowgraph of the TPMP algorithm when applied to a random parity-check matrix, (c) normal parity-check matrix decomposed into $r \times n$ submatrices, (d) message flowgraph of the TDMP algorithm, (e) message flowgraph of a turbo decoder, (f) architecture-aware parity-check matrix, and (g) message flowgraph corresponding to an AA-LDPC code.

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.

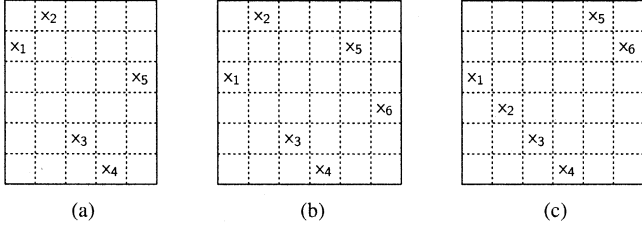


Fig. 9. Examples of pseudo-random permutation matrices. (a) Matrix with $S_1 = 6, S_2 = 5$, (b) an acyclic permutation matrix with $S_1 = 6$, and (c) a cyclic permutation matrix with $S = 6$. The label x_i denotes a one at column position i .

C. Architecture-Aware LDPC Codes

We propose an architecture-aware decomposition of the parity-check matrix that limits the column positions of the ones in a row, while still yielding LDPC codes with comparable BER performance to that of randomly constructed codes. In this method, the parity-check matrix \mathbf{H}_{AA} is decomposed into $S_1 \times S_2$ binary submatrices where S_1, S_2 are code-independent parameters such that $S_1 \geq S_2$. These submatrices, which we call pseudo-permutation matrices, satisfy the following two conditions: each column has at most one 1, and each row has at most one 1. For example, binary permutation matrices with $S_1 = S_2 = S$ satisfy these conditions. Note that zero columns, zero rows, and null matrices are allowed under this definition. Fig. 9 shows three examples of pseudo-permutation matrices.

A regular (c, r) -LDPC code can be constructed such that its parity-check matrix has r permutation matrices per block row and c permutation matrices per block column ($S_1 = S_2 = S$, and S is arbitrary). An irregular LDPC code would have these matrices positioned according to the degree distribution of the code, with $S_1 \geq S_2$ in general. Note that a normal matrix can be made architecture-aware by setting $S_1 = n/r$ and $S_2 = r$, and permuting the columns in each block row independently to satisfy the above two conditions. We associate the following two parameters with \mathbf{H}_{AA} : $B \triangleq n/S_2$ and $D = m/S_1$. Figs. 8(f) and 10 show the parity-check matrix of a $(3, 4)$ -regular and an irregular AA-LDPC code, respectively. The regular code has $S_1 = S_2 = 4$ and $B = 4, D = 3$. The irregular code has $S_1 = S_2 = 6$ and $B = 6, D = 4$, where the block rows of \mathbf{H}_{AA} contain four or five permutation matrices and the block columns two to four permutations matrices. Since each row of the submatrices has at most a single nonzero element, only $(S_1 : 1)$ instead of $(n : 1)$ -(de-)multiplexers are needed to access messages from memory, where $S_1 \ll n$, a reduction of order

$$\frac{rS(BS - 1)}{rS \log(S)} = \frac{n - 1}{\log(S)} \sim \mathcal{O}(n).$$

Further, these (de-)multiplexers can be controlled by simple sequencers programmed to trace the corresponding permutations, resulting in a reduction in control overhead of

$$\frac{rS \log(n)}{rS/2 \log(S)} = \frac{2 \log(n)}{\log S} \sim \mathcal{O}(\log(n)).$$

Fig. 8(g) shows the message flowgraph corresponding to an AA-LDPC code where the interleavers are factored into smaller

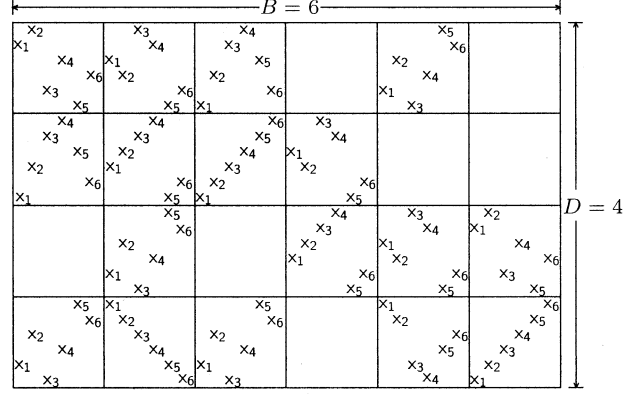


Fig. 10. An architecture-aware parity-check matrix with parameter $S = 6$.

interleavers of size S_1 for forwarding messages between supercodes.

While the parity-check matrix shown in Fig. 10 has desirable architectural properties, LDPC codes having such structure also provide comparable BER performance to randomly constructed codes of similar complexity. Fig. 11 (a)–(b) compares the BER performance of randomly constructed LDPC codes free of 4-cycles and two structured code-design methods: 1) using cyclotomic cosets [13] in Fig. 11(a), and 2) using Ramanujan graphs [15], [21] in Fig. 11(b). These two methods generate LDPC codes whose parity-check matrices are architecture-aware [13], [21], [22]. The figures clearly show that the BER performance of the structured codes using the techniques in [13], [15], [21], [22] compares favorably with the randomly generated codes in the BER- E_b/N_0 regions shown in the figures.

Note that it is also possible to randomly specify the size, number, structure, and positions of the pseudo-permutation submatrices in an AA-LDPC code. We call this method the generalized pseudo-permutation construction method of AA-LDPC codes. Fig. 11(c) compares the performance of an AA-LDPC code with $S_1 = S_2 = 42, D = 12, B = 24$ and rate 0.5 where the permutation matrices are chosen and positioned in \mathbf{H}_{AA} such that the girth of its graph is 6, with a randomly constructed code of the same length, rate, and girth. For code lengths more than 2K, careful consideration must be made regarding the minimum distance of the code to avoid error-floors at high SNR.

D. Discussion

The interconnect problem was identified by analyzing the message flow between two-state trellises pertaining to the TPMP algorithm. By decomposing the parity-check matrix of an LDPC code in such a way as to restrict the column positions of the ones, the LDPC decoding problem is transformed into a turbo-decoding problem where messages flow in tandem only between the adjacent supercodes as opposed to potentially all the subcodes absent any structure on the parity-check matrix. The interleavers are factored into smaller interleavers that are practical to implement. Fig. 8(e) illustrates the analogy with turbo codes where the supercodes are convolutional codes represented by contiguous trellises as opposed to SPC subcodes, and Fig. 8(g) shows how the interleavers factor out in an AA-LDPC code. Three conclusions can be drawn:

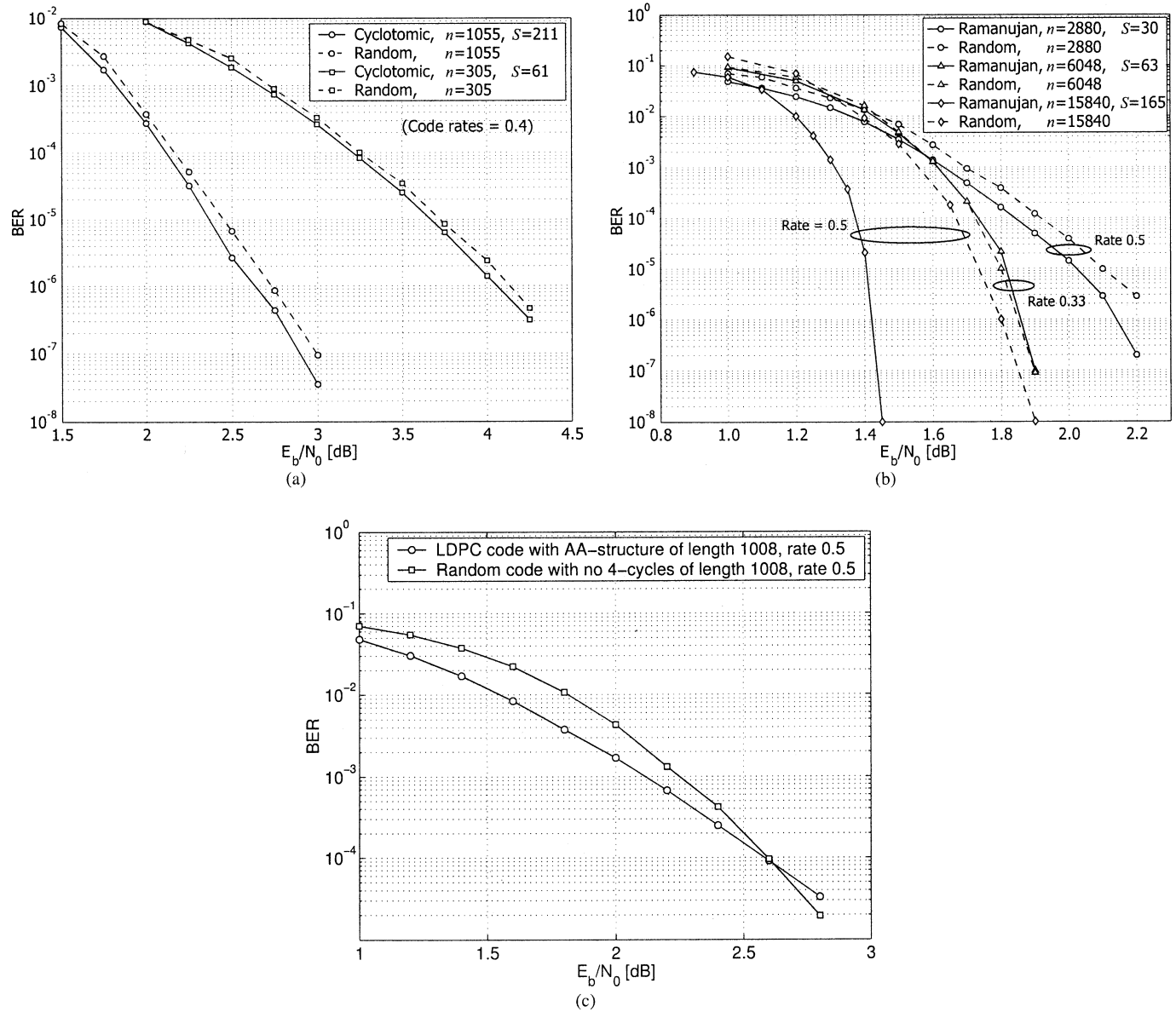


Fig. 11. Performance of AA-LDPC codes constructed from: (a) cyclotomic cosets [13], (b) Ramanujan graphs [15], [21], and (c) generalized pseudo-permutation matrices.

