Transactions Papers

Implementation Aspects of LDPC Convolutional Codes

Ali Emre Pusane, *Student Member, IEEE,* Alberto Jiménez Feltström, Arvind Sridharan, *Member, IEEE,* Michael Lentmaier, *Member, IEEE,* Kamil Sh. Zigangirov, *Fellow, IEEE,* and Daniel J. Costello, Jr., *Life Fellow, IEEE*

***Abstract*—Potentially large storage requirements and long initial decoding delays are two practical issues related to the decoding of low-density parity-check (LDPC) convolutional codes using a continuous pipeline decoder architecture. In this paper, we propose several reduced complexity decoding strategies to lessen the storage requirements and the initial decoding delay without significant loss in performance. We also provide bit error rate comparisons of LDPC block and LDPC convolutional codes under equal processor (hardware) complexity and equal decoding delay assumptions. A partial syndrome encoder realization for LDPC convolutional codes is also proposed and analyzed. We construct terminated LDPC convolutional codes that are suitable for block transmission over a wide range of frame lengths. Simulation results show that, for terminated LDPC convolutional codes of sufficiently large memory, performance can be improved by increasing the density of the syndrome former matrix.**

***Index Terms*—Low-density parity-check codes, convolutional codes, iterative decoding, message-passing decoding, decoder implementation.**

I. Introduction

LOW-density parity-check (LDPC) block codes were invented four decades ago by Gallager [1]. These codes are known to achieve excellent bit error rate (BER) performance on a variety of channels. Analysis and design of these codes has attracted great interest in the literature, particularly since their rediscovery in the last decade by Wiberg [2], MacKay and Neal [3], Kou *et al.* [4], and many others. Several analytical tools have been presented [5], [6] to obtain performance limits when LDPC codes are decoded by iterative message-passing algorithms. These tools have been successfully employed to design codes that achieve near capacity performance [7], [8].

The convolutional counterpart of LDPC block codes, LDPC convolutional codes, were first proposed[1] in [10]. Analogous to LDPC block codes, LDPC convolutional codes are defined by sparse parity-check matrices, which allow them to be decoded using iterative message-passing algorithms. The so-called pipeline decoder [10], that is typically used to decode these codes, has a potentially long initial decoding delay and high storage requirements. The pipeline decoder outputs a continuous stream of decoded data once this initial decoding delay has elapsed. Therefore it is essential for delay-sensitive scenarios that this initial decoding delay be minimized. Both the delay and storage requirements of the decoder are proportional to the number of iterations performed and to the convolutional code constraint length. In this paper, we propose several improvements to the pipeline decoder proposed in [10] to decrease both the storage requirements and the initial decoding delay.

The first practical VLSI hardware architecture for LDPC convolutional decoders was proposed by Bates *et al.* [11], and in [12] it was shown that terminated LDPC convolutional codes are appropriate for packet data transmission in Ethernet networks. In this paper, we consider several implementation aspects of LDPC convolutional codes, such as enhanced code constructions, pipeline and circular decoding architectures, stopping rules for the decoder, an on-demand decoding schedule, and terminated transmission of LDPC convolutional codes.

Given their excellent BER performance, it is quite natural to compare LDPC convolutional codes with corresponding LDPC block codes. For finite block lengths, a systematic comparison of these codes has not yet been proposed in the literature. In this paper, we consider several comparisons of these codes under certain assumptions, i.e., equal processor (hardware) complexity and equal decoding delay.

Paper approved by S. G. Wilson, the Editor for Coding Theory and Applications of the IEEE Communications Society. Manuscript received November 16, 2005; revised March 21, 2007. This work was supported in part by NSF Grants CCR02-05310 and CCF05-15012, and by NASA Grant NNX07AK536. Portions of this paper were presented at the IEEE International Symposium of Information Theory, Chicago, IL, June 2004.

A. E. Pusane, K. Sh. Zigangirov, and D. J. Costello, Jr. are with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556, USA (e-mail: dcostel1@nd.edu).

A. Jiménez Feltström is with DS2 Design of Systems on Silicon, Valencia, Spain.

A. Sridharan was with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. He is now with Seagate Technology, Longmont, CO, USA.

M. Lentmaier was with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. He is now with Vodafone Chair Mobile Communications Systems, Technische Universitat Dresden, Dresden, Germany.

Digital Object Identifier 10.1109/TCOMM.2008.050519.

