Lower-Complexity Layered Belief-Propagation Decoding of LDPC Codes

Yuan-Mao Chang, Andres I. Vila Casado, Mau-Chung Frank Chang, and Richard D. Wesel
Department of Electrical Engineering, University of California, Los Angeles, CA 90095-1594
Email: {ymchang, avila, mfchang, wesel}@ee.ucla.edu

Abstract— **The design of LDPC decoders with low complexity, high throughput, and good performance is a critical task. A well-known strategy is to design structured codes such as quasi-cyclic LDPC (QC-LDPC) that allow partially-parallel decoders. Also, several works show that sequential schedules, such as Layered Belief-Propagation (LBP), converge faster than the traditional flooding schedule. In this paper, we propose a novel low-complexity sequential schedule called Zigzag LBP (Z-LBP). Current LBP schedules do not allow partially-parallel architectures for some codes, such as high-rate codes with small-to-medium blocklengths. Our proposed algorithm can still be implemented in a partially-parallel manner in these codes. Z-LBP provides the same benefits of LBP such as faster convergence speed and achieves lower frame error rates than flooding.**

Index Terms— **Belief-propagation, channel codes, error-control codes, low-density parity-check codes.**

I. Introduction

Low-Density Parity-Check (LDPC) codes, linear block codes defined by a very sparse parity-check matrix $\mathbf{H}$, are often proposed as the channel coding solutions for modern wireless communication systems. Medium-rate LDPC codes are used in standards such as DVB-S2 [1], WiMax (IEEE 802.16e) [2], and wireless LAN (IEEE 802.11n) [3]. Furthermore, high-rate LDPC codes have been selected as the channel coding scheme for mmWave WPAN (IEEE 802.15.3c) [4]. This recent success of LDPC codes is mainly due to structures that allow partially-parallel decoders [5]. These structured codes, called quasi-cyclic LDPC (QC-LDPC), are used in all the standards mentioned above.

QC-LDPC codes are represented as an array of sub-matrices as follows

$$\mathrm{H}_{QC} = \begin{bmatrix} A_{1,1} & \cdots & A_{1,t} \\ \vdots & & \vdots \\ A_{s,1} & \cdots & A_{s,t} \end{bmatrix}, \qquad (1)$$

where each sub-matrix $A_{i,j}$ is a $p \times p$ circulant matrix. A circulant matrix is a square matrix in which each row is a one-step cyclic shift of the previous row, and the first row is a one-step cyclic shift of the last row.

QC-LDPC decoders have a significantly higher throughput than the decoders of random sparse matrices [6]. The QC-LDPC structure guarantees that at least p messages can be computed in a parallel fashion at all times if flooding schedule is used [5]. Well designed QC-LDPC codes perform as well as random sparse matrices [7].

This work was supported by the US DARPA through the TEAM contract and the Boeing corporation.

The original message-passing schedule, called flooding scheduling, updates all the variable nodes simultaneously using the previously generated check-to-variable messages and then updates all the check nodes simultaneously using the previously generated variable-to-check messages. Sequential message-passing schedules are used to update the nodes sequentially instead of simultaneously. Several studies show that sequential scheduling not only improves the convergence speed in terms of number of iterations but also outperforms the traditional flooding scheduling for a large number of iterations. There are different types of sequential schedules, such as a sequence of check-node updates [8] [9] and a sequence of variable-node updates [10] [11]. Sequential scheduling is also presented in [12] under the name of Layered Belief Propagation (LBP). Similar ideas are presented and analyzed in [13], [14], and [15].

In this paper, we will use the term LBP to denote all sequential schedules. Check-node-centric LBP (C-LBP) indicates a sequence of check-node updates, and variable-node-centric LBP (V-LBP) indicates a sequence of variable-node updates. Simulations and theoretical results in [8]-[15] show that LBP converges twice as fast as flooding because the messages are updated using the most recent information available as opposed to updating several messages with the same pre-update information. C-LBP has the same decoding complexity per iteration as flooding [9], thus allowing the convergence speed increase at no cost. However, the V-LBP solutions proposed in [10] and [11] have a higher complexity per iteration than flooding and C-LBP. This higher complexity arises from the check-to-variable message computations as will be shown in Section II-B.

Furthermore, QC-LDPC codes where the sub-matrices can have at most one "1" per column and one "1" per row facilitate C-LBP and V-LBP decoding in a partially-parallel fashion as shown in [9] and [12]. This parity-check matrix structure allows the partially-parallel processing each of the p nodes over the bi-partite graph, and each processor uses the most recent information available. Thus, QC-LDPC structures guarantee that C-LBP and V-LBP can perform partially-parallel computations and maintain sequential schedule.

