High Throughput Low-Density Parity-Check Decoder Architectures

Engling Yeo, Payam Pakzad, Borivoje Nikolić, and Venkat Anantharam
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720-1770.

Abstract—Two decoding schedules and the corresponding serialized architectures for low-density parity-check (LDPC) decoders are presented. They are applied to codes with parity-check matrices generated either randomly or using geometric properties of elements in Galois fields. Both decoding schedules have low computational requirements. The original concurrent decoding schedule has a large storage requirement that is dependent on the total number of edges in the underlying bipartite graph, while a new, staggered decoding schedule which uses an approximation of the belief propagation, has a reduced memory requirement that is dependent only on the number of bits in the block. The performance of these decoding schedules is evaluated through simulations on a magnetic recording channel.

I. Introduction

Sequences of Low Density Parity Check (LDPC) codes have been demonstrated to achieve information rates very close to the Shannon limit when iteratively decoded [1]. Using the message-passing algorithm, LDPC decoders require an order of magnitude less arithmetic computations than equivalent Turbo decoders [2]. However, the implementations of decoders for these codes suffer from complex interconnect or large memory requirements due to the sparse nature of the underlying bipartite graph [2].

A number of recent theoretical publications [3] [4] presented the construction of LDPC codes using finite field geometries. These codes result in reduced implementation complexities for both encoder and decoder, but the decoder continues to suffer the disadvantages caused by the sparseness of the graph.

This paper presents a new staggered decoding schedule that reduces the amount of memory requirement to the size of a code block, independent of the number of edges in the graph. In addition, it shows that the large memory requirement can be attributed to the feature that messages are updated concurrently through each round of decoding, and each edge in the graph represents a unique message.

An LDPC code can be employed in magnetic recording channels using the structure shown in Fig. 1, with an LDPC code serially concatenated with a partial response channel [5]. All systems considered in this paper limit the number of user bits to a sector size of 4096 bits. Targeted throughput rates are above 1Gbps, in line with current trends in high throughput applications.

The arithmetic computation and hardware complexity of the iterative soft decoding algorithm will be reviewed in Section II. In Section III, a new staggered decoding schedule is proposed and compared with the original concurrent decoding schedule. Section IV presents an implementation of the staggered decoding schedule for codes based on finite geometries. Simulation results are presented for these finite field-based codes and for random codes in Section V.

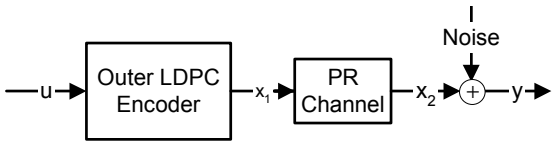


Fig. 1. Serial concatenation of LDPC encoder with a partial response channel.

II. Soft Decoding Of LDPC Codes

A. Message Passing Algorithm

For each received bit, x_i for i=1, 2, …, N, in an N-bit block, an LDPC decoder accepts as input, the log-likelihood ratio (LLR) of probability of possible values for x_i, as defined in (1). LDPC codes are often represented by bipartite graphs made up of two families of nodes, bit nodes and check nodes. Bit nodes represent the transmitted bits in the code. Each bit node is connected to a number of check nodes through an unstructured array of edges. Each check node represents a parity check constraint on the set of adjacent bit nodes.

$$LLR = \ln\left[\frac{\Pr(x_i = 1)}{\Pr(x_i = 0)}\right] \tag{1}$$

LDPC decoders implement the message passing algorithm, which specifies the computation and communication of messages between bit nodes and check nodes as defined by the edges in the graph. An iteration of LDPC decoding consists of a round of message passing from each bit node to all adjacent check nodes, followed by another round of message passing from each check node to all adjacent bit nodes. Optimum decoding performance is achieved through repeated iterations of message passing along edges in the graph.