[1]The basic concept of LDPC convolutional codes was independently presented by Tanner in a patent application [9].

$$\mathbf{H}^{\mathrm{T}}_{[t,t']} = \begin{bmatrix} \mathbf{H}^{\mathrm{T}}_0(t) & & \cdots & \mathbf{H}^{\mathrm{T}}_{m_s}(t+m_s) & \mathbf{0} & & \\ \mathbf{0} & \mathbf{H}^{\mathrm{T}}_0(t+1) & \cdots & & \mathbf{H}^{\mathrm{T}}_{m_s}(t+m_s+1) & \ddots & \\ \vdots & \mathbf{0} & \ddots & \vdots & \vdots & \ddots & \\ & & \ddots & \mathbf{H}^{\mathrm{T}}_0(t') & \cdots & & \mathbf{H}^{\mathrm{T}}_{m_s}(t'+m_s) \end{bmatrix},\ t \le t',$$

II. LDPC Convolutional Codes

We start with a brief definition of a rate $R = b/c$ binary LDPC convolutional code $\mathcal{C}$. (A more detailed description can be found in [10], [13], [14].) Let

$$\mathbf{u}_{[0,t-1]} = [\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{t-1}]\,, \tag{1}$$

where $\mathbf{u}_i = (u_i^{(1)}, u_i^{(2)}, ..., u_i^{(b)}), 0 \le i < t, t \in \mathcal{Z}^+$, and $u_i^{(\cdot)} \in GF(2)$, be an information sequence. The encoder maps this sequence into the code sequence

$$\mathbf{v}_{[0,t-1]} = [\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_{t-1}]\,, \tag{2}$$

where $\mathbf{v}_i = (v_i^{(1)}, v_i^{(2)}, ..., v_i^{(c)}), 0 \le i < t, t \in \mathcal{Z}^+$, and $v_i^{(\cdot)} \in GF(2)$. We assume that the encoder is systematic and will therefore use the notation $\mathbf{v}_i = [\mathbf{v}_i^{(0)}, \mathbf{v}_i^{(1)}]$, where $\mathbf{v}_i^{(0)} = \mathbf{u}_i$ and $\mathbf{v}_i^{(1)}$ is a parity-check vector of length $(c-b)$.

A code sequence $\mathbf{v}_{[0,\infty]}$ satisfies the equation

$$\mathbf{v}_{[0,\infty]}\mathbf{H}^{\mathrm{T}}_{[0,\infty]} = \mathbf{0}, \tag{3}$$

where the matrix at the top of this page is a transposed parity check matrix, also called the syndrome former of the convolutional code $\mathcal{C}$. The submatrices $\mathbf{H}_i(t)$, $i = 0, 1, \cdots, m_{\mathrm{s}}$, are binary $(c-b) \times c$ submatrices given by

$$\mathbf{H}_i(t) = \begin{bmatrix} h_i^{(1,1)}(t) & \cdots & h_i^{(1,c)}(t) \\ \vdots & & \vdots \\ h_i^{(c-b,1)}(t) & \cdots & h_i^{(c-b,c)}(t) \end{bmatrix}. \tag{4}$$

They satisfy the following properties:

1) $\mathbf{H}_i(t) = \mathbf{0},\ i < 0$ and $i > m_s,\ \forall\ t$.
2) There is a t such that $\mathbf{H}_{m_{\mathrm{s}}}(t) \neq \mathbf{0}$.
3) $\mathbf{H}_0(t) \neq \mathbf{0}\ \forall\ t$, has full rank.

We call m_{s} the syndrome former memory, and $\nu_{\mathrm{s}} = (m_{\mathrm{s}}+1) \cdot c$ is the decoding constraint length. It determines the width of the nonzero diagonal region of $\mathbf{H}^{\mathrm{T}}_{[0,\infty]}$. Sparsity of the syndrome former is ensured by demanding that its rows have very low Hamming weight, i.e., $w_H(\mathbf{h}_i) << (m_{\mathrm{s}}+1) \cdot c,\ i \in \mathcal{Z}^+$, where $\mathbf{h}_i$ denotes the i-th row of $\mathbf{H}^{\mathrm{T}}_{[0,\infty]}$. The code is said to be regular if its syndrome former $\mathbf{H}^{\mathrm{T}}_{[t,\infty]}$ has exactly J ones in every row and K ones in every column starting from the $(m_{\mathrm{s}} \cdot c + 1)$-th column. The other entries are zeros. We will refer to a code with these properties as an (m_s, J, K)-LDPC convolutional code. An (m_s, J, K)-LDPC convolutional code is called periodic with period T if $\mathbf{H}_i(t),\ i \in \mathcal{Z}^+$, is periodic, i.e., $\mathbf{H}_i(t) = \mathbf{H}_i(t+T), \forall\ i, t$.

A description of the construction of a periodic syndrome former $\mathbf{H}^{\mathrm{T}}_{[0,\infty]}$ from the transposed parity-check matrix of an LDPC block code is given in [10]. For the LDPC convolutional codes considered in this paper, we chose an LDPC block code parity-check matrix randomly and then, using the procedure in [10], transformed it to a syndrome former.