However, small-to-medium blocklength high-rate QC-LDPC codes generally need more than one diagonal per sub-matrix and only allow one row of sub-matrices. If there were more than one row of sub-matrices, the sub-matrix size will be small, thus diminishing the possible throughputs. Therefore, the C-LBP decoders presented in [9] can-

not be implemented in a partially-parallel fashion. This issue will be shown in Section IV-A.

We propose a zigzag LBP scheduling scheme called Z-LBP that can decode any LDPC code and allows partially-parallel decoding for QC-LDPC codes. This novel strategy reduces the computation complexity per iteration. Moreover, Z-LBP keeps the advantages of the sequential scheduling such as faster convergence speed and better decoding performance with respect to flooding.

This paper is structured as follows: Section II discusses the issues that arise when implementing flooding and LBP. Section III introduces Z-LBP. Section IV discusses the partially-parallel decoding implementation for high-rate QC-LDPC codes that do not allow C-LBP decoding. Section V delivers the conclusions of this paper.

II. LBP implementation issues

A. Efficient computation of the check-to-variable messages

The message from check node c_i to variable node v_j is generated using the following equation,

$$m_{c_i \to v_j} = \prod_{v_b \in \mathcal{N}(c_i) \setminus v_j} \operatorname{sgn}\left(m_{v_b \to c_i}\right) \times \phi\left(\sum_{v_b \in \mathcal{N}(c_i) \setminus v_j} \phi\left(\mid m_{v_b \to c_i} \mid\right)\right), \tag{2}$$

where $\mathcal{N}(c_i) \setminus v_j$ denotes the neighbors of c_i excluding v_j, and $\phi(x)$ is defined as $\phi(x) = -\log\left(\tanh(\frac{x}{2})\right)$. $m_{c_i \to v_j}$ is usually generated using a binary operator called Soft-XOR denoted by $\boxplus$

$$x \boxplus y = \phi\left(\phi(x) + \phi(y)\right). \tag{3}$$

Soft-XOR is commutative, associative and easy to implement [9] [16] [17]. Eq. (2) can be implemented in practice as follows,

$$m_{c_i \to v_j} = \prod_{v_b \in \mathcal{N}(c_i) \setminus v_j} \operatorname{sgn}\left(m_{v_b \to c_i}\right) \times \boxplus_{v_b \in \mathcal{N}(c_i) \setminus v_j} m_{v_b \to c_i}. \tag{4}$$

Eq. (4) shows that $d_c - 2$ Soft-XORs are required to compute each $m_{c_i \to v_j}$. Therefore, $d_c(d_c - 2)$ Soft-XORs are required to separately compute all the $m_{c_i \to v_j}$ from the same check node c_i.

However, if a message-passing schedule requires the decoder to compute all the $m_{c_i \to v_j}$ from the same c_i simultaneously, there is an efficient implementation [18]. This efficient implementation is illustrated in Fig. 1. For any degree-d_c check node, first generate $d_c - 2$ intermediate values, $f_{c_i,1} = m_{v_1 \to c_i}$, and $f_{c_i,j} = f_{c_i,j-1} \boxplus m_{v_j \to c_i}$ for $j = \{2, \ldots, d_c - 1\}$. This first step successively accumulates $m_{v_j \to c_i}$ in a forward order. Then, generate $d_c - 2$ intermediate values, $b_{c_i,d_c} = m_{v_{d_c} \to c_i}$, and $b_{c_i,j} = b_{c_i,j+1} \boxplus m_{v_j \to c_i}$ for $j = \{d_c - 1, \ldots, 2\}$. This second step successively accumulates $m_{v_j \to c_i}$ in a backward order. Finally, $m_{c_i \to v_j}$ is computed by doing $f_{c_i,j-1} \boxplus b_{c_i,j+1}$. This method uses $3(d_c - 2)$ Soft-XORs to correctly compute all the $m_{c_i \to v_j}$ from the same c_i at the same time. This algorithm is optimal in the sense that no algorithm using fewer Soft-XORs

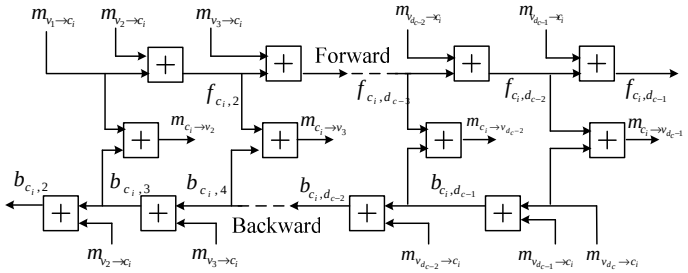

Fig. 1. The most efficient way to calculate $m_{c_i \to v_j}$ from the same c_i

can correctly compute all the $m_{c_i \to v_j}$ from the same c_i simultaneously. Flooding and C-LBP decoders use this strategy because they compute all the $m_{c_i \to v_j}$ from the same c_i at the same time.