Let $\boldsymbol{H}$ be the $N \times M$ parity check matrix of an LDPC code comprising N bit nodes and M check nodes. $\nu(m) = \{n: \boldsymbol{H}_{m,n} = 1\}$ is the set of bits that are connected to check m. $\mu(n) = \{m: \boldsymbol{H}_{m,n} = 1\}$ is the set of checks that are connected to bit n. Q_{nm} and R_{mn} refer to the messages that are passed between bit n and check m, as defined according to (2) and (3) respectively.

Message from bit n to check m:

$$Q_{nm} = \ln\left[\frac{p_i(1)}{p_i(0)}\right] + \left(\sum_{m' \in \mu(n)} R_{m'n}\right) - R_{mn} \tag{2}$$

Message from check m to bit n:

$$R_{mn} = \Phi^{-1}\left\{\left(\sum_{n' \in N(m)} \Phi(Q_{n'm})\right) - \Phi(Q_{nm})\right\} \times \left(\operatorname{sgn}(Q_{nm}) \cdot \prod_{n' \in \nu(m)} \operatorname{sgn}(Q_{n'm})\right) \cdot (-1)^{|N(m)|} \tag{3}$$

0-7803-7206-9/01/$17.00 © 2001 IEEE

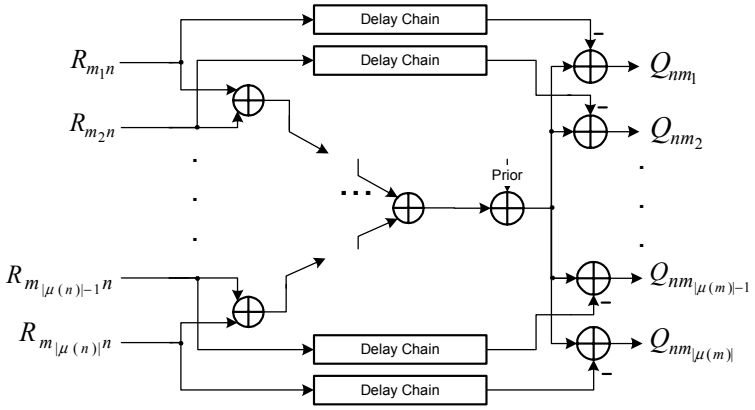


Fig. 2. Bit-to-Check message (Q_{nm}) computation.

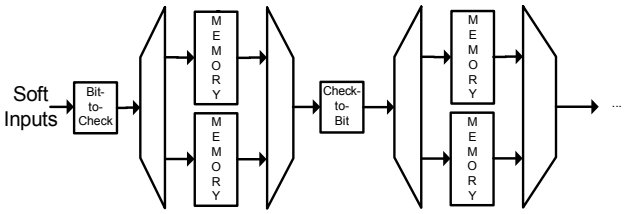


Fig. 3. Serialized and fully pipelined implementation requires two memory buffers per stage, alternating between read/write.

$$\Phi(x) = -\log\left(\tanh(\frac{1}{2}x)\right) = \Phi^{-1}(x); \qquad x \geq 0 \tag{4}$$

The primary objective of an LDPC decoder is to perform the above message computations and to ensure the proper transportation of messages between the two classes of nodes.

Fig. 2 illustrates an example functional block that uses a tree of adders to evaluate (2). The computational complexity required for evaluation of either bit-to-check or check-to-bit messages is low, compared to existing types of decoders such as the Viterbi decoder or Turbo decoders [2]. However, the irregular structure of the underlying bipartite graph leads to interconnect complications with parallel implementation architectures and excessive memory requirement in alternative serial implementation architectures.

B. Parallel Implementation Architecture

The algorithmic description of the message passing algorithm suggests an inherently parallel organization because there is no dependency between computation of either Q_{nm} for n=1, 2, …, N or R_{nm} for m=1, 2, …, M. A parallel decoder architecture directly maps the nodes in the bipartite graph onto message computation units and the edges of the graph onto a network of interconnect. A rate-½, 1024-bit LDPC decoder [6] implemented in 0.16µm technology occupies an area of 7mm×7mm, where logic density is reduced to 50% to accommodate the complexity of the interconnect fabric. This approach leads to higher throughput and lower power dissipation because of the inherent parallelism. It is however, not scalable to codes with larger block sizes. A silicon implementation of a 4096-bit decoder will have 4 times more interconnect wires that are also prohibitively long.