III. Encoding of LDPC Convolutional Codes

In this section, we consider two realizations for encoding rate $R = b/c$, (m_{s}, J, K)-LDPC convolutional codes – a syndrome former and a partial syndrome former realization.

A. Syndrome Former Realization

Equation (3) can be rewritten as

$$\mathbf{v}_t\mathbf{H}^{\mathrm{T}}_0(t) + \mathbf{v}_{t-1}\mathbf{H}^{\mathrm{T}}_1(t) + \ldots + \mathbf{v}_{t-m_{\mathrm{s}}}\mathbf{H}^{\mathrm{T}}_{m_{\mathrm{s}}}(t) = \mathbf{0},\ t \in \mathbb{Z}. \tag{5}$$

Since the submatrices $\mathbf{H}^{\mathrm{T}}_0(t)$, $t \in \mathbb{Z}$, have full rank, (5) can be used to define the code block $\mathbf{v}_t$. Given the information block $\mathbf{u} = \mathbf{v}_t^{(0)}$ and the previous code blocks $\mathbf{v}_{t-1}, \mathbf{v}_{t-2}, \cdots, \mathbf{v}_{t-m_{\mathrm{s}}}$, we can calculate parity block $\mathbf{v}_t^{(1)}$.

When the last $c-b$ linearly independent rows of $\mathbf{H}^{\mathrm{T}}_0(t)$ are chosen as the $(c-b) \times (c-b)$ identity matrix, a systematic encoder can be defined using (5) as follows:

$$v_t^{(j)} = u_t^{(j)},\ j = 1, \ldots, b\,, \tag{6}$$

$$v_t^{(j)} = \sum_{k=1}^{b} v_t^{(k)} h_0^{(j-b,k)}(t) + \sum_{i=1}^{m_{\mathrm{s}}} \sum_{k=1}^{c} v_{t-i}^{(k)} h_i^{(j-b,k)}(t),\quad j = b+1, \ldots, c\,, \tag{7}$$

and the parity symbols $v_t^{(j)}$, $j = b+1, \cdots, c$, can be determined using shift-registers [10]. This syndrome former encoder realization requires $c \cdot m_{\mathrm{s}} + b$ memory units and the encoding complexity per bit is proportional to $K-1$, independent of the codeword length and the syndrome former memory m_{s}. On the other hand, a straightforward encoder for a length N LDPC block code that multiplies the information sequence by the generator matrix has a complexity per bit that is proportional to N. Therefore, the special structure of LDPC convolutional codes gives them a clear advantage compared to LDPC block codes in terms of encoding complexity.

B. Partial Syndrome Former Realization

Let $\mathbf{v}_{[0,t-1]}$ be the sequence of code symbols corresponding to a systematically encoded block of information symbols $\mathbf{u}_{[0,t-1]}$. For any $t > 0$ this sequence satisfies

$$\mathbf{v}_{[0,t-1]}\mathbf{H}^{\mathrm{T}}_{[0,t-1]} = [\ \mathbf{0}_{[0,t-1]} \mid \mathbf{p}_t\ ]. \tag{8}$$

Here, $\mathbf{0}_{[0,t-1]}$ is a zero vector of length $(c-b) \cdot t$ and

$$\mathbf{p}_t = [\ \mathbf{p}_{t,1},\ \mathbf{p}_{t,2},\ \ldots,\ \mathbf{p}_{t,m_{\mathrm{s}}}], \tag{9}$$

where $\mathbf{p}_{t,i} = (\ p_{t,i}^{(1)},\ p_{t,i}^{(2)},\ \ldots,\ p_{t,i}^{(c-b)}\ )$, $i = 1, 2, \cdots, m_{\mathrm{s}}$. Vector $\mathbf{p}_t$ is called the *partial syndrome*. By definition, $\mathbf{p}_t$ is the state of a partial syndrome former encoder at time t.