This check-node update is equivalent to the BCJR algorithm [19] over the trellis representation of the check-node equation in the log-likelihood domain. The forward accumulation of $f_{c_i,j}$ corresponds to the BCJR α recursion in the log-likelihood domain. Also, the backward accumulation of $b_{c_i,j}$ corresponds to the BCJR β recursion in the log-likelihood domain.

B. V-LBP implementation issues

The V-LBP solutions proposed in [10] and [11] have a higher complexity per iteration than flooding and C-LBP which arises from the check-to-variable message computations. Since the V-LBP algorithm sequentially updates variable nodes, it does not allow computing all the $m_{c_i \to v_j}$ from the same c_i at the same time. Hence, the required number of Soft-XORs to compute all the $m_{c_i \to v_j}$ from the same c_i is $d_c(d_c - 2)$.

There is a method, proposed in [14], to reduce the complexity of V-LBP. Define M_{c_i} as

$$M_{c_i} = \prod_{v_b \in \mathcal{N}(c_i)} \operatorname{sgn}\left(m_{v_b \to c_i}\right) \times \boxplus_{v_b \in \mathcal{N}(c_i)} m_{v_b \to c_i}. \tag{5}$$

M_{c_i} is the Soft-XOR of all $m_{v_j \to c_i}$ destined the same c_i. A Soft-XOR's inverse operator, Soft-NXOR, denoted by $\boxminus$, is defined as

$$x \boxminus y = \phi\left(\phi(x) - \phi(y)\right). \tag{6}$$

Thus, the message from check-node c_i to variable-node v_j can be computed by

$$m_{c_i \to v_j} = M_{c_i} \boxminus m_{v_j \to c_i}. \tag{7}$$

The decoder first initializes all M_{c_i} for each check node. Then, separately generates all the $m_{c_i \to v_j}$ using Eq. (7). Also, when a new $m_{v_j \to c_i}$ is computed, M_{c_i} is re-calculated using

$$M_{c_i} = m_{c_i \to v_j} \boxplus m_{v_j \to c_i}. \tag{8}$$

In each iteration, computing all the $m_{c_i \to v_j}$ from the same c_i requires d_c Soft-NXORs. Moreover, d_c Soft-XORs are needed to re-calculate M_{c_i} since there will be d_c new $m_{v_j \to c_i}$ on every iteration. Assuming that the complexity of Soft-XOR and Soft-NXOR is the same, the number of

required operations per iteration needed to update a check node is $2d_c$. We omit the d_c Soft-XORs required to compute M_{c_i} initially in Eq. (5).

However, Soft-XOR is not invertible on every point. Without loss of generality, assume $m_{v_1 \to c_i}$ is 0. Then, M_{c_i} is 0, so $m_{c_i \to v_1} = \phi(\phi(0) - \phi(0)) = \infty$. Also, even if all $|m_{v_j \to c_i}|$ are bigger than 0, this algorithm is still numerical unstable because the dynamic range of Soft-NXOR is $[0, \infty)$. When the two arguments of Soft-NXOR are similar, the output is very large and out of quantization levels. The large quantization noise makes this strategy not practical for implementation. The authors of [17] also arrived to the conclusion that this technique is not feasible in practice.

III. Zigzag LBP

We propose a novel LBP schedule that requires fewer number of operations per iteration than flooding, C-LBP, and V-LBP to compute all the $m_{c_i \to v_j}$. Zigzag LBP is a V-LBP strategy that performs variable-node updates in a zigzag pattern over the parity-check matrix. One directional updating, forward updating or backward updating of all variable nodes, corresponds to one iteration. Zigzag updating guarantees that all the $m_{c_i \to v_j}$ can be generated by using the technique presented in Section II-A.