C. Serial Implementation Architecture

Another approach is to serialize and distribute an inherently parallel algorithm amongst a number of processing units [2] [7]. The hardware complexity of the functional units of such a decoder will be significantly lowered, but the implementation still requires a substantial amount of memory to avoid complex interconnect and a large number of algorithmic units. The lack of spatial locality between any subset of bit nodes connected to the same check node (and vice versa) makes it impossible to update the messages for the next round of decoding until a majority of the messages in a particular direction are received. Therefore, a majority of the Q_{nm} and R_{mn} messages need to be stored in a memory whose size is dependent on the total number of edges in the particular code design. A serialized rate 8/9 4608-bit LDPC decoder described in [2] would have to perform memory read and write operations for 37,000 messages in each iteration of decoding.

In addition, serialized implementations are slower and will require hardware pipelining (through segmenting the check-to-bit and bit-to-check stages) in order to sustain high throughput rates. Full utilization of all pipelined stages is only achievable if each section of the message computation is operating on independent blocks of data. This requires two memory buffers per stage, alternating between read/write, as shown in Fig. 3. The amount of memory required for each iteration is therefore four times the total number of edges in the graph, and grows linearly with the number of iterations. Ad-hoc rescheduling [7] of the serialized computations leads to increased efficiency of the computational units but remains largely limited by the communication between memory and functional units.

III. LDPC Decoding Schedules

This section addresses the memory issue common to all serialized architectures by reviewing the original concurrent schedule of the LDPC decoding algorithm, and proposes an algorithm with an alternative schedule, with modifications to the scheduling theorem discussed in [8].

In particular, the notation M and $S(k)$ are used to denote the set of check nodes in the code, and the subset of check nodes that are processed at time step k, respectively.

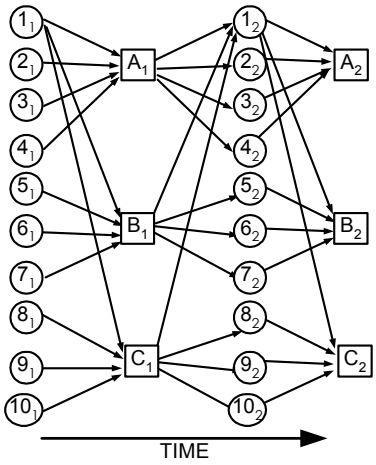


Fig. 4. Concurrent decoding schedule of the message passing algorithm; circles represent bit nodes; boxes represent check nodes.

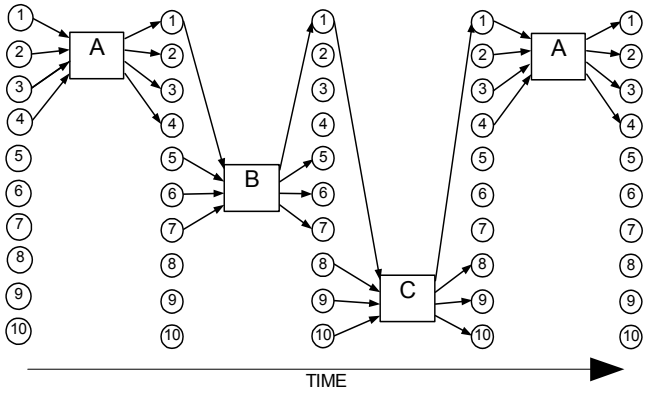


Fig. 5. Staggered decoding schedule of the modified message passing algorithm with $|S(t)|=1$.