1) the turbo decoding algorithm is a practical algorithm for decoding AA-LDPC codes resulting in equivalent message communication complexity as that of turbo codes as discussed in Section V; 2) the algorithm processes one type of messages (SISO messages) corresponding to the extrinsic information from each supercode as opposed to bit and check messages in the TPMP algorithm; and 3) the computations of the messages are simpler in the case of LDPC codes compared to convolutional codes since they are based on short and independent trellises as discussed in Section VI.

V. TURBO DECODING OF AA-LDPC CODES

So far, we have established the analogy between LDPC codes and turbo-codes and identified an appropriate structure for the parity-check matrix to solve the interconnect problem. The next step is to solve the memory overhead problem. To this end, we

propose a new turbo-decoding message-passing (TDMP) algorithm [5] for AA-LDPC codes. The throughput and improvement in coding gain of the proposed algorithm over the TPMP algorithm, in addition to its memory advantages, are identified.

A. The Turbo-Decoding Message-Passing (TDMP) Algorithm

In this section, we propose the TDMP algorithm in the context of regular LDPC codes for simplicity. The algorithm applies to irregular codes as well with some minor modifications. As before, assume that an $m \times n$ normal parity-check matrix \mathbf{H}_{AA} defines a regular (c, r) -LDPC code \mathcal{C} with structure similar to that shown in Fig. 8(f) or Fig. 10. The decoding procedure is described with reference to Fig. 8(f) and Fig. 12. For each bit, extrinsic reliability values are computed using SISO decoder \mathbf{D}_1 assuming that the bit belongs to the first code \mathcal{C}^1 . This extrinsic information is fed as *a priori* information through an interleaver

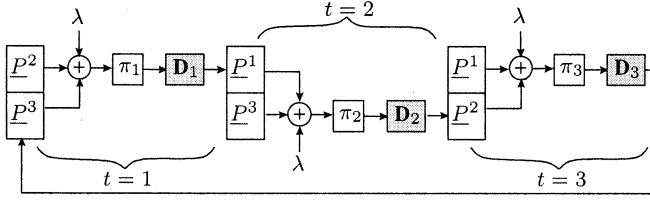


Fig. 12. Block diagram of the TDMP algorithm showing the message exchange between decoders for the case of three super-codes. P^i , π_i , and D_i are the memory, interleaver, and decoder blocks of the i th super-code, $i = 1, 2, 3$.

to the SISO decoder D_2 for C^2 . The second SISO decoder updates the extrinsic reliability values assuming that the bits belong to C^2 . The process is repeated for C^3 . A single update of messages based on one supercode is referred to as a sub-iteration, and a round of updates across all the supercodes 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 SISO decoder operating on the last supercode C^3 .

- **Input:** A regular (c, r) -LDPC code $C = C^1 \cap \dots \cap C^c$ of length n defined by $H_{m \times n} = [H^1; H^2; \dots; H^c]^T$; permutations $\{\pi_1, \dots, \pi_c\}$; intrinsic channel reliability values λ_j , $j = 1, \dots, n$.
- **Output:** Codeword \underline{x} such that $H\underline{x}^T = \mathbf{0}^T$.
- **Storage:** n memory buffers of size $(c - 1)$ denoted by P_j , $j = 1, \dots, n$ and initialized to zero. P_j^i denotes the i th element of P_j . A set of n counters $ctr_j := 1$ that point to the head of these buffers.
- **At iteration k :** Carry out c decoding sub-iterations corresponding to the supercodes C^i , $i = 1, \dots, c$, such that:
 - At sub-iteration t :

- 1) Compute the extrinsic reliabilities Q_j , $j = 1, \dots, n$ using the channel values λ_i , assuming P_j , as prior information that the codeword belongs to all supercodes except the t th supercode. This computation can be done using the BCJR algorithm as discussed in Section VI, or using (G2) with a slight modification as

$$Q_j = \psi^{-1} \left[\sum_{j' \in S_t[j] \setminus \{j\}} \psi \left(\left| \lambda_{j'} + \sum_{i=1}^{c-1} P_{j'}^i \right| \right) \right] \cdot \delta_{S_t[j]},$$

where $S_t[j]$ is the set of column indices of the bits of the subcode in C^t that contains bit j , and $\delta_{S_t[j]}$ is a sign-correction term that depends on the size of $S_t[j]$ and the sign of the arguments of

- 2) Permute Q according to π_t .
- 3) Save Q_j in P_j^{ctr} and advance $ctr_j = 1 + ctr_j \bmod (c - 1)$ for $j = 1, \dots, n$. The positions of the messages corresponding to each supercode rotate across the buffers with

each sub-iteration. The natural positions are restored after $(c - 1)$ iterations.

• **Final iteration:**

- Repeat steps 1–3 for the first $(c - 1)$ sub-iterations.
- At sub-iteration c :

- 1) Compute the posterior reliabilities Λ_j , $j = 1, \dots, n$ using

$$\Lambda_j = \psi^{-1} \left[\sum_{j' \in S_c[j]} \psi \left(\left| \lambda_{j'} + \sum_{i=1}^{c-1} P_{j'}^i \right| \right) \right] \cdot \delta_{S_c[j]}.$$

- 2) Make hard decisions: $\underline{x} \leftarrow \text{sgn}(\underline{\Lambda})$.

B. Throughput Advantage of the TDMP Algorithm

The BER performance and the rate of convergence of the TDMP algorithm are compared with the TPMP algorithm by simulating two LDPC codes C_1 and C_2 over a memoryless AWGN channel assuming BPSK modulation. C_1 is a regular (3, 5)-LDPC code of length 1200, while C_2 is a regular (4, 7)-LDPC code of length 4200. 100 000 frames were simulated using a maximum of 32 decoding iterations. Fig. 13(a) and (c) show the BER plots for C_1 and C_2 , respectively. The figures demonstrate that at the same SNR and for the same number of iterations, the proposed TDMP algorithm achieves much better BER. In particular, at SNR = 2.5 dB and with five iterations, the TDMP algorithm provides an order of magnitude improvement in the BER.

The number of iterations required for convergence using both algorithms is plotted in Fig. 13(b) and (d). As shown, the TDMP algorithm requires significantly less iterations to converge, where in some cases it requires close to half the number of iterations to converge compared to the TPMP algorithm. Hence, the decoder throughput can be improved by decreasing the number of iterations in order to achieve the same performance as that of the TPMP algorithm.

C. Memory Advantage of the TDMP Algorithm

In terms of memory, the total storage requirements of the TDMP algorithm for a general irregular LDPC code having bit node degrees c_j , $j = 1, \dots, n$, is $\sum_{j=1}^n (c_j - 1)$, corresponding to the extrinsic messages of all but one of the supercodes. On the other hand, a parallel architecture implementing the TPMP algorithm needs to store $2 \sum_{j=1}^n c_j$ messages, serial and interconnect-aware architectures need to store $4 \sum_{j=1}^n c_j$ messages, while the MSMP architecture needs to store $2n + \sum_{j=1}^n c_j$ messages. Therefore, the savings compared to parallel, serial, interconnect-aware TPMP architectures, and memory-aware MSMP architectures are $(50 + 50n / (\sum_{j=1}^n c_j))\%$, $(75 + 25n / (\sum_{j=1}^n c_j))\%$, and $(300n / (2n + \sum_{j=1}^n c_j))\%$, respectively. Fig. 14 plots the savings for regular (3, 6)-LDPC codes of length $n = 2048$ and rate 0.5. The plot shows significant savings in memory overhead are achieved by the TDMP algorithm over state-of-the-art which is valuable in applications where power is at a premium.