The Z-LBP algorithm is formally presented in Algorithm 1. First, the decoder initializes all $f_{c_i,j}$ of the every check node. Then, the first iteration, as well as all the odd iterations, consists of the sequential update of variable nodes v_j, $j = \{N \dots 1\}$ in a backward fashion. All the $m_{c_i \to v_j}$ destined to the same variable node v_j are generated using $f_{c_i,j-1} \boxplus b_{c_i,j+1}$. Then, the decoder generates all the $m_{v_j \to c_i}$ from the same v_j. Finally, the decoder calculates all the $b_{c_i,j}$ for every c_i that is a neighbor of v_j using $b_{c_i,j+1} \boxplus m_{v_j \to c_i}$. The second iteration, as well as all even iterations, updates the variable nodes v_j, $j = \{1 \dots N\}$ in a forward fashion. All the $m_{c_i \to v_j}$ of the same variable node v_j are still generated using $f_{c_i,j-1} \boxplus b_{c_i,j+1}$, and then all the $m_{v_j \to c_i}$ are generated. Finally, the decoder calculates all the $f_{c_i,j}$ for every c_i that is a neighbor of v_j using $f_{c_i,j-1} \boxplus m_{v_j \to c_i}$.

The decoder initializes all the $f_{c_i,j}$ in Line 3 of Algorithm 1 following the order of the received channel information. Hence, the decoder simultaneously receives all the channel information and initializes all the $f_{c,j}$. The Z-LBP algorithm computes all the $m_{c_i \to v_j}$ using the forward and backward technique which is described in Section II-A in a distributed fashion. However, the decoder computes $m_{c_i \to v_j}$ and either $f_{c_i,j}$ or $b_{c_i,j}$ instead of both of them in each iteration. Thus, it requires a fewer number of Soft-XORs to update a check node. Z-LBP requires $2(d_c - 2)$ Soft-XORs in order to update a check node. Flooding and C-LBP require $3(d_c - 2)$ Soft-XORs to update a check node, and V-LBP needs $d_c(d_c - 2)$ Soft-XORs. Thus, if we assume the complexity of computing check-to-variable messages is much higher than the complexity of computing variable-to-check messages [9] [18], Z-LBP is 1.5 times simpler than flooding and C-LBP and $d_c/2$ times simpler than V-LBP per iteration.

Algorithm 1 Z-LBP

1: Initialize all $m_{c_i \to v_j} = 0$
2: Initialize all $m_{v_j \to c_i}$ = Channel Information
3: Initialize all $f_{c_i,j} = f_{c_i,j-1} \boxplus m_{v_j \to c_i}$
4: $Iter = 1$
5: **if** $Iter$ is odd **then**
6: **for** every v_j, $j = \{N, \dots, 1\}$ **do**
7: **for** every $c_i \in N(v_j)$ **do**
8: Generate and propagate $m_{c_i \to v_j} = f_{c_i,j-1} \boxplus b_{c_i,j+1}$
9: **end for**
10: **for** every $c_i \in N(v_j)$ **do**
11: Generate and propagate $m_{v_j \to c_i}$
12: Compute $b_{c_i,j} = b_{c_i,j+1} \boxplus m_{v_j \to c_i}$
13: **end for**
14: **end for**
15: **else**
16: **for** every v_j, $j = \{1, \dots, N\}$ **do**
17: **for** every $c_i \in N(v_j)$ **do**
18: Generate and propagate $m_{c_i \to v_j} = f_{c_i,j-1} \boxplus b_{c_i,j+1}$
19: **end for**
20: **for** every $c_i \in N(v_j)$ **do**
21: Generate and propagate $m_{v_j \to c_i}$
22: Compute $f_{c_i,j} = f_{c_i,j-1} \boxplus m_{v_j \to c_i}$
23: **end for**
24: **end for**
25: **end if**
26: $Iter = Iter + 1$
27: **if** Stopping rule is not satisfied **then**
28: Go to Step 5;
29: **end if**

Let us denote the number of the edges of the bi-partite graph as N_E. There are N_E $f_{c,j}$ values and N_E $b_{c,j}$ values. This suggests that the Z-LBP decoder needs a memory of size $2N_E$. However, in the case of an odd iteration, the decoder computes a new $b_{c,j}$ after updating $m_{v_j \to c_i}$. Thus, the new $b_{c,j}$ can be written in the same memory address of $f_{c,j}$ given that $f_{c,j}$ is not needed anymore. The same can also be said about the even iterations, the new $f_{c,j}$ can be written in the same memory address of $b_{c,j}$. Therefore, the required memory size is only N_E. This is the same memory size required for a C-LBP decoder which is half the memory required for a flooding decoder [9].

