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. Sequential schedules, such as Layered Belief-Propagation (LBP), converge faster than the traditional flooding schedule while allowing parallel decoding of QC-LDPC codes. 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 in the regime of high-rate codes with small-to-medium blocklengths. Our proposed algorithm can still be implemented in a partially-parallel manner in this regime. Z-LBP provides the same benefits as LBP including faster convergence speed and 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 $H$, are often proposed as 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

$$H_{QC} = \begin{bmatrix}
A_{1,1} & \cdots & A_{1,t} \\
\vdots & & \vdots \\
A_{s,1} & \cdots & A_{s,t}
\end{bmatrix},$$

(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].

The original message-passing schedule, called flooding, 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 out-dated 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] cannot 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,

\[
m_{c_i \rightarrow v_j} = \prod_{v_k \in \mathcal{N}(c_i) \setminus v_j} \text{sgn}(m_{v_k \rightarrow c_i}) \times \phi \left( \sum_{v_k \in \mathcal{N}(c_i) \setminus v_j} \phi(\{ m_{v_k \rightarrow c_i} \}) \right),
\]

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

\[
x \boxplus y = \phi(\phi(x) + \phi(y)).
\]

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

\[
m_{c_i \rightarrow v_j} = \prod_{v_k \in \mathcal{N}(c_i) \setminus v_j} \text{sgn}(m_{v_k \rightarrow c_i}) \times \boxplus m_{v_k \rightarrow c_i}.
\]

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

However, if a message-passing schedule requires the decoder to compute all the \( m_{c_i \rightarrow v_j} \) 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, \( f_{c_i,1} = m_{v_1 \rightarrow c_i} \), and \( f_{c_i,j} = f_{c_i,j-1} \boxplus m_{v_j \rightarrow c_i} \) for \( j = \{2, \ldots, d_c - 1\} \). This first step successively accumulates \( m_{v_j \rightarrow c_i} \) in a forward order. Then, generate \( d_c - 2 \) intermediate values, \( b_{c_i,1} = m_{v_{d_c} \rightarrow c_i} \), and \( b_{c_i,j} = b_{c_i,j-1} \boxplus m_{v_j \rightarrow c_i} \) for \( j = \{d_c - 1, \ldots, d_c - 2\} \). This second step successively accumulates \( m_{v_j \rightarrow c_i} \) in a backward order. Finally, the magnitude of the \( m_{c_i \rightarrow v_j} \) is computed \( f_{c_i,j-1} \boxplus b_{c_i,j-1} \). This method uses \( 3(d_c - 2) \) Soft-XORS to correctly compute all the \( m_{c_i \rightarrow v_j} \) from the same \( c_i \) at the same time. This algorithm is optimal in the sense that no algorithm using fewer Soft-XORS can correctly compute all the \( m_{c_i \rightarrow v_j} \) from the same \( c_i \) simultaneously. Flooding and C-LBP decoders use this strategy because they compute all the \( m_{c_i \rightarrow v_j} \) 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 \( \alpha \) recursion in the log-likelihood domain. Also, the backward accumulation of \( b_{c_i,j} \) corresponds to the BCJR \( \beta \) 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 \( m_{c_i \rightarrow v_j} \) from the same \( c_i \) at the same time. Hence, the required number of Soft-XORS to compute all the \( m_{c_i \rightarrow v_j} \) 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

\[
M_{c_i} = \prod_{v_k \in \mathcal{N}(c_i)} \text{sgn}(m_{v_k \rightarrow c_i}) \times \boxplus m_{v_k \rightarrow c_i}.
\]

\( M_{c_i} \) is the Soft-XOR of all \( m_{v_j \rightarrow c_i} \) destined the same \( c_i \). A Soft-XOR’s inverse operator, Soft-XNOR, denoted by \( \boxminus \), is defined as

\[
x \boxminus y = \phi(\phi(x) - \phi(y)).
\]

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

\[
m_{c_i \rightarrow v_j} = \text{sgn}(m_{v_j \rightarrow c_i}) \times M_{c_i} \boxminus m_{v_j \rightarrow c_i}.
\]

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

\[
M_{c_i} = \text{sgn}(m_{v_j \rightarrow c_i}) \times m_{c_i \rightarrow v_j} \boxminus m_{v_j \rightarrow c_i}.
\]

In each iteration, computing all the \( m_{c_i \rightarrow v_j} \) from the same \( c_i \) requires \( d_c \) Soft-XNORS. Moreover, \( d_c \) Soft-XNORS are needed to re-calculate \( M_{c_i} \) since there will be \( d_c \) new \( m_{v_j \rightarrow c_i} \) on every iteration. Assuming that the complexity of Soft-XOR and Soft-XNOR is the same, the number of required operations per iteration needed to update a check node is \( 2d_c \). We omit

Fig. 1. The most efficient way to calculate \( m_{c_i \rightarrow v_j} \) from the same \( c_i \)
the $d_c$ Soft-XORs required to compute $M_{ci}$ initially in Eq. (5).

However, Soft-XOR is not invertible on every point. Without loss of generality, assume $m_{vj \rightarrow ci}$ is 0. Then, $M_{ci}$ is 0, so $m_{ci \rightarrow vi} = \phi (\phi(0) - \phi(0)) = \infty$. Also, even if all $|m_{vj \rightarrow ci}|$ are bigger than 0, this algorithm is still numerical unstable because the dynamic range of Soft-XNOR is $[0, \infty)$. When the two arguments of Soft-XNOR 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 $m_{ci \rightarrow vj}$. 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 $m_{ci \rightarrow vj}$ can be generated using the technique presented in Section II-A.