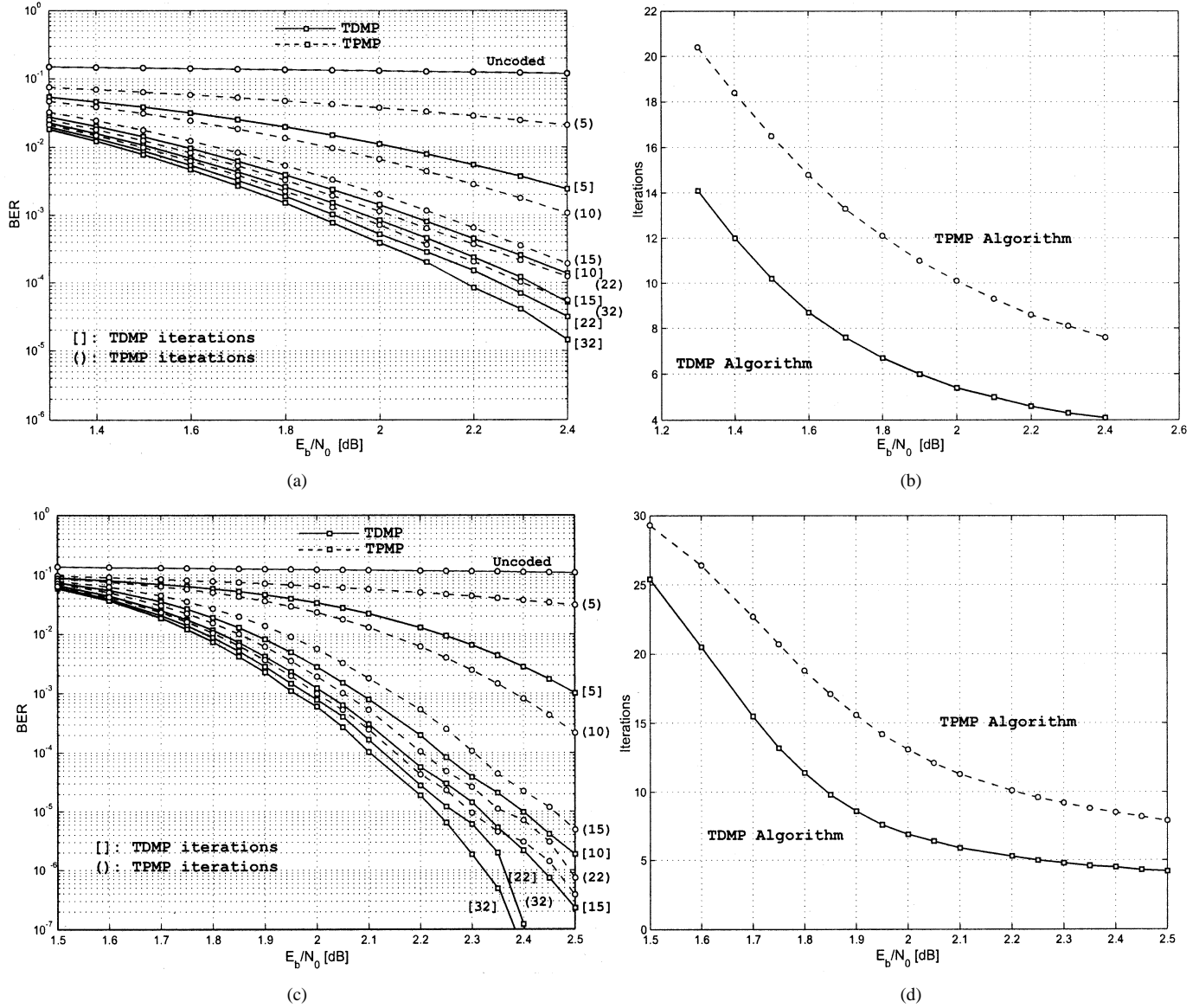


Fig. 13. BER performance of the TDMP algorithm versus the TPMP algorithm for the two codes: (a), (b) C_1 and (c), (d) C_2 .

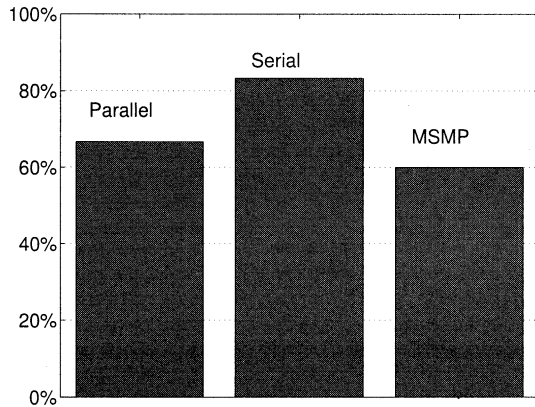


Fig. 14. Memory savings of the TDMP architecture compared to parallel serial and MSMP architectures for regular (3, 6)-LDPC codes of length $n = 2048$ and rate 0.5.

VI. REDUCED-COMPLEXITY MESSAGE COMPUTATIONS

This section presents an algorithmic optimization that addresses the way messages are generated. A reduced-complexity

message update mechanism in the form of a soft-input soft-output message processing unit (MPU) based on the BCJR algorithm [13], [17] is proposed. These MPUs are used in the TDMP decoder architecture to be discussed in the next section, and replace the BFU and CFU blocks of the basic and improved decoder architectures of Fig. 5.

The check-to-bit message update equation (G2) reproduced below,

$$R_{ij}[k] = \psi^{-1} \left[\sum_{j' \neq j' \in R[i]} \psi(|Q_{j'i}[k]|) \right] \cdot \delta_{ij}$$

involves the nonlinear function $\psi(x) = -(1/2) \log(\tanh(x/2))$ and its identical inverse $\psi^{-1}(x)$. Typically, $\psi(x)$ is implemented in hardware using a lookup table (LUT). This update equation has two drawbacks:

- 1) It is prone to quantization noise due to the nonlinearity of $\psi(x)$ and its inverse. In [13], it was shown that $\psi \circ$

$\psi^{-1} \neq 1$ and the dynamic range is limited when $\psi(x)$ quantized. These disadvantages translate to algorithmic performance loss where convergence speed is reduced thereby increasing the decoding latency, switching activity and hence the power consumption of the decoder.

- 2) A typical parallel implementation of (G2) requires $2r$ LUTs, where r is the check node degree. The size of these LUTs becomes significant with increase in precision. Moreover, they tend to increase the minimum clock period since they fall on the critical path, and hence must be pipelined. For large r , a parallel implementation becomes impractical to implement, while serial implementations create a throughput bottleneck.

An alternative approach for computing the check-to-bit messages was proposed in [13] by using a simplified form of BCJR algorithm [17] tailored to the syndrome trellis of an $(r, r-1)$ -SPC code. Related ideas were independently proposed in [6], [18]. The key equations of the algorithm for any section of such trellis reduce to

$$\alpha'_1 = \ln(e^{\alpha_1} + e^{\alpha_2 + \lambda}), \quad \alpha'_2 = \ln(e^{\alpha_1 + \lambda} + e^{\alpha_2}) \quad (1a)$$

$$\beta'_1 = \ln(e^{\beta_1} + e^{\beta_2 + \lambda}), \quad \beta'_2 = \ln(e^{\beta_1 + \lambda} + e^{\beta_2}) \quad (1b)$$

$$\Lambda = \ln(e^{\alpha_1 + \beta_2} + e^{\alpha_2 + \beta_1}) - \ln(e^{\alpha_1 + \beta_1} + e^{\alpha_2 + \beta_2}) \quad (1c)$$

where λ is the *input* prior and intrinsic channel reliability value of the code bit associated with that trellis section, Λ is the updated *output* reliability of that code bit, and α, β are intermediate forward and backward state metrics, respectively. Equations (1a) and (1b) are called the forward and backward state metric recursions, respectively. In the context of LDPC codes, the length of the trellis is the number of bits per check node, or r for regular (c, r) -LDPC codes, and is typically very small (3 to 8) compared to ~ 1 K for turbo codes.

The key equations involve the function $f(x, y) = \ln(e^x + e^y)$ which can be approximated using the Jacobian algorithm [23] as

$$\begin{aligned} f(x, y) &= \max(x, y) + \ln(1 + e^{-|x-y|}) \\ &= \max(x, y) + \delta(x - y) \end{aligned} \quad (2)$$

with $\delta(\cdot)$ regarded as a univariate correction function. This correction function can be approximated by a LUT or a piecewise-linear function [6]. Both approximations are not well suited for high throughput applications since their computation time becomes the dominant part in (2).

We propose a simpler approximation of $\ln(1 + e^{-|x-y|})$ given by

$$\delta(x) \approx \max\left(\frac{5}{8} - \frac{|x|}{4}, 0\right) \quad (3)$$

where the factors are easy to implement in hardware. In fact, using signed-magnitude representation and b quantization bits, $\delta(\cdot)$ can be implemented using $3b$ logic gates, which is even less than a full adder which requires $9b - 5$ gates. Fig. 15 compares the ideal $f(x, y)$ as a function of x for two values of y with the proposed approximation in (3). The approximation almost perfectly matches the ideal $f(x, y)$. Also shown in the figure is the

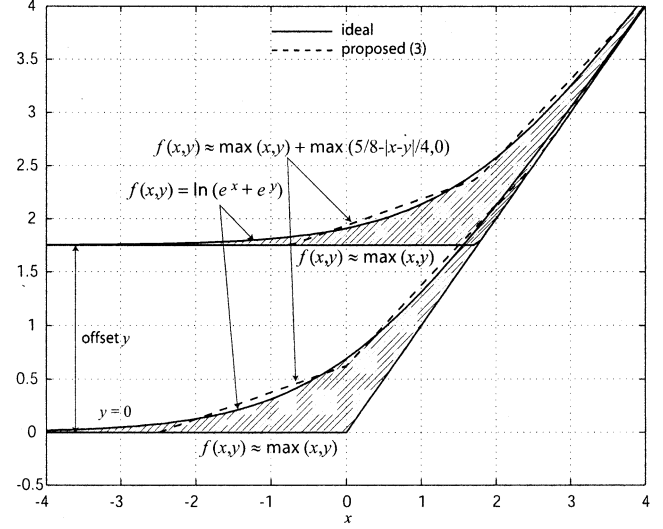


Fig. 15. Approximation of $f(x, y)$ using the correction function in (3). Also shown is the approximation using the principal part only and the error incurred (dashed regions).

approximation without a correction function and the resulting error between the two (hatched regions). Note that this approximation can be applied in turbo decoders as well.

To minimize the effects of quantization noise on the computations, (1a)–(1c) can be transformed so that metric differences $\Delta\alpha \triangleq \alpha_2 - \alpha_1$ and $\Delta\beta \triangleq \beta_2 - \beta_1$ rather than absolute metrics are involved. This transformation was independently proposed in [13] and [18]. Such a transformation maximizes the effectiveness of the limited dynamic range of the metrics and eliminates the need for normalization to avoid overflow. Moreover, only one set of forward and backward state metrics, instead of two, needs to be processed. The transformed key equation (1a) becomes

$$\Delta\alpha' = \ln(e^{\Delta\alpha} + e^\lambda) - \ln(1 + e^{\Delta\alpha + \lambda}) = q(\Delta\alpha, \lambda)$$

where

$$q(x, y) \triangleq f(x, y) - f(x + y, 0) = q_0(x, y) + \delta_q(x, y) \quad (4)$$

and $q_0(x, y) = \max(x, y) - \max(x + y, 0)$ is the *principal* part of the approximation, while $\delta_q(x, y)$ is a bivariate correction function. A similar argument holds for (1b) and (1c). The simplest approximation of $q(x, y)$ is to use the principal part $q_0(x, y)$. Although this approximation is very simple to implement, it incurs a loss of more than 1 dB in coding gain. Another method is to approximate the correction function $\delta_q(x, y)$ by a 2-D LUT indexed by $x - y$ and $x + y$, or use two copies of a piecewise linearized univariate correction function $\delta(x)$. However, the overhead of these approximations in terms of delay and area becomes substantial for large r .