Fig. 2 shows the AWGN performance of four different scheduling strategies, flooding, V-LBP, C-LBP, and Z-LBP as the number of iterations increases. All the simulations correspond to the blocklength-1944 rate-1/2 LDPC code presented in the IEEE 802.11n standard [3]. This figure shows that Z-LBP has a better convergence speed than flooding across all iterations. The frame error rate of flooding around 20 and 40 iterations are equal to the frame error rate of Z-LBP around 15 and 30 iterations respectively. However, since the computation complexity of Z-LBP is 1.5 times simpler than that of flooding, Z-LBP's convergence speed in terms of the number of Soft-XORs is twice as much as that of flooding. C-LBP and V-LBP's convergence speeds in terms of the number of iterations are

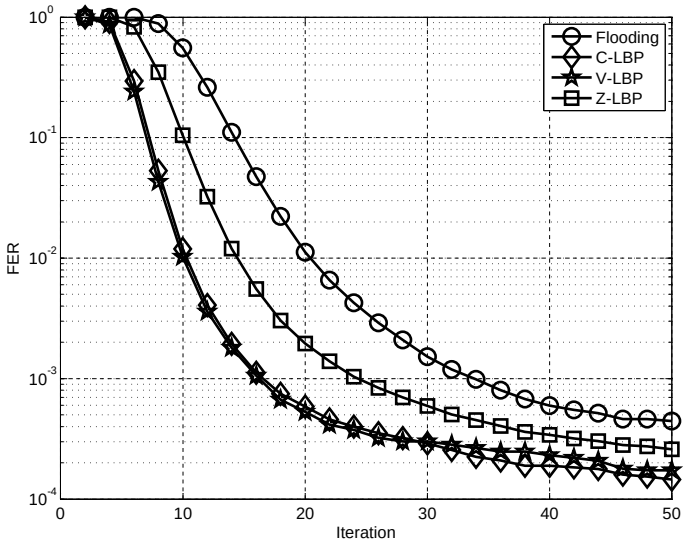

Fig. 2. Performance of flooding, C-LBP, V-LBP, and Z-LBP at different iterations for a fixed $E_b/N_0 = 1.75$ dB

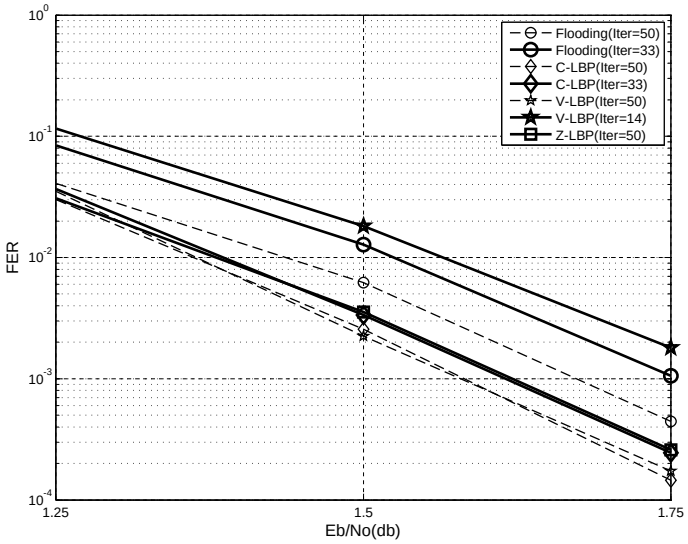

Fig. 3. Frame error rate performance of flooding, C-LBP, V-LBP, and Z-LBP for different iterations v.s. E_b/N_0

similar. Although Z-LBP's convergence speed in terms of the number of iterations is around 1.5 slower than that of C-LBP, the computation complexity of Z-LBP is 1.5 times simpler than that of C-LBP per iteration. Thus, C-LBP and Z-LBP's convergence speeds in terms of the number of Soft-XORs are almost the same.

For a degree-d_c check node, the computation complexity of Z-LBP is $d_c/2$ times simpler than that of V-LBP. The code in the IEEE 802.11n standard has the check-node degrees 7 and 8. Thus, the computation complexity of Z-LBP is 3.5 times simpler than that of V-LBP. Hence, Z-LBP's convergence speed in terms of the number of Soft-XORs is around 2 times faster than V-LBP.

Fig. 3 shows frame error rates of these four scheduling strategies presented above at different SNRs. Since the complexity of Z-LBP is 1.5 times simpler than flooding and C-LBP, the 50-iteration computation complexity of Z-LBP is equivalent to the 33-iteration that of flooding and C-LBP. Similarly, Z-LBP is 3.5 times simpler than V-LBP. Thus, the 50-iteration Z-LBP corresponds to 14-iteration V-LBP. The performance of Z-LBP is 0.15 dB better than flooding. There is no difference between C-LBP and Z-LBP's performance. However, the coding gain between Z-LBP and V-LBP is around 0.2 dB.