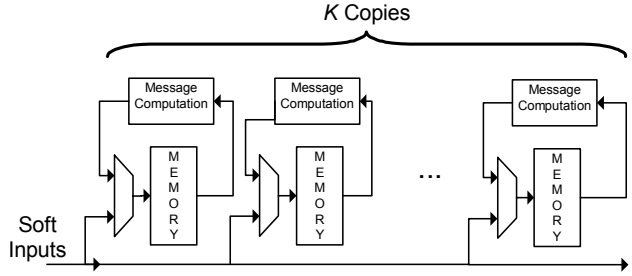


Fig. 6. Architecture for random LDPC decoder with K iterations of staggered schedule.

A. Concurrent Decoding Schedule

The original concurrent decoding schedule, illustrated in Fig. 4 with an example 10 bit nodes and 3 check nodes, shows that all messages in a round of computation are updated and passed between the two classes of nodes concurrently. This is equivalent to $S(k)$ being the set of all check nodes.

In order to compute the marginalized output messages from each node on the bipartite graph, the update equations given by (2) and (3) require that all messages, R_{mn} or Q_{nm}, be available during the message computation. This is the basis of belief propagation [9], and leads to a large memory requirement in order to store all messages in serialized decoder implementation.

B. Staggered Decoding Schedule

An approximation of the update equation is shown in (5). It omits the rightmost term in (2), R_{mn}, and leads to significant reduction of memory requirement for serialized decoder implementations. The effect of this simplification should be minimal when the ratio $R_{mn} / \sum_{m'\in\mu(n)} R_{m'n}$ is small. This is related to the average edge degree of the bit nodes.

$$Q_{nm} \approx \ln\left[\frac{p_i(1)}{p_i(0)}\right] + \left(\sum_{m'\in\mu(n)} R_{m'n}\right) \tag{5}$$

The approximation is combined with a new staggered schedule, which processes only a limited subset of the check nodes. Similar to the original algorithm, parity check node m computes messages R_{mn}, $n \in \nu(m)$, according to (3). However, each bit node, n, is associated only with a single message $Q'_n(k)$, which is broadcasted to the subset of active check nodes, $S(k)$, at step k. Each message, $Q'_n(0)$ for n=1, 2, …, N is initialized with the input LLR of the corresponding bit n, and updated according to (6) at the end of each step.

$$Q'_n(k) = Q'_n(k-1) + \sum_{m'\in\{S(t)\cap M(n)\}} R_{m'n} \tag{6}$$

Fig. 5 illustrates an example schedule that processes only one check node per step, according to the following definition.

$$S(k) = \left\{ m_{k \bmod |M|} \right\} \tag{7}$$

An implementation of the staggered schedule only needs to store one message, $Q_n(k)$, for each bit n. This compares favorably with the concurrent schedule where a list of messages Q_{mn}, $m\in\mu(n)$, need to be stored for each bit node n. For typical LDPC codes where the number of edges in the graph ranges between 4 to 8 times the number of bits in the code-block, this schedule provides more than 75% savings in memory requirement.

Each iteration of the staggered LDPC decoding is defined as one complete cycle through all the sequential parity checks in the graph. This ensures that the number of messages processed in an iteration of decoding is the same in both types of schedules. As such, the decoding arithmetic complexity is similar. Fig. 5 shows the staggered decoding schedule with only one check node active per step.

Fig. 6 illustrates the use of one sector of memory and one constraint check arithmetic unit, per code block processed. A full throughput, K-iteration LDPC decoder, demultiplexes each input block of data into one of the K copies of memory and computation unit pair. This removes the need to move large amounts of data through memory, at a slight cost of control mechanisms to perform the multiplex operations.

TABLE 1.
LDPC CODES CONSIDERED FOR MAGNETIC RECORDING APPLICATIONS

Finite Field Construction	Matrix Dimension	Column Split Factor	PC Matrix Dimension.	Row Wt. × Col Wt.
2D× GF(2^5)	1023 × 1023	4	1023 × 4092	32×8
2D× GF(2^4)	255 × 255	16	255 × 4080	16×1
3D× GF(2^3)	511 × 511	8	511 × 4088	8×1

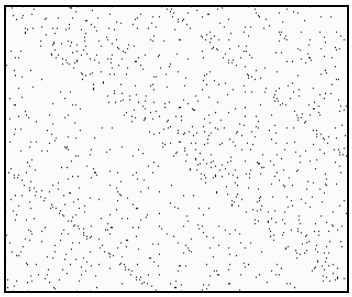


Fig. 7. Rendition of 1023×4092 parity check matrix used in codes based on finite fields. Black dots represent non-zero entries.