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

The decoder initializes all the $f_{ci,j}$ 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 $f_{ci,j}$. The Z-LBP algorithm computes all the $m_{ci \rightarrow vj}$, using the forward and backward technique which is described in Section II-A in a distributed fashion. However, the decoder computes $m_{ci \rightarrow vj}$ and either $f_{ci,j}$ or $b_{ci,j}$ 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.

Let us denote the number of the edges of the bi-partite graph as $N_E$. There are $N_E$ $f_{ci,j}$ values and $N_E$ $b_{ci,j}$ values.

**Algorithm 1 Z-LBP**

1: Initialize all $m_{ci \rightarrow vj} = 0$
2: Initialize all $m_{vj \rightarrow ci} =$ Channel Information
3: Initialize all $f_{ci,j} = f_{ci,j-1} \oplus m_{vj \rightarrow ci}$
4: Iter = 1
5: if Iter is odd then
6: for every $v_j$, $j = \{N, \ldots, 1\}$ do
7: for every $c_i \in N(v_j)$ do
8: Generate and propagate $m_{ci \rightarrow vj} = f_{ci,j-1} \oplus b_{ci,j+1}$
9: end for
10: for every $c_i \in N(v_j)$ do
11: Generate and propagate $m_{vj \rightarrow ci}$
12: Compute $b_{ci,j} = b_{ci,j+1} \oplus m_{vj \rightarrow ci}$
13: end for
14: end for
15: else
16: for every $v_j$, $j = \{1, \ldots, N\}$ do
17: for every $c_i \in N(v_j)$ do
18: Generate and propagate $m_{ci \rightarrow vj} = f_{ci,j-1} \oplus b_{ci,j+1}$
19: end for
20: for every $c_i \in N(v_j)$ do
21: Generate and propagate $m_{vj \rightarrow ci}$
22: Compute $f_{ci,j} = f_{ci,j-1} \oplus m_{vj \rightarrow ci}$
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

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 $b_{ci,j}$ after updating $m_{vj \rightarrow ci}$. Thus, the new $b_{ci,j}$ can be written in the same memory address of $f_{ci,j}$ given that $f_{ci,j}$ is not needed anymore. The same can also be said about the even iterations, the new $f_{ci,j}$ can be written in the same memory address of $b_{ci,j}$. 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 which is shown in Fig. 3. Fig. 3 also shows that Z-LBP’s convergence speeds in terms of the number of Soft-XORS is a little bit better than C-LBP.

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 2.5 times faster than V-LBP.

Fig. 4 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.

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) 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. 5 shows the structure of the parity-check matrix of a regular high-rate LDPC code. Diagonal lines represent the “1”s of $H$. 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. 5. 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 $C_i$ in the same row of sub-matrices $l$. Different variable-to-check messages $m_{V \rightarrow C_i}$ 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 parity-check matrix contains only one row of sub-matrices, and there are more-than-one-one per row and column of sub-matrix which prevents decoding from being sequential. Moreover, 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 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 $m_{c_i \rightarrow v_j} = 0$
2. for every row of sub-matrix $l$
   3. Generate and propagate $m_{V \rightarrow C_i}$
   4. Generate and propagate $m_{C_i \rightarrow V}$
3. end for
4. if Stopping rule is not satisfied then
5.   Position = 2;
6. 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. 6. Assume there are $N_{\text{mat}}$ sub-matrices, and each sub-matrix has $N_{\text{diag}}$ cyclic-shift diagonals ($N_{\text{diag}} > 1$). Then, the order of variable-node updates at step 6 in Algorithm 1 is slightly changed to “for every column of sub-matrix $SM_j$, $j = \{N_{\text{mat}}, \ldots, 1\}$.” This labeling prevents memory access conflicts when all processors process $p$ variable nodes at the same time. All the $|m_{c_i \rightarrow v_j}|$ are still computed using $f_{e_{c,i,j+1}} \oplus b_{c_i,j+1}$. However, since $N_{\text{diag}} > 1$, the decoder requires extra $d_c - N_{\text{mat}}$ 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 $f_{c_{i,N_{\text{diag}}+j+1}} = \{1, \ldots, N_{\text{diag}} - 1\}$ 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 steps 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 $m_{c_i \rightarrow v_j}$ from the same check node $c_i$. V-LBP needs 1935 Soft-XORs to compute all the $m_{c_i \rightarrow v_j}$ from the same $c_i$. The flooding schedule requires 129 Soft-XORs to compute all the $m_{c_i \rightarrow v_j}$ from the same $c_i$. Therefore, Z-LBP is 17 times and 1.13 times simpler than V-LBP and flooding respectively.

Fig. 7 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. Fig. 8 illustrates the convergence speed in terms of the number of Soft-XORs. Z-LBP’s convergence speed 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. 9 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, Fig. 9 compares the 50-iteration performance of Z-LBP, the 3-iteration performance of V-LBP and the 44-iteration performance of flooding. Fig. 9 shows the performance 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 an alternative implementation of LBP.

However, for small-to-medium blocklength high-rate QC-LDPC codes whose parity-check matrix contains only one row of sub-matrices, and each sub-matrix contains multiple shifted diagonals of ones, 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.
Fig. 7. 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. 8. FER of flooding, V-LBP, and Z-LBP at different numbers of Soft-XOR operations for a fixed $E_b/N_0 = 6.0$ dB

Fig. 9. Frame error rate performance of flooding with 44 iterations, V-LBP with 3 iterations, and Z-LBP with 50 iterations in a partially-parallel fashion at different $E_b/N_0$. These numbers of iterations equalize the complexity in terms of the number of Soft-XORs.

REFERENCES