IV. Implementation issues for medium blocklength High-Rate LDPC codes

Modern wireless communication systems provide higher and higher throughputs. IEEE 802.11a [20] can provide tens of Mbps, and IEEE 802.11n [3] improves the throughput to hundreds of Mbps. Recently, a wireless communication standard, IEEE 802.15.3c [4] targets throughputs on the order of Gbps. Hence, high-rate LDPC codes with high-throughput decoders are needed.

Parity-check matrices of small-to-medium blocklength high-rate QC-LDPC codes have one row of sub-matrices, where each sub-matrix consists of several (more than one)

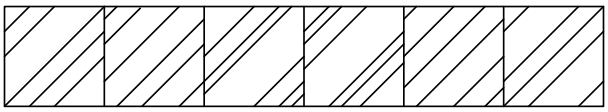

Fig. 4. The structure of a parity check matrix

cyclic-shift diagonals in order to avoid degree-1 variable nodes. In these cases, the single row of sub-matrices is necessary because multiple rows would require the sub-matrix size to be too small to provide the necessary throughput. Fig. 4 shows the structure of the parity-check matrix of a regular high-rate LDPC code. Diagonal lines represent the "1"s of $\mathbf{H}$. For example, the rate-14/15 LDPC code proposed in the IEEE 802.15.3c standard is a regular code with a similar structure to the one shown in Fig. 4. Its blocklength is 1440, and its check-node degree d_c is 45.

A. C-LBP implementation issues

Algorithm 2 describes the partially-parallel version of the C-LBP algorithm [9]. The C-LBP decoder processes one row of sub-matrices at the same time. Separate processors simultaneously update all check nodes C_l in the same row of sub-matrices l. Different variable-to-check messages $m_{V\to C_l}$ must be generated and propagated at the same time. If each sub-matrix contains at most one "1" per column and one "1" per row, the processors access disjoint sets of variable nodes. This guarantees that each processor uses the most recent information available even if all the processors perform in parallel.

However, for small-to-medium blocklength high-rate QC-LDPC codes, the C-LBP algorithm presented in [9] cannot be implemented in a partially-parallel fashion. The reason is that the parity-check matrix contains only one row of sub-matrices. Thus step 3 and 4 in Algorithm 2 become the variable-node update and check-node update of the flooding scheduling respectively. Therefore, partially-parallel C-LBP becomes exactly the same as flooding in

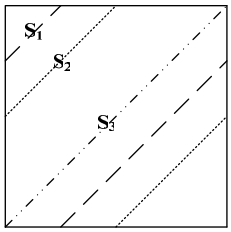

Fig. 5. The labeled cyclic-shift diagonals in one sub-matrix

complexity, convergence speed, and decoding capability. Partially-parallel C-LBP for small-to-medium high-rate QC-LDPC codes is not a sequential schedule.

Algorithm 2 Partially-Parallel C-LBP

1: Initialize all $m_{c_i \to v_j} = 0$
2: **for** every row of sub-matrix l **do**
3: Generate and propagate $m_{V \to C_l}$
4: Generate and propagate $m_{C_l \to V}$
5: **end for**
6: **if** Stopping rule is not satisfied **then**
7: Position = 2;
8: **end if**

B. Partially-parallel implementation of Z-LBP

Z-LBP can perform in a partially-parallel fashion by updating a column of sub-matrices. First, label the cyclic-shift diagonals in each sub-matrix as shown in Fig. 5. Assume there are N_{mat} sub-matrices, and each sub-matrix has N_{diag} cyclic-shift diagonals ($N_{diag} > 1$). Then, the order of variable-node updates at step 6 in Algorithm 1 is slightly changed to "**for** every column of sub-matrix SM_j $j = \{N_{mat}, \ldots, 1\}$." This labeling prevents memory access conflicts when all processors process p variable nodes at the same time. All the $m_{c_i \to v_j}$ are still computed using $f_{c,j-1} \boxplus b_{c,j+1}$. However, since $N_{diag} > 1$, the decoder requires extra $d_c - N_{mat}$ Soft-XORs in order to compute $f_{c_i,j}$ or $b_{c_i,j}$ in advance. For example, when the decoder prepares to update the sub-matrix SM_2 in a forward fashion, the decoder needs to compute $f_{c,N_{diag}+j}$ $j = \{1, \ldots, N_{diga} - 1\}$ in advance. Because of the computation $f_{c_i,j}$ or $b_{c_i,j}$ in advance, the decoder does not use the recent information available at step 8 and 18 in Algorithm 1. However, this does not diminish the performance significantly.