We propose a simpler (in terms of hardware implementation) and more accurate approximation of $\delta_q(x, y)$ based on (3) given by

$$\delta_Q(x, y) = \max\left(\frac{5}{8} - \frac{|x-y|}{4}, 0\right) - \max\left(\frac{5}{8} - \frac{|x+y|}{4}, 0\right) \quad (5)$$

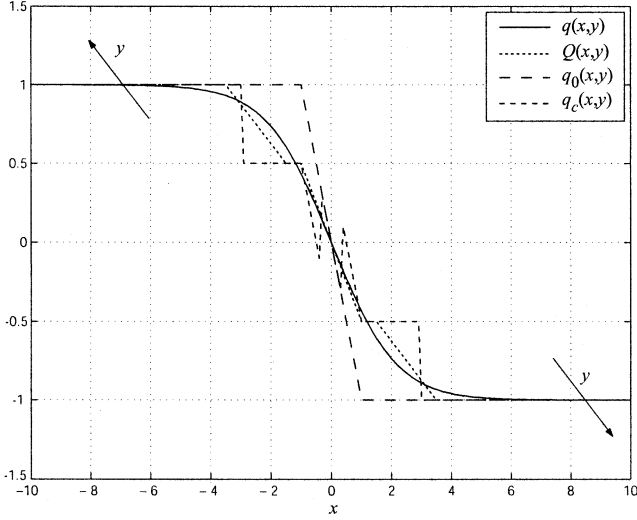


Fig. 16. Approximation of $q(x, y)$ using $q_0(x, y)$, $q_c(x, y)$, and $Q(x, y)$.

with the resulting approximation of $q(x, y)$ given by the “max-quartet” function

$$Q(x, y) = \max(x, y) - \max(x + y, 0) + \max\left(\frac{5}{8} - \frac{|x - y|}{4}, 0\right) - \max\left(\frac{5}{8} - \frac{|x + y|}{4}, 0\right). \quad (6)$$

In [6] and [18], $\delta_q(x, y)$ was approximated using the function

$$\delta_c(x, y) = \begin{cases} c_{\text{SNR}}, & \text{if } |x - y| < 2 \text{ and } |x + y| > 2|x - y| \\ -c_{\text{SNR}}, & \text{if } |x + y| < 2 \text{ and } |x - y| > 2|x + y| \\ 0, & \text{otherwise,} \end{cases} \quad (7)$$

where c_{SNR} is a constant that depends on the channel SNR. However, there is no constructive method to compute this constant. Moreover, this approximation requires routing the constant to all relevant logic units on chip, which in a parallel implementation amounts to adding wires more than twice the number of edges in the bipartite graph defining the code¹.

Fig. 16 compares the ideal function $q(x, y)$ and the proposed $Q(x, y)$, together with the no-correction case $q(x, y) \approx q_0(x, y)$, and the constant-correction case $q_c(x, y) = q_0(x, y) + \delta_c(x, y)$ of [6, 18]. The plots shown in the figure are a function of x with y regarded as a “stretching factor.” The figure demonstrates that the proposed approximation $Q(x, y)$ matches the ideal $q(x, y)$ best. In terms of the function $Q(x, y)$, the key equations simplify to

$$\Delta\alpha' = Q(\Delta\alpha, \lambda), \quad \Delta\beta' = Q(\Delta\beta, \lambda), \quad \Lambda = Q(\Delta\alpha, \Delta\beta). \quad (8)$$

¹More specifically, for a regular (c, r) -LDPC code having m check nodes, the number of units that require the constant is $m(3r - 1)$ compared to mr edges in the graph.

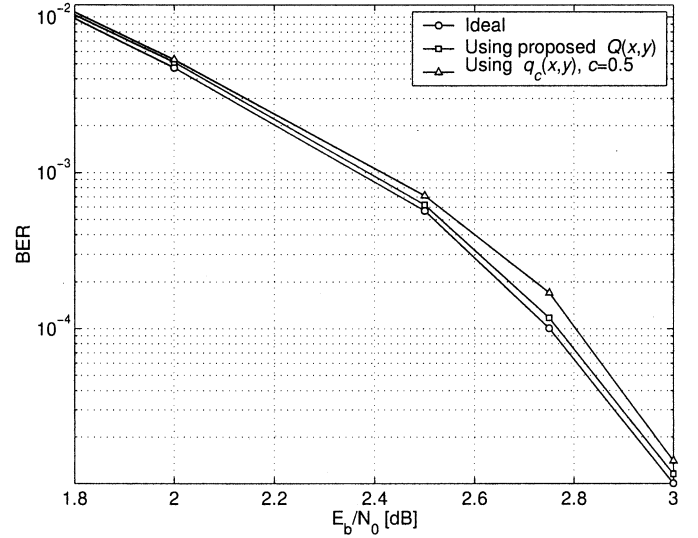


Fig. 17. Comparison of the BER performance achieved using versus the constant approximation $q_c(x, y)$ and the ideal $q(x, y)$.

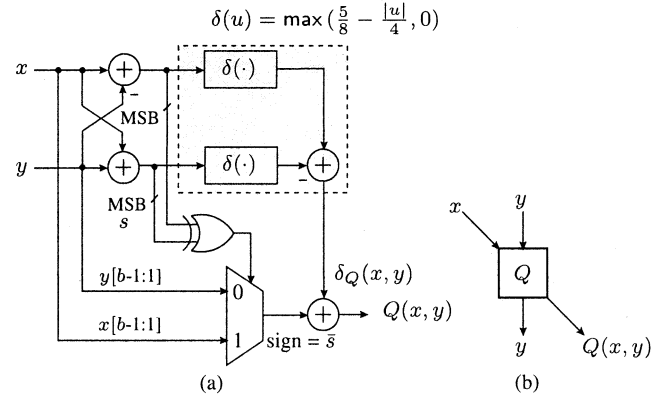


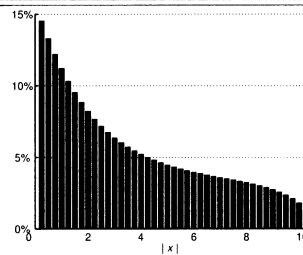
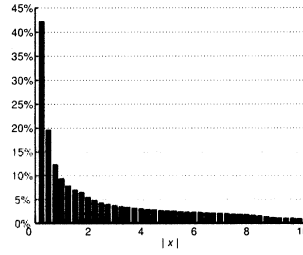
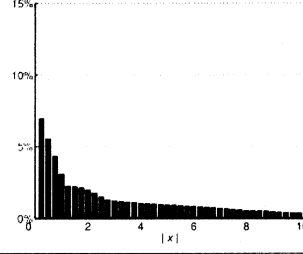
Fig. 18. Max-quartet function $Q(x, y)$: (a) Logic circuit, and (b) symbol.

Fig. 17 compares the BER performance of the TDMP algorithm employing the BCJR key equations using the ideal function $q(x, y)$, the “max-quartet” function $Q(x, y)$, and the constant approximation function $q_c(x, y)$. The figure demonstrates that there is negligible loss in coding gain incurred by the proposed approximation compared to the ideal case.

In terms of hardware, the correction function $\delta_Q(x, y)$ can be implemented using $(15b - 5)$ gates, assuming signed-magnitude arithmetic on b bits, compared to $(21b - 6)$ gates for $\delta_c(x, y)$. Fig. 18 shows a logic circuit implementing the function $Q(x, y)$. Table I compares the number of gates required to implement the function $q(x, y)$ using the above approximations together with the average approximation error incurred. The proposed approximation has a clear advantage over $q_c(x, y)$ both in terms of accuracy and hardware complexity. The error incurred using $q_0(x, y)$ on the other hand outweighs its advantage in terms of hardware simplicity.

Fig. 19 shows a trellis and the corresponding parallel dataflow graph (DFG) aligned below the trellis sections implementing the key equations. Since this DFG accepts soft messages (λ) and generates update soft messages (Λ), we call it a soft-input soft-output message processing unit (MPU) and designate it

TABLE I
NUMBER OF LOGIC GATES AND PERCENTAGE ERROR INCURRED USING THE
FUNCTIONS $q_0(x, y)$, $q_c(x, y)$, $Q(x, y)$ TO APPROXIMATE $q(x, y)$

Function	# Gates	% Error
$q_0(x, y)$	$21b - 11$	
$q_c(x, y)$	$42b - 17$	
$Q(x, y)$	$36b - 16$	

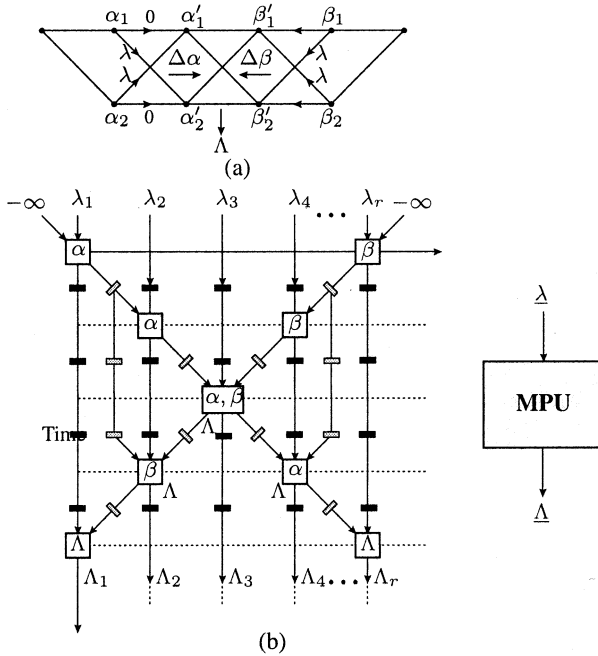


Fig. 19. A SISO MPU based on the BCJR algorithm used for message computation: (a) a two-state trellis and (b) a parallel dataflow graph of the MPU showing the resource utilization of the α and β -recursions with time.

by the symbol shown in the figure. The function units in the DFG employ the “max-quartet” function of Fig. 18 to implement each of the three key equations in (8). The α and β re-

cursions are first performed by traversing the trellis from both ends, saving the intermediate metrics in the buffers designated by the grey-shaded rectangles in the figure. When the middle of the trellis is reached, the α and β units perform the dual operation of generating the output reliabilities Λ and then updating the state metrics. The state metrics are not saved from this point onward. For full throughput operation, skew buffers are provided to align the input and output metrics. Note that this dataflow graph is similar to the parallel-window DFG proposed in [24] in the context of turbo MAP-decoding. The latency of the MPU is proportional to r , where r is typically less than eight. The approach in [6] results in a latency of $\log(r) + 1$, however in order to achieve comparable performance to our approach, it requires the use of one or two LUTs for every computation of $q_c(x, y)$ which tend to increase the clock period by more than a factor of 2 as well as the area of the MPU.