IV. Finite Field Based LDPC Decoder

LDPC codes based on finite geometries avoid short cycles of length 4 in the underlying graph [4], which are known to degrade the performance of the code [1]. An example 1023×1023 LDPC matrix was constructed from a 2-dimensional (Euclidean geometry) Galois Field, GF(2^5). Consecutive rows in this parity check matrix are cyclic shifts. In order to reduce its density, each column in the matrix was further split into four to obtain a 1023×4092 parity check matrix, with the non-zero entries in each column in the initial matrix rotated amongst the 4 new columns. This matrix, shown as an image in Fig. 7 with black dots representing non-zero entries, produces a code that comprises 3070 user bits and 1022 parity bits; a code rate of 3/4.

The number of user bits and code rate are less than typical values for magnetic recording applications. A list of the alternative code constructions that may lead to more suitable (higher) code rates is provided in Table 1. It shows that the 2D×GF(2^5) construction results in the only feasible parity check matrix, which does not have a column weight of one.

An implementation of an LDPC decoder using the staggered decoding schedule with the above code is illustrated in Fig. 9. The cyclic property of the rows in the 1023×1023 matrix (before column splitting) is exploited by replacing random access memory with fast shift registers. Since the 1:4 column splitting ensures that no more than 1 bit node in each group of 4 consecutive bit nodes is connected to any parity check node, the running $Q_n(t)$ messages are grouped into clusters of four, forming four parallel shift register chains. This has the advantage that the circuit can be operated at a quarter of the frequency while maintaining the same throughput.

Feedback loops in the shift-register chains enable multiple decoding iterations. Multiplexers provide the appropriate register contents to the Message Computation Block that evaluates (3). Signed-magnitude representation is a natural choice for implementation of the Check-to-Bit computation block because the lookup table functions are sign-invariant, while the XOR operators are magnitude invariant. Fig. 8 shows a 4-bit example implementation of a check-to-bit message computation block, where the most-significant bits are fed into a tree of XOR operators, while the lower 3 bits are fed into a tree of adders.

V. Simulation Results

The empirical results of comparisons performed between the concurrent and staggered decoding schedules with $|S(t)| = 1$ are presented. Two LDPC codes with parity check matrices based on random and 2-dimensional GF(2^5) construction are evaluated. The experiments are performed with a binary antipodal Gaussian channel and LLR inputs.

Performance is evaluated through 1, 3, and 5 iterations of message passing decoding. These iteration counts represent the expected number of iterations that are realizable in serialized decoder implementations, with considerations given to logic density and overall memory requirement.