Consider the rate-14/15 QC-LDPC code used in IEEE 802.15.3c. The check-node degree d_c is equal to 45, and there are 15 sub-matrices. Hence, Z-LBP in a partially-parallel fashion requires 114 Soft-XORs to compute all the $m_{c_i \to v_j}$ from the same check node c_i. V-LBP needs 1935 Soft-XORs to compute all the $m_{c_i \to v_j}$ from the same c_i. The flooding schedule requires 129 Soft-XORs to compute all the $m_{c_i \to v_j}$ from the same c_i. Therefore, Z-LBP is 17 times and 1.13 times simpler than V-LBP and flooding respectively.

Fig. 6 shows the AWGN performance of three different scheduling strategies, flooding, V-LBP, and Z-LBP in a partially-parallel fashion, as the number of iterations increases. All the simulations use the same blocklength-1440 rate-14/15 LDPC code proposed in the IEEE 802.15.3c standard. The figure shows that Z-LBP in a partially-parallel fashion has better convergence speed than flooding across all iterations. Z-LBP's convergence speed in terms of the number of Soft-XORs is around 3 times faster than flooding. Moreover, the convergence speed in terms of iterations of Z-LBP and V-LBP are similar. However, Z-LBP is 17 times simpler than V-LBP. Hence, the convergence speed in terms of the number of Soft-XORs of Z-LBP is much faster than that of V-LBP.

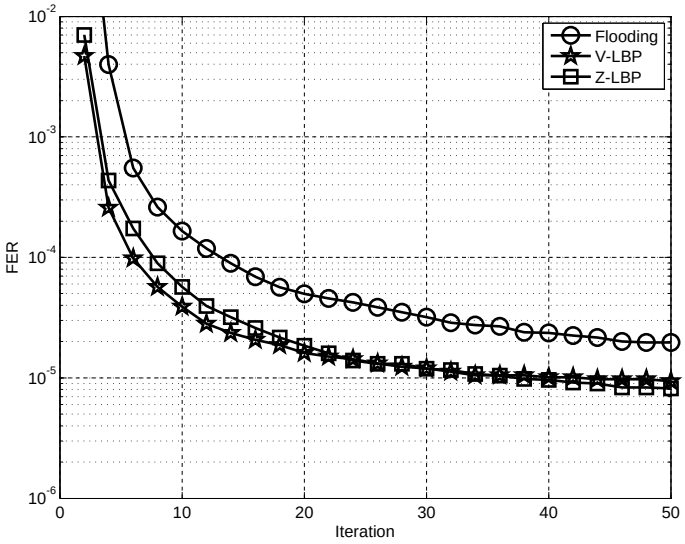

Fig. 6. Performance of flooding, C-LBP, V-LBP, and Z-LBP in a partially-parallel fashion at different iterations for a fixed $E_b/N_0 = 6.0$ dB

Fig. 7 shows frame error rates of these three scheduling strategies presented above in a partially-parallel fashion at different SNRs. Since the complexity of Z-LBP is 17 times and 1.13 times simpler than V-LBP and flooding respectively, the 50-iteration complexity of Z-LBP is equal to 3-iteration V-LBP and 44-iteration flooding. Fig. 7 shows the coding gain's gap between flooding and Z-LBP is 0.125 dB. The performance of Z-LBP is 0.5 dB better than that of V-LBP.

V. CONCLUSION

We propose Z-LBP, a low-complexity sequential schedule of variable node updates. For a degree-d_c check-node, the computation complexity per iteration of Z-LBP is $d_c/2$ times simpler than that of V-LBP. Also, Z-LBP is 1.5 times simpler than flooding and C-LBP. Z-LBP outperforms flooding with a faster convergence speed and better decoding capability.

For QC-LDPC codes where the sub-matrices can have at most one "1" per column and one "1" per row, Z-LBP can perform partially-parallel decoding. It provides the same performance as C-LBP. Therefore, Z-LBP is alternative implementation of LBP.

However, for small-to-medium blocklength high-rate

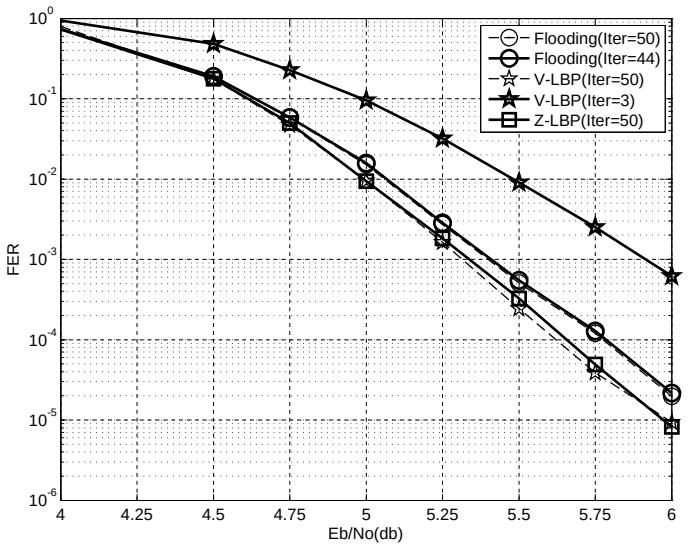

Fig. 7. Frame error rate performance of flooding, V-LBP, and Z-LBP in a partially-parallel fashion at different E_b/N_0