VII. PROGRAMMABLE TDMP DECODER ARCHITECTURE

In this section, we propose a programmable decoder architecture implementing the TDMP algorithm for both regular and irregular AA-LDPC codes that builds upon the optimizations performed in earlier sections. The architecture is shown in Fig. 20, and the underlying parity-check matrix of the code is decomposed as shown in Fig. 10, and has the following parameters: 1) S , the size of the sub-matrix partitions (assuming permutation matrices); 2) $B = n/S$, the number of submatrices per block row; 3) $D = m/S$, the number of submatrices per block column; 4) r , the maximum number of permutation matrices per block row; and 5) c_1, \dots, c_B , the number of permutation matrices in the block columns. The architecture includes B memory modules for storing messages, S MPUs that operate in parallel, and read/write networks for transporting messages between memory and these MPUs. The parameter S acts as a scaling parameter.

The decoder completes a single decoding iteration by performing a round of D updates across the supercodes. An update corresponding to a single supercode \mathcal{C}^j constitutes a subiteration which involves the following steps:

- 1— The read-network performs c read operations from memory, where c is the maximum node degree of \mathcal{C}^j . It then forwards r messages to each of the S MPUs.
- 2— The MPUs update the messages in parallel as described in Section VI.
- 3— The updated messages are routed by the write-network to the appropriate memory modules scheduled to receive messages related to \mathcal{C}^j .
- 4— The messages are written in the designated memory modules and the address counters are updated.

In the remainder of this section, we describe the memory architecture of the decoder, followed by the architectures of the read and write-networks.

A. Memory Architecture

The messages of the decoder are assumed to be stored column-wise in B memory modules such that the i th module stores the messages of the i th block column of the parity-check matrix. The memory modules store messages column-wise in S

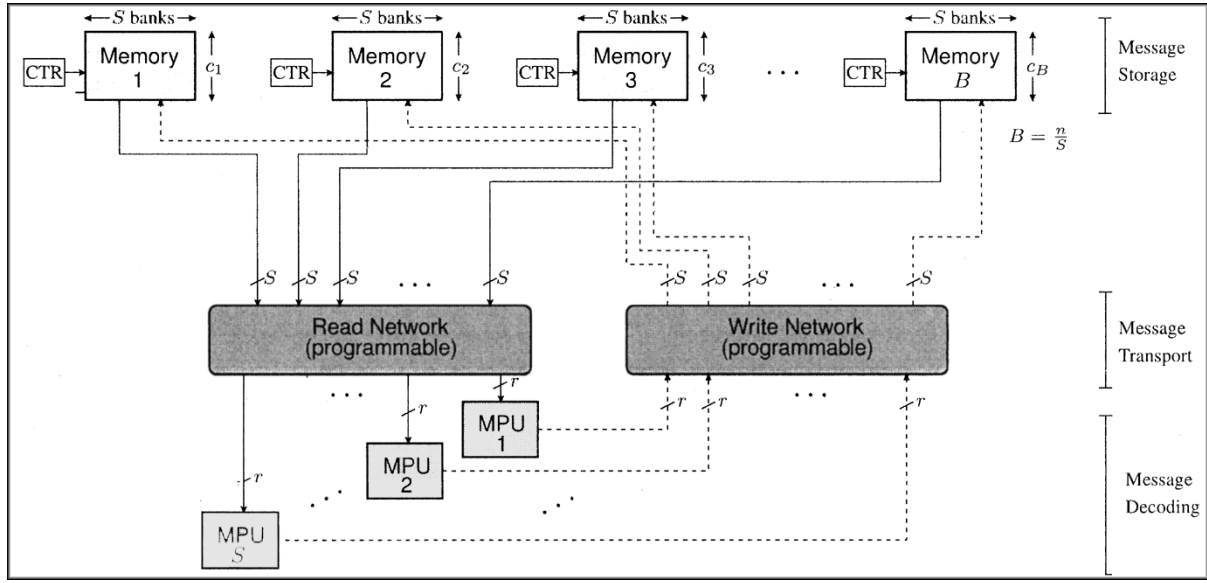


Fig. 20. Decoder architecture implementing the TDMP algorithm. The last row in the banks of the memory modules store the outputs of the decoder.

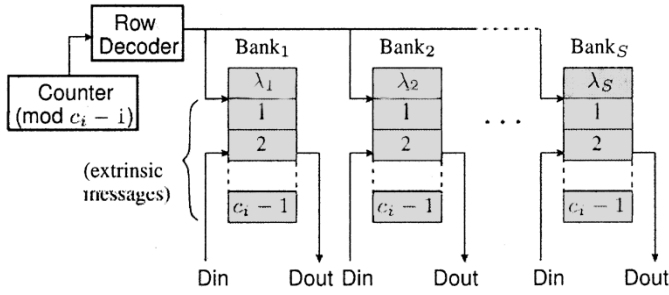


Fig. 21. Architecture of a memory module.

memory banks as shown in Fig. 21. The first row of each bank in the i th module always stores the intrinsic channel reliability messages, while the remaining rows store $(c_i - 1)$ messages extrinsic with respect to the first target supercode scheduled to utilize these messages. The extrinsic messages generated earlier by this supercode are not stored since they will not be used by the decoder operating on the target supercode to generate new messages. The contents of the banks in each module are consumed sequentially from top to bottom. The banks however are updated row-wise in a circular fashion using a counter that counts modulo $(c_i - 1)$ and keeps track of the next row scheduled to receive messages from the target supercode being decoded. The use of pointers to update memory constitutes an improvement targeted for restricting data movement (and hence reducing power consumption) over our earlier work in [5] and [21] which employed rotating buffers that physically rotate the data itself rather than a pointer.

The last row in the banks of the memory modules at the final iteration contains the output messages of the decoder. These messages can either be extrinsic messages suitable for an outer joint iterative equalization and decoding scheme, or posterior messages otherwise. For the latter case, the MPUs operating on the last supercode at the final iteration need to add the prior reliabilities to their outputs for making hard decisions. Table II

shows a snapshot of the memory contents across four decoding sub-iterations (or one iteration) to update the four supercodes of the LDPC code defined by the parity-check matrix in Fig. 10. For simplicity, the contents of the memory modules are labeled with the supercode number, and the stars in the third column denote the most recently updated messages. The horizontal arrows in the middle column point to the locations to be written, while the third column shows the updated positions of the pointers after the write operation.

B. Read-Network Architecture

This section presents a dynamic network architecture for transporting messages from memory to the constituent SISO MPUs according to the TDMP algorithm in Section V-A. Fig. 22 shows the architecture of the read-network on the left side, and the right side shows the sequence and order in which the messages are read and processed at different stages of the network. The candidate messages (corresponding to multiple supercodes) to be read are shown as contents of the memory modules directly above the network. The structure of the target supercode that will receive these messages is shown in the lower right part of the figure. The read-network performs the following functions:

- a) **Block selection:** The appropriate messages of the source supercodes are read from memory using a row of r block selectors. The i th block selector selects the i th memory module (permutation matrix) out of the J possible blocks as shown in Fig. 23 using a set of S ($J : 1$)-multiplexers, where $J < B - r \ll n$, controlled by a single sequencer (X-SEQ) that keeps track of the position of the i th permutation matrix (if present) in each of the D block rows. Table III summarizes the hardware complexity of the logic units of the network. The four blocks of the example supercode are selected as shown next to the row of r block selectors in Fig. 22.

TABLE II
CONTENTS OF THE MEMORY MODULES AS THEY ARE ACCESSED DURING THE FOUR SUB-ITERATIONS TO DECODE THE LDPC CODE OF FIG. 10

Subiteration #	Read Action	Write Action
1 (super-code C^1)		
2 (super-code C^2)		
3 (super-code C^3)		
4 (super-code C^4)		

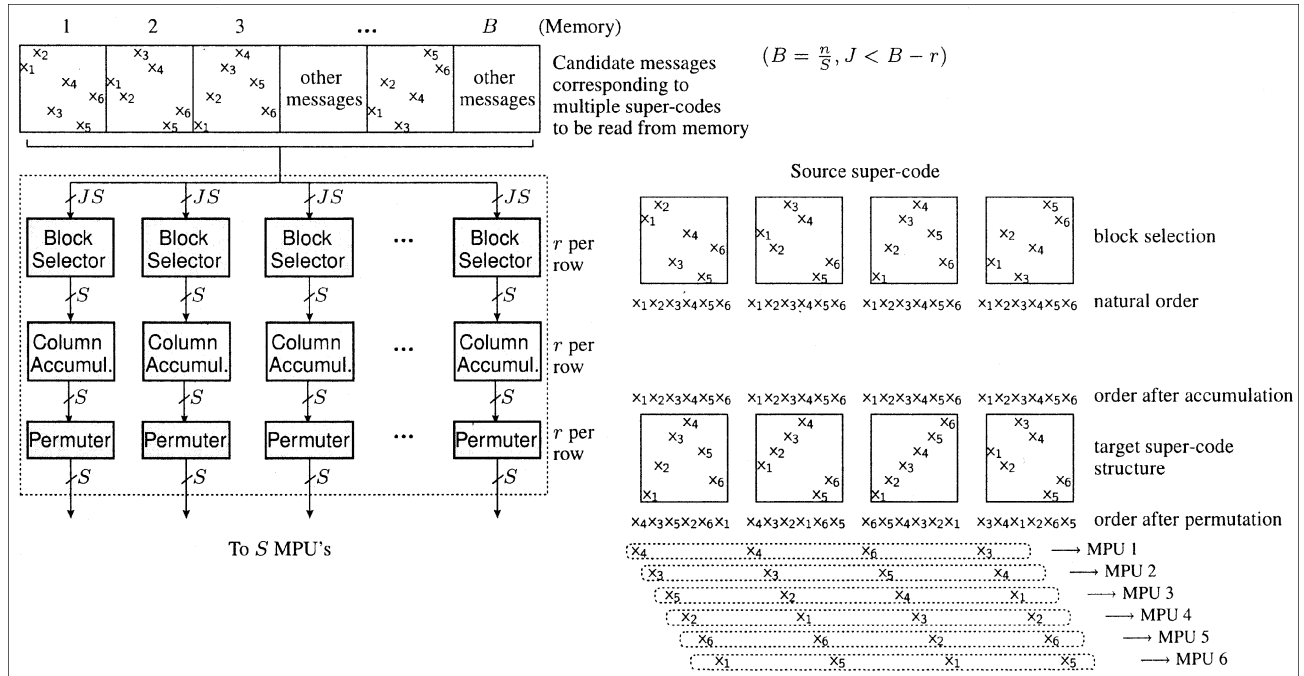


Fig. 22. Read-network architecture of the TDMP algorithm. Candidate messages corresponding to multiple supercodes to be read are shown in the memory modules above the network. The right part of the figure shows the action of the network on these messages as they get routed to the decoders.