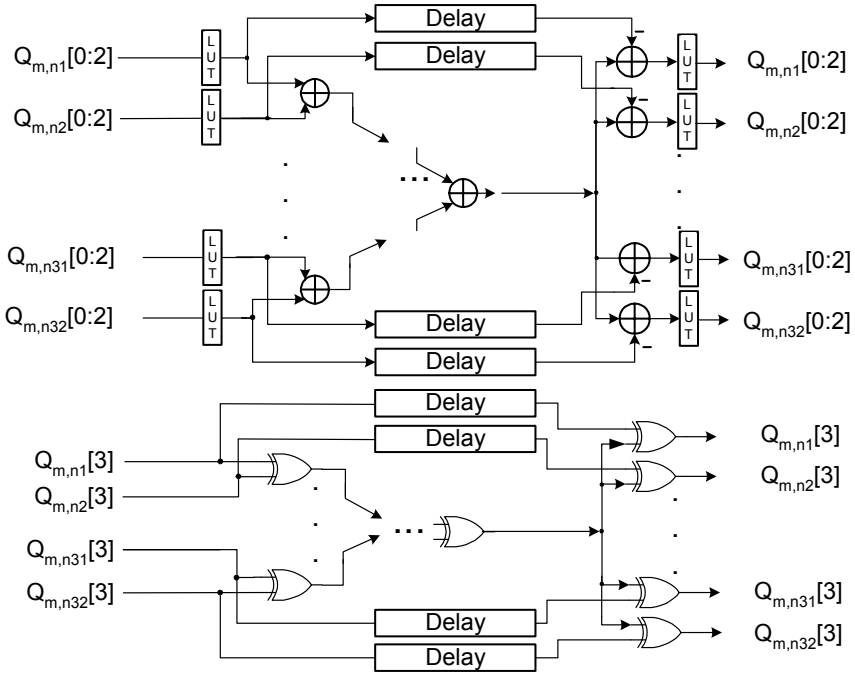


Fig. 8. Check-to-bit message computation block for staggered decoding with finite field codes.

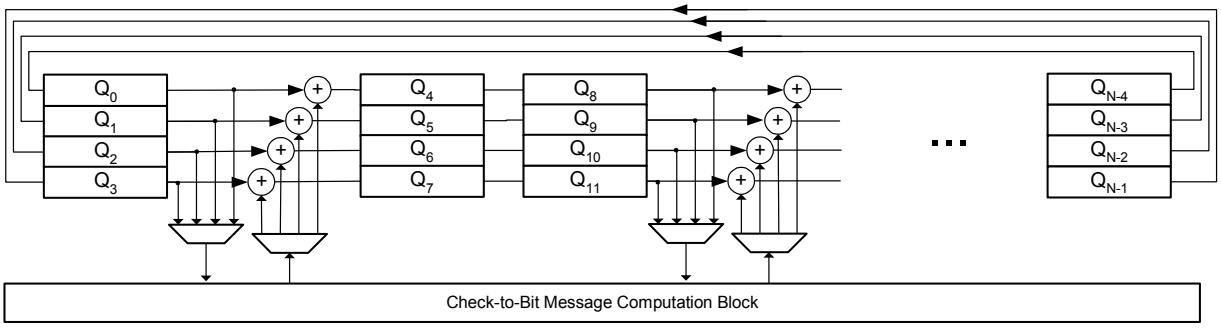
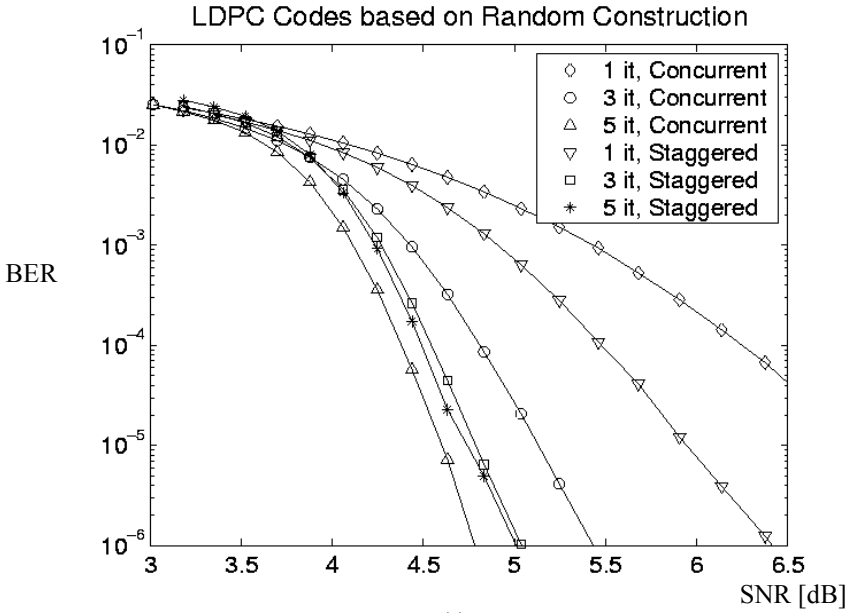
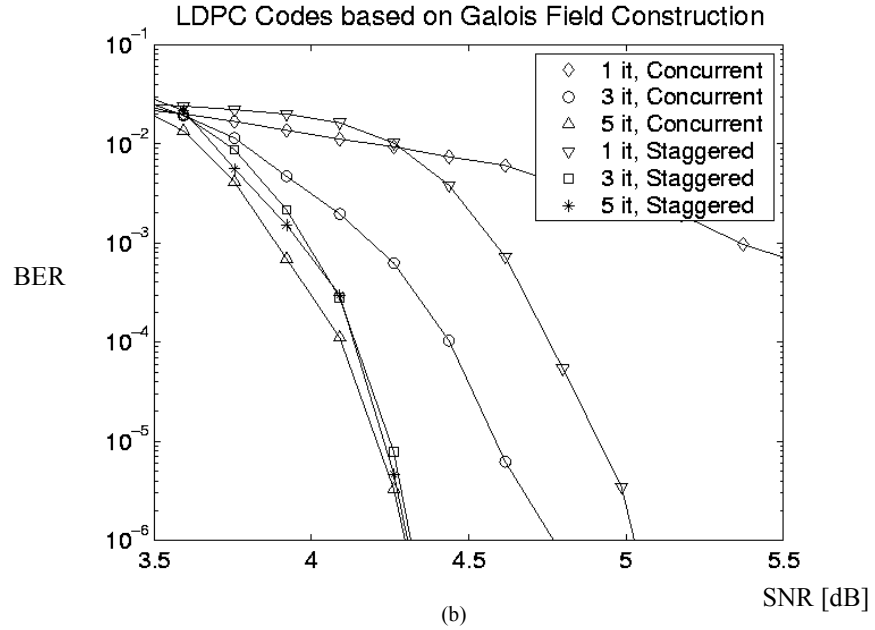