QC-LDPC codes whose parity-check matrix contains only one row of sub-matrices, partially-parallel C-LBP is exactly the same as flooding. In contrast, the proposed Z-LBP can still perform partially-parallel decoding and maintains a sequential schedule.

References

[1] European Telecommunications Standards Institute (ETSI). Digital Video Broadcasting (DVB) Second generation, framing structure for broadband satellite applications; EN 302 307 V1.1.1. 2005.

[2] IEEE 802.16e: Air Interface for Fixed and Mobile Broadband Wireless Access Systems, IEEE, 2004.

[3] IEEE P802.11n/D1.05 October 2006, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications - Enhancements for Higher Throughput (Draft).

[4] IEEE P802.15.3c/July 2007, Wireless Personal Area Network (WPAN) Standard Physical Layer (PHY) specifications (Draft).

[5] M.M. Mansour and N.R. Shanbhag. Turbo decoder architectures for low-density parity-check codes. In *Proc. IEEE Global Conference on Communications*, pages 1383–1388, Taipei, Taiwan, November 2002.

[6] Z. Li, L. Chen, L. Zeng, S. Lin, and W. Fong. Efficient encoding of low-density parity-check codes. *IEEE Trans. on Commun.*, 54:71–81, January 2006.

[7] L. Chen, J. Xu, I. Djurdjevic, and S. Lin. Near Shannon limit quasi-cyclic low-density parity-check codes. *IEEE Trans. on Commun.*, 52(7):1038–1042, July 2004.

[8] E. Yeo and P. Pakzad and B. Nikolic and V. Anantharam. High throughput low-density parity-check decoder architectures. In *Proc. 2001 Global Conference on Communications*, pages 3019–3024, San Antonio, TX, November 2001.

[9] M.M. Mansour and N.R. Shanbhag. High-throughput LDPC decoders. *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 11:976–996, December 2003.

[10] H. Kfir and I. Kanter. Parallel versus sequential updating for belief propagation decoding. *Physica A*, 330:259–270, 2003.

[11] J. Zhang and M. Fossorier. Shuffled belief propagation decoding. *IEEE Trans. on Commun.*, 53:209–213, February 2005.

[12] D. Hocevar. A reduced complexity decoder architecture via layered decoding of LDPC codes. In *Proc. Signal Processing Systems SIPS 2004*, pages 107–112, October 2004.

[13] E. Sharon and S. Litsyn and J. Goldberger. An efficient message-passing schedule for LDPC decoding. In *Proc. 23rd IEEE Convention of Electrical and Electronics Engineers in Israel*, pages 223–226, September 2004.

[14] P. Radosavljevic, A. de Baynast, and J. R. Cavallaro. Optimized message-passing schedules for LDPC decoding. In *Proc. Thirty-Ninth Asilomar Conference on Signals, Systems and Computers*, pages 591–595, 2005.

[15] A. I. Vila Casado, M. Griot, and R. D. Wesel. Informed dynamic scheduling for belief-propagation decoding of LDPC codes. In *IEEE ICC 2007*, pages 932–937, Glasgow, Scotland, July 2007.

[16] P. Roberston, E. Villebrun, and P. Hoeher. A comparison of optimal and sub-optimal MAP decoding algorithm operating in the log domain. In *Proc. IEEE Int. Conf. Communications*, pages 1009–1013, 1995.

[17] C Jones, S. Dolinar, K. Andrews, D. Divsalar, Y. Zhang, and W. Ryan,. Functions and Architectures for LDPC Decoding. In *IEEE ITW 2007*, pages 577–583, Lake Tahoe, California, September 2007.

[18] X.Y. Hu, E. Eleftheriou, D.M. Arnold, and A. Dholakia. Efficient Implementations of the Sum-Product Algorithm for Decoding LDPC Codes. In *Proc. IEEE Global Conference on Communications*, pages 1036–1036E, San Antonio, TX, November 2001.

[19] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate. *IEEE Trans. Info. Theory*, IT-20:284–287, March 1974.

[20] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, IEEE std. 802.11a, 1999.