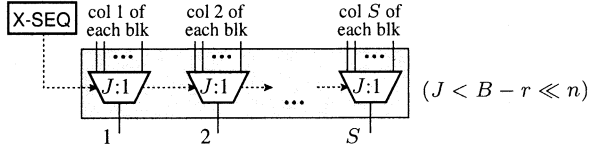


Fig. 23. Architecture of a block selector.

TABLE III
MESSAGE-TRANSPORT NETWORK LOGIC AND CONTROL RESOURCES

Block Name	Logic		Control	
	Type	Count	States	Sel. bits
Block Selector	$J:1$ MUX	S	$\lg(D)$	$\lg(J)$
Column Accum.	ADDER	S	—	—
Permuter	$S:1$ MUX	S	$S \lg(D)$	$S \lg(S)$
Block Relocator	$K:1$ MUX	S	$\lg(D)$	$\lg(K)$

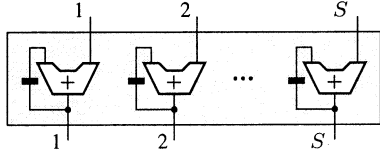


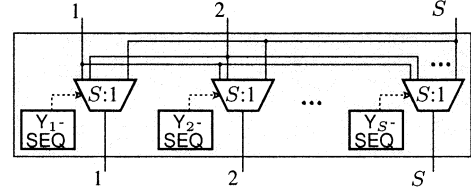
Fig. 24. Architecture of a column accumulator.

- b) **Column accumulation:** The S messages from each block selector are accumulated on top of earlier messages read from previous supercodes (including the intrinsic channel reliabilities of the bits) using a row of r column accumulators. Fig. 24 shows the architecture of a column accumulator composed of S full adders. This operation does not change the order of the messages as shown in the row of messages next to the row of adders in Fig. 22.
- c) **Message permutation:** The messages from the r blocks are independently permuted to match the structure of the permutation matrices of the target supercode using a row of r permeters. A permuter is an S -to- S mapper that can be implemented using a set of S ($S:1$)-multiplexers as shown in Fig. 25. The multiplexers are controlled independently by S sequencers, where the j th sequencer of the i th permuter keeps track of the position of the j th message of the i th permutation matrix in all D block rows. The permuter reduces to a simple shift register in the case of cyclic permutation matrices. The order of the messages after permutation according to the target code is shown next to the row of permeters in Fig. 22. Finally, these messages are forwarded to the S MPUs such that the k th MPU receives the k th message from every block as shown in the lower right corner of the figure.

The resources of the read-network are summarized in Table III. The overall complexity of the read-network in terms of standard library gates is given by

$$\mathcal{O}_{\text{Read}} = r \times S \times (\text{MUX}_J + \text{FA} + \text{MUX}_S) \\ \approx br(n - rS + S^2)$$

where $\text{MUX}_J = b \cdot (J - 1)$ 1-bit multiplexers (similarly for MUX_S), and $\text{FA} = 9b - 5$ gates, with $J < B - r \ll n$. Note that

Fig. 25. Architecture of an S -to- S permuter.

multiplexers are very cheap to implement in current VLSI technology using transmission gates or pass-transistor logic. The sequencers of the read-network can be programmed to implement various instances of a parity-check matrix of an AA-LDPC code for given parameters n , S , B , and D . This feature makes the architecture a programmable architecture which is attractive in applications where a single decoder is used to decode multiple AA-LDPC codes of the same parameters n , S , B , and D , but with different permutation matrices.

C. Write-Network Architecture

Fig. 26 shows the write-network architecture which performs the reverse operations of the read-network excluding column accumulation. Briefly, it performs the following functions:

- a) **Message permutation:** The updated messages from the MPUs are restored back to their natural order by inverse permuting them using a row of r permeters such as the ones shown in Fig. 25. The restored order of the messages is shown next to the row permeters in Fig. 26.
- b) **Block relocation:** The messages corresponding to the target supercode are written back to memory using a row of B block relocators. A block locator as shown in Fig. 27 writes the appropriate block to the i th memory module if the i th sub-matrix of the target supercode is not the zero matrix, and remains idle otherwise. The lower right part of Fig. 26 shows the result of the block relocation operation.

The overall complexity of the write-network is given by

$$\mathcal{O}_{\text{Write}} = r \times S \times (\text{MUX}_S + n \times \text{MUX}_K) \\ \approx b((r - 1)n + r(S^2 - S))$$

where $K \leq r$. Like the read-network, the write-network is also programmable.

VIII. SIMULATION RESULTS

To demonstrate the effectiveness of the proposed TDMP algorithm coupled with the reduced-complexity MPUs and the dynamic message-transport networks, a decoder architecture implementing these ideas was synthesized. Power and area characterizations were done using the Synopsys tool suite based on a 0.18- μm , 1.8-V standard cell library. During power characterization, a frame consisting of random noise was assumed as input to the decoder. The datapath of the decoder is five bits wide. The target LDPC code is a rate-2/3 irregular LDPC code of length 2304 constructed from Ramanujan graphs [15]. The edge merging transformation [21] was used to obtain the factor graph and the parity-check matrix of the code. The

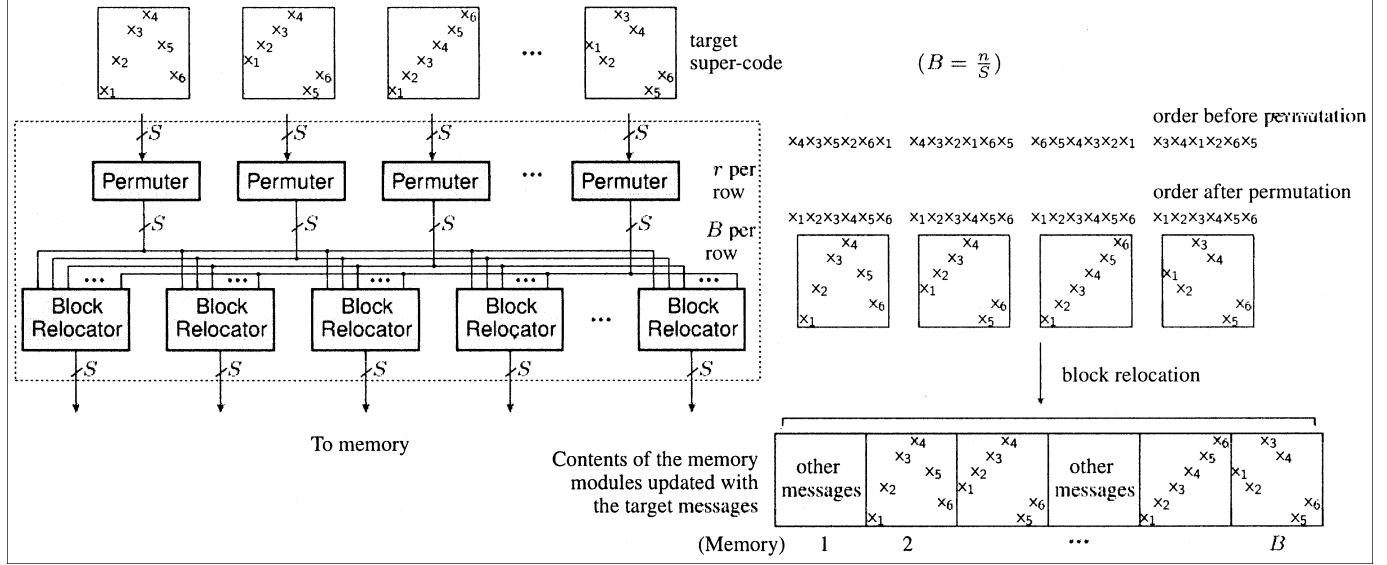


Fig. 26. Write-network architecture of the TDMP algorithm. Messages from the target supercode to be written in memory are shown above the network. The action of the network on these messages and the resulting contents of the memory modules are shown on the right part of the figure.

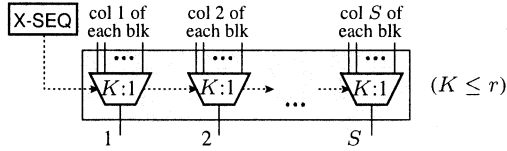


Fig. 27. Architecture of a block relocater.

resulting permutation matrices of the parity-check matrix are acyclic matrices of size $S = 64$.

At a clock rate of 200 MHz, the maximum throughput of the decoder is 1.92 Gbits/s, which corresponds to an iterative decoder throughput of 192 Mbits/s assuming ten decoding iterations are performed per frame. The average power consumption is 1,176 mW, and the decoder area is 9.41 mm². The high throughput is mainly attributed to the fast convergence behavior of the TDMP algorithm which requires between 20%–50% less iterations to converge compared to the TPMP algorithm. The distributed memory architecture which allows for parallel access of messages, and the message-transport networks which keep the follow of these messages into and out of the MPUs uninterrupted (e.g., due to resource conflicts and scheduling as is the case in [12]) also contribute to the high throughput of the decoder.