Fig. 9. Shift register-based implementation of LDPC code generated from 2D GF(2^M) with 1:4 column splitting and a message computation latency of $L=3$.



(a)



(b)

Fig. 10. Simulation results from random codes (a), and GF codes (b) with concurrent vs. staggered decoding schedule.

In order to determine the BER for a given signal-noise ratio (SNR), the simulation model adds Gaussian noise to a maximum of 30,000 blocks of data. The BER figure is continuously evaluated over the total number of decoded message bits until it has converged within 1% of the eventual value. The SNR definition in (9) represents the user and code bit energies with E_B and E_C respectively, and the noise variance with σ^2.

$$R = \frac{Num.ofUserBits}{Num.ofCodeBits};\ E_c = R \cdot E_b \tag{8}$$

$$SNR = 10 \cdot \log\left(\frac{E_b}{N_0}\right) = 10 \cdot \log\left(\frac{E_c}{2R\sigma^2}\right) \tag{9}$$

A. LDPC Codes Generated From Random Matrices

A 512×4608 parity check matrix is obtained by appending 4 random 128x4608 matrices with column weight of 1 and average row weight of 36. Each block of data comprises 4096 user bits and 512 parity bits; a code rate of $^8/_9$. Results of performing 1, 3, and 5 iterations with the concurrent decoding schedule and the staggered decoding schedule are plotted in Fig. 10a.

It is observed that the concurrent decoding schedule yields an improvement in error performance with each increase in number of iterations. The staggered decoding schedule, however, shows little improvement between 3 to 5 iterations. With 3 iterations and at BER=10^{-5}, the staggered decoding schedule achieves 0.4dB improvement over the concurrent decoding schedule. With 5 iterations, it results in less than 0.2dB degradation.

B. LDPC Codes Generated From 2D GF(2^5)

A 1023×4092 parity check matrix is based on 2D lines in GF(2^5), and a 1:4 column splitting as described previously. Each block of data comprises 3070 user bits and 1022 parity bits; a code rate of $^3/_4$. Results of performing 1, 3, and 5 iterations with both decoding schedules are shown in Fig. 10b.

The concurrent decoding schedule yields an improvement in error performance with each increase in number of iterations, while the staggered decoding schedule shows little improvement between 3 to 5 iterations. At BER=10^{-5}, the staggered decoding schedule achieves 0.3dB improvement with 3 iterations, and has less than 0.1dB degradation with 5 iterations.

C. Performance Differences

In the concurrent decoding schedule, messages from each bit node are relayed to the closest neighbors in a single iteration. Fig. 4 shows that the path for a message from bit node 4 to bit node 10 would require 2 iterations ($4_1 \rightarrow A_1 \rightarrow 1_1 \rightarrow C_2 \rightarrow 10_2$). On the other hand, the staggered decoding schedule has the capacity for messages to be relayed to more distant nodes; in the example given in Fig. 5, a path exists from bit node 4 to bit node 10 within a single iteration ($4 \rightarrow A \rightarrow 1 \rightarrow B \rightarrow 1 \rightarrow C \rightarrow 10$). The further reaching schedule is expected to converge faster, thus the better performance with low number of iterations.

The staggered decoding schedule is an approximation of the belief propagation algorithm [9], as noted in Section III-B. The use of a running sum $Q'_n(k)$ gathers cycles, with effects that are increasingly noticeable at higher number of decoding iterations. Fig. 5 contains a cycle, which starts and ends at check node A ($A \rightarrow 1 \rightarrow B \rightarrow 1 \rightarrow C \rightarrow 1 \rightarrow A$). These cycles limit the performance of the code, as is evident in Fig. 10a and Fig. 10b, which show that the staggered schedule has no BER performance advantage over the concurrent decoding schedule if the decoding is repeated for 5 iterations.

VI. Conclusion

The implementation of LDPC decoders for two classes of LDPC codes, based on random matrices and Euclidean geometries in Galois Fields, have been described. The arithmetic structures are straightforward, and meet the high throughput requirements of magnetic recording applications. The proposed staggered decoding schedule decouples the memory requirement from its dependency on the number of edges in the graph. It is recognized that the staggered decoding schedule will not achieve the same results as LDPC decoding under belief propagation. However, it represents a heuristic approach that results in significant reduction in implementation complexity. In consideration that practical power and area constraints are likely to limit the number of iterations to three, the staggered decoding schedule performs systematically better than the concurrent decoding schedule.

Besides magnetic recording applications, the staggered decoding schedule is also appropriate for forward error correction applications in wireless, wireline, and optical communication systems.

VII. References

[1] D. J. C. Mackay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," IEE Electronics Letters, vol.33, no.6, pp.457-8, March 1997.

[2] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. Magnetics*, vol.37, no.2, pp. 748-55, March 2001.

[3] R.M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol.IT-27, no.5, pp.533-47, Sept. 1981.

[4] Y. Kou, S Lin, and M. P.C. Fossorier, "Low density parity check codes based on finite geometries: a rediscovery", *Proc. IEEE ISIT*, Sorrento, Italy, p. 200, Jun 2000.

[5] J. Fan and J. Cioffi, "Constrained coding techniques for soft iterative decoders," *Proc.GLOBECOM'99*, Rio de Janeireo, Brazil, pp 723-7, Dec. 1999.

[6] A. Blanksby and C. J. Howland, "A 220mW 1-Gbit/s 1024-Bit Rate-1/2 Low Density Parity Check Code Decoder," *Proc IEEE CICC,* Las Vegas, NV, USA, pp. 293-6, May 2001.

[7] G. Al-Rawi; J. Cioffi, and M. Horowitz, "Optimizing the mapping of low-density parity check codes on parallel decoding architectures," *Proc. IEEE ITCC*, Las Vegas, NV, USA, pp.578-86, Apr 2001.

[8] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory*, vol.46, no.2, pp.325-43, March 2000.

[9] J. Pearl, "Probabilistic reasoning in intelligent systems: Networks of plausible inference," San Mateo, CA, Morgan Kaufmann, 1988.