A. Quantization Effects

To compare the effects of quantization on the algorithmic performance of the TDMP algorithm using Gallager's equations and the proposed MPU based on the "max-quartet" approximation, a gate-level simulator was developed for both techniques, and the results are shown in Fig. 28. In [13], a similar experiment was performed considering the TPMP algorithm instead. Fig. 28(a), (d) show the percentage of valid frames decoded and the switching activity of the decoder (the number of iterations required for convergence), respectively, using unquantized Gallager equations, 6-bit quantized Gallager equations, and 6-bit quantized MPUs using the "max-quartet" approximation in (6).

Fig. 28(b), (e) and (c), (f) show the results for 5-bit and 4-bit quantization, respectively. The results demonstrate that the optimized MPU is superior to the Gallager's equations particularly for 4-bit quantization where it attains more than 1 dB of improvement in coding gain. Moreover, 5-bit quantized MPUs achieve even better performance than 6-bit quantized Gallager equations. The average improvement in coding gain achieved at 6-bit, 5-bit, and 4-bit quantization levels is 6.02%, 12.19%, and 120.92%, respectively. Note also the reduction in switching activity due to the decrease in the number iterations.

B. Power and Area Distribution

Fig. 29(a) shows the average power distribution profile as a function of the supercodes in each of the main blocks of the decoder. The vertical bars show (starting from the bottom) the power consumed by memory due to reads, the readnetwork, the MPUs, the write-network, and memory due to writes. The bimodal nature of the profile is due to the structure of the parity-check matrix and the edge-merging transformation (see [21] for details). The power consumption of the decoder is 1,176 mW. As expected, memory reads consume most of the power (50.42%) followed by the MPUs (20.73%). The remaining blocks of the architecture consume 28.85% of the power.

Fig. 29(b) shows the area distribution of the decoder among the read and write-networks, memory modules, and MPUs. The area of the decoder is 9.41 mm². The networks (including the peripheral interconnects) occupy 5.17 mm² or 54.94% of the area of the decoder. The optimized MPUs occupy a small portion of the area, 1.73 mm² (18.38%). More importantly, the reduced memory requirement of the TDMP algorithm is evident in that memory occupies only 2.51 mm² or 26.67% of the area.

C. Comparison with a TPMP Decoder Architecture

The TDMP decoder architecture was also compared with a decoder architecture implementing the TPMP algorithm and Gallager's update equations such as [8]. The TDMP decoder

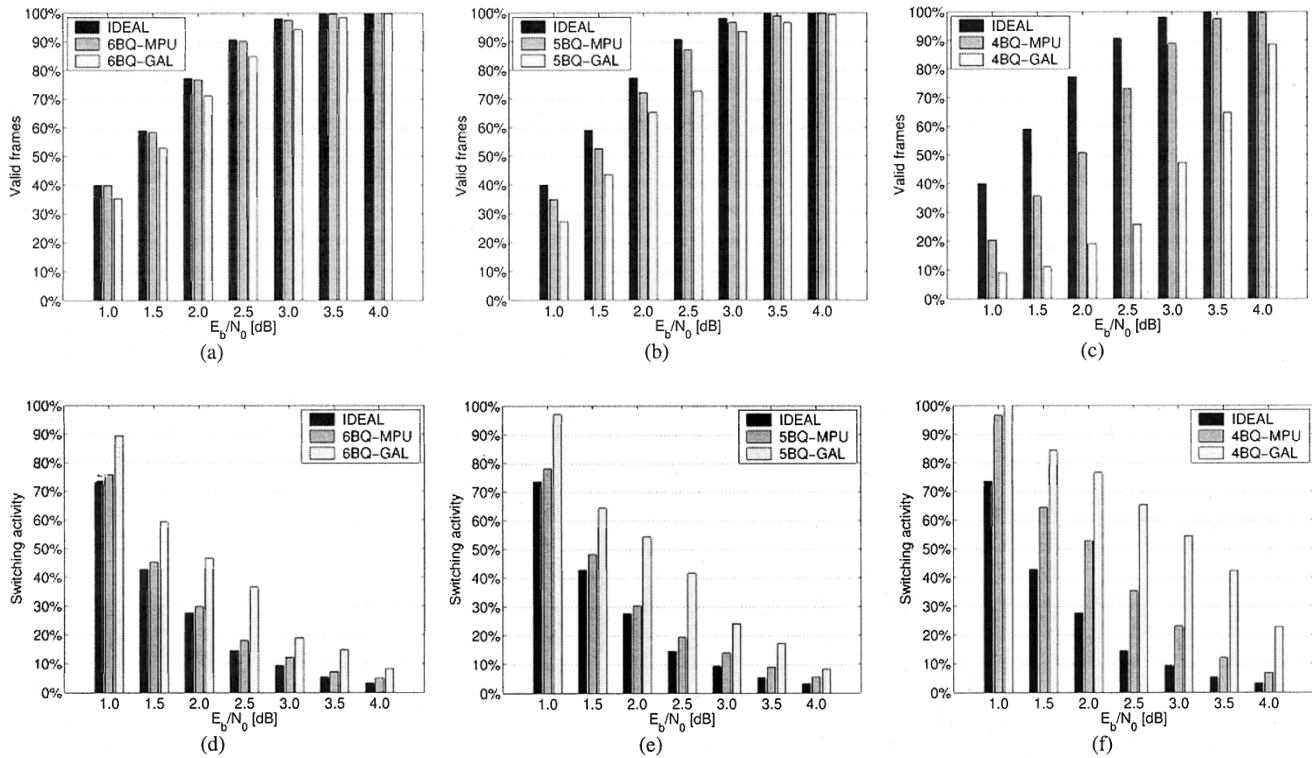


Fig. 28. Comparison of quantization effects on the performance of Gallager's update equations versus the SISO MPU based on the BCJR algorithm for: (a), (d) 6-bit, (b), (e) 5-bit, and (c), (f) 4-bit quantization levels using the TDMP algorithm.

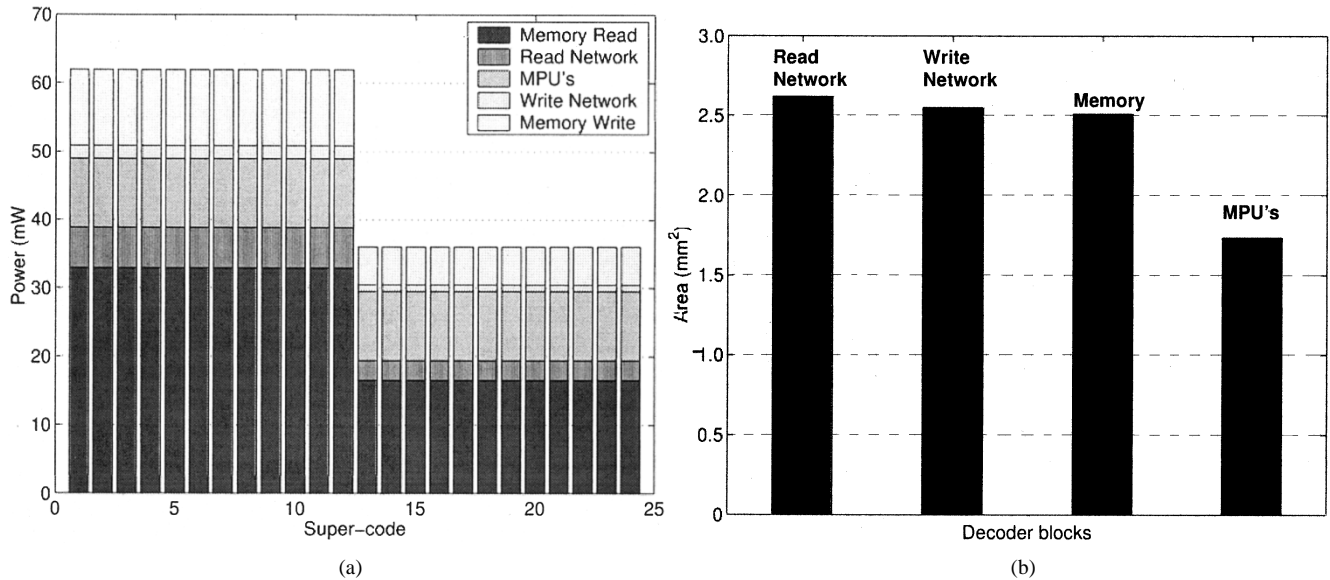


Fig. 29. (a) Average power dissipation in the main blocks of the decoder as a function of the super-codes and (b) area of the decoder as distributed amongst the read and write-networks, memory modules, and MPUs.

architecture was designed to deliver the same throughput as the TDMP architecture using also a five bit datapath. Fig. 30(a) compares the power consumed by the transport networks, MPUs (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, MPUs, and memory modules, respectively. The overall savings in power consumption is 89.13%. In terms of area, Fig. 30(b) similarly shows significant reduction in the

area of the networks (61.98%), MPUs (63.28%), and memory modules (80.52%), respectively. The overall area savings is 69.83%. Finally, the TDMP decoder requires 60.5% less interconnect wires (including both networks and peripherals) than the TPMP decoder.

IX. CONCLUSIONS

The design of high-throughput and memory efficient decoder architectures for regular and irregular LDPC codes has

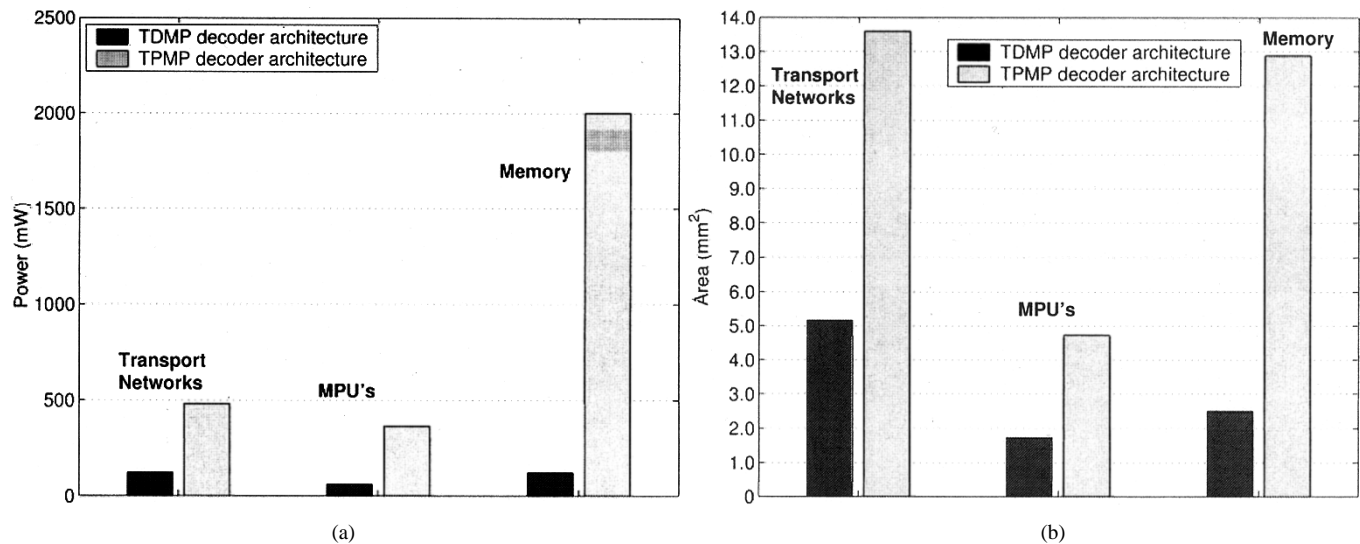


Fig. 30. Comparison of (a) power consumption and (b) area occupied by the transport networks, MPUs, and memory modules of the TDMP architecture versus the TPMP architecture.

been considered at the code-design, decoding algorithm, and architectural levels. Optimizations at the code-design level aim at decoupling the decoder architecture from the code properties by decomposing the parity-check matrix of the code into permutation matrices resulting in architecture-aware LDPC codes. Reducing memory requirements and improving decoder throughput have been addressed algorithmically through a novel turbo decoding algorithm of LDPC codes. Moreover, an efficient message update mechanism has been presented in the form of a message processing unit that reduces the switching activity of the decoder and requires fewer quantization bits than other methods in the literature. Finally, a scalable memory architecture and message-transport networks have been proposed that constitute the main building blocks of a generic LDPC decoder architecture.

Simulations have demonstrated the effectiveness of the methodology pursued for efficient decoder designs in terms of power consumption, area, and interconnect complexity.

REFERENCES

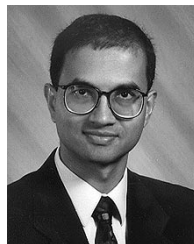
- [1] C. Berron, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error correcting coding and decoding: Turbo codes," in *IEEE Int. Conf. on Communications*, 1993, pp. 1064–1070.
- [2] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
- [3] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.
- [4] S.-Y. Chung *et al.*, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limits," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, Feb. 2001.
- [5] M. M. Mansour and N. R. Shanbhag, "Turbo decoder architectures for low-density parity-check codes," in *Proc. IEEE GLOBECOM*, Taipei, Taiwan, R.O.C., Nov. 2002, pp. 1383–1388.
- [6] X. Y. Hu *et al.*, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *Proc. IEEE GLOBECOM*, vol. 2, San Antonio, TX, 2001, pp. 1036–1036E.
- [7] C. Howland and A. Blanksby, "Parallel decoding architectures for low density parity check codes," in *Proc. 2001 IEEE Int. Symp. on Circuits and Systems*, Sydney, Australia, May 2001, pp. 742–745.
- [8] E. Yeo *et al.*, "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. Magn.*, vol. 37, no. 2, pp. 748–755, Mar. 2001.
- [9] E. Yeo *et al.*, "High throughput low-density parity-check decoder architectures," in *Proc. IEEE GLOBECOM*, San Antonio, TX, Nov. 2001, pp. 3019–3024.
- [10] G. Al-Rawi and J. Cioffi, "A highly efficient domain-programmable parallel architecture for LDPC decoding," in *Proc. Int. Conf. on Information Technology: Coding and Computing*, Apr. 2001, pp. 569–577.
- [11] *Vector-LDPC™ Core Solutions*, Flarion Technologies, Feb. 2002.
- [12] T. Zhang and K. K. Parhi, "VLSI implementation-oriented $(3, k)$ -regular low-density parity-check codes," in *Proc. IEEE Workshop on Signal Processing Systems (SIPS)*, Antwerp, Belgium, Sept. 2001, pp. 25–36.
- [13] M. M. Mansour and N. R. Shanbhag, "Low-power VLSI decoder architectures for LDPC codes," in *Proc. Int. Symp. on Low Power Electronics and Design (ISLPED)*, Monterey, CA, Aug. 2002, pp. 284–289.
- [14] J. Rosenthal and P. O. Vontobel, "Constructions of LDPC codes using Ramanujan graphs and ideas from Margolis," in *Proc. 38th Allerton Conf. on Communication, Control, and Computing*, 2000, pp. 248–257.
- [15] M. M. Mansour and N. R. Shanbhag, "Construction of LDPC codes from Ramanujan graphs," presented at the 36th Annu. Conf. on Information Sciences and Systems 2002 (CISS'02), Princeton, NJ, Mar. 2002.
- [16] J. Boutros, O. Pothier, and G. Zemor, "Generalized low density (Tanner) codes," in *Proc. ICC'99*, Vancouver, Canada, June 1999, pp. 441–445.
- [17] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.
- [18] E. Eleftheriou, T. Mittelholzer, and A. Dholakia, "Reduced-complexity decoding for low-density parity-check codes," *Electron. Lett.*, vol. 37, pp. 102–104, Jan. 2001.
- [19] M. Lentmaier and K. S. Zigangirov, "Iterative decoding of generalized low-density parity-check codes," in *Proc. ISIT 1998*, Aug. 1998, p. 149.
- [20] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, pp. 498–519, Feb. 2001.
- [21] M. M. Mansour and N. R. Shanbhag, "Memory-efficient turbo decoder architectures for LDPC coding," presented at the IEEE Workshop on Signal Processing Systems (SiPS'02), Oct. 2002, pp. 159–164.
- [22] M. M. Mansour and N. R. Shanbhag, "On the architecture-aware structure of LDPC codes from generalized Ramanujan graphs and their decoder architectures," in *Proc. 37th Annu. Conf. Information Sciences and Systems (CISS'03)*, Baltimore, MD, Mar. 2003, pp. 215–220.
- [23] P. Robertson, E. Vilebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc. IEEE Int. Conf. Communications*, 1995, pp. 1009–1013.
- [24] H. Dawid and H. Meyr, "Real-time algorithms and VLSI architectures for soft output MAP convolutional decoding," in *Proc. Personal, Indoor, and Mobile Radio Communications (PIMRC). Wireless: Merging onto the Information Superhighway*, vol. 1, 1995, pp. 193–197.



Mohammad M. Mansour (S'98–M'03) received the B.E. degree with distinction in 1996 and the M.S. degree in 1998, all in computer and communications engineering, from the American University of Beirut (AUB), Beirut, Lebanon. He received the M.S. degree in mathematics in August 2002, and the Ph.D. degree in electrical engineering in May 2003, from the University of Illinois at Urbana-Champaign (UIUC).

Currently, he is an Assistant Professor of Electrical Engineering with the Electrical and Computer Engineering Department of AUB. During the summer of 2003, he was a Postdoctoral Research Associate at the Coordinated Science Laboratory (CSL), UIUC. From 1998 to 2003, he was a research assistant at CSL. In 1997, he was a research assistant at the ECE department of AUB, and in 1996 he was as a teaching assistant with the same department. His research interests are VLSI architectures and integrated circuit (IC) design for communications and coding theory applications, digital signal processing systems, and general purpose computing systems.

Dr. Mansour received the Harriri Foundation award twice, in 1996 and 1998, the Charli S. Korban award twice, in 1996 and 1998, the Makhzoumi Foundation Award in 1998, and the Phi Kappa Phi awards twice, in 2000 and 2001.



Naresh R. Shanbhag (M'88–SM'00) received the B. Tech. degree from the Indian Institute of Technology, New Delhi, India, in 1988, the M.S. degree from Wright State University, Dayton, OH, in 1990, and the Ph.D. degree from the University of Minnesota, Minneapolis, all in electrical engineering in 1993.

From July 1993 to August 1995, he was with AT&T Bell Laboratories at Murray Hill, NJ, where he was responsible for the development of VLSI algorithms, architectures, and implementation of broadband data communications transceivers. In particular, he was the lead chip architect for AT&T's 51.84 Mb/s transceiver chips over twisted-pair wiring for Asynchronous Transfer Mode (ATM)-LAN and broadband access chip-sets. Since August 1995, he has been with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign (UIUC), where he is presently an Associate Professor and the Director of the Illinois Center for Integrated Microsystems. He founded the VLSI Information Processing Systems (ViPS) Group, whose charter is to explore issues related to low-power, high-performance, and reliable integrated circuit implementations of broadband communications and digital signal processing systems at UIUC. He has published numerous journal articles/book chapters/conference publications in this area and holds three U.S. patents. He is also a coauthor of the research monograph *Pipelined Adaptive Digital Filters* (Norwell, MA: Kluwer, 1994).

Dr. Shanbhag received the the 2001 IEEE Transactions on VLSI Systems Best Paper Award, the 1999 IEEE Leon K. Kirchmayer Best Paper Award, the 1999 Xerox Faculty Award, the National Science Foundation CAREER Award in 1996, and the 1994 Darlington best paper award from the IEEE Circuits and Systems society. From July 1997–2001, he was a Distinguished Lecturer for the IEEE Circuits and Systems Society. From 1997 to 1999, he served as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS: PART II. He is currently the Associate Editor for the IEEE TRANSACTIONS ON VLSI SYSTEMS. He was the Technical Program Chair of 2002 IEEE Workshop on Signal Processing Systems.