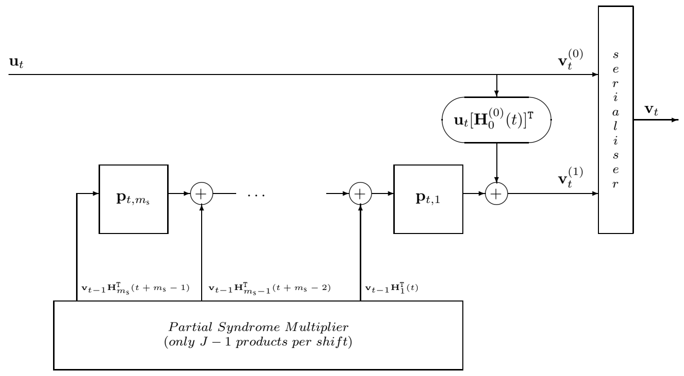
$\mathbf{u}_t$
$\mathbf{v}_t^{(0)}$
$\mathbf{u}_t[\mathbf{H}_0^{(0)}(t)]^{\mathrm{T}}$
$\mathbf{v}_t^{(1)}$
serialiser
$\mathbf{v}_t$
$\mathbf{p}_{t,m_s}$
$\mathbf{p}_{t,1}$
$\mathbf{v}_{t-1}\mathbf{H}_{m_s}^{\mathrm{T}}(t+m_s-1)$
$\mathbf{v}_{t-1}\mathbf{H}_{m_s-1}^{\mathrm{T}}(t+m_s-2)$
$\mathbf{v}_{t-1}\mathbf{H}_1^{\mathrm{T}}(t)$
Partial Syndrome Multiplier
(only $J-1$ products per shift)

Fig. 1. Block diagram of an $R = b/c$, (m_s, J, K)-LDPC convolutional code partial syndrome former encoder.

From (8), we calculate $\mathbf{p}_t$ recursively as a function of $\mathbf{p}_{t-1}$ and $\mathbf{v}_{t-1}$ using

$$\mathbf{p}_{t,i} = \begin{cases} \mathbf{p}_{t-1,i+1} + \mathbf{v}_{t-1}\mathbf{H}_i^{\mathrm{T}}(t+i-1), & i = 1, 2, \ldots, m_s - 1, \\ \mathbf{v}_{t-1}\mathbf{H}_{m_s}^{\mathrm{T}}(t+m_s-1), & i = m_s. \end{cases} \tag{10}$$

This partial syndrome former encoder can be implemented using a shift register that employs $(c-b)\cdot m_s$ memory units (Figure 1). The contents of the shift register at any time t is $\mathbf{p}_t$.

Given the encoder state at time t, we can generate the parity bits at time t by using

$$[\mathbf{v}_t^{(0)},\ \mathbf{v}_t^{(1)}]\mathbf{H}_0^{\mathrm{T}}(t) = \mathbf{p}_{t,1}. \tag{11}$$

Let us express the matrix $\mathbf{H}_0(t)$ as

$$\mathbf{H}_0(t) = [\mathbf{H}_0^{(0)}(t),\ \mathbf{H}_0^{(1)}(t)], \tag{12}$$

where $\mathbf{H}_0^{(0)}(t)$ is a $(c-b)\times b$ matrix and $\mathbf{H}_0^{(1)}(t)$ is a $(c-b)\times(c-b)$ matrix having full rank. Then, from (11) we obtain

$$\mathbf{v}_t^{(0)}[\mathbf{H}_0^{(0)}(t)]^{\mathrm{T}} + \mathbf{v}_t^{(1)}[\mathbf{H}_0^{(1)}(t)]^{\mathrm{T}} = \mathbf{p}_{t,1}. \tag{13}$$

In the case when $\mathbf{H}_0^{(1)}(t)$ is an identity matrix, we have

$$\mathbf{v}_t^{(1)} = \mathbf{v}_t^{(0)}[\mathbf{H}_0^{(0)}(t)]^{\mathrm{T}} + \mathbf{p}_{t,1}. \tag{14}$$

Equation (14) defines the parity subblock in terms of the partial syndrome $\mathbf{p}_{t,1}$ and the information sequence $\mathbf{v}_t^{(0)} = \mathbf{u}_t$.

This partial syndrome former encoder realization needs only $(c-b)\cdot m_s$ memory units, less than that of the syndrome former realization proposed in [10] and outlined in the previous subsection, but the encoding (computational) complexity is the same. However, the partial syndrome former realization is more convenient because, as we will see in section IV, it allows us to encode terminated LDPC convolutional codes in a simple way.

IV. Termination of LDPC Convolutional Codes

LDPC convolutional codes are very efficient for the transmission of streaming data since they allow continuous encoding/decoding. However, in some applications, it is preferable to have the data encoded in frames of pre-determined size in order to maintain compatibility with some standard format. We now consider termination of LDPC convolutional codes in this context.

The information sequences must be terminated with a tail of symbols to force the encoder to the zero state at the end of the encoding process. For conventional polynomial convolutional encoders, the terminating tail consists of a sequence of zeros. For LDPC convolutional code encoders, the tail is, generally speaking, non-zero and depends on the encoded information bits. Therefore, we need to solve a system of linear equations. In this section we propose a convenient way to calculate the tail bits using the partial syndrome encoder.

Consider the case when the sequence $\mathbf{u}_{[0,L-1]}$ has been encoded and we must find the tail $\mathbf{v}_{[L,L+\tau-1]}$ of $c\tau$ symbols, corresponding to an input sequence $\mathbf{u}_{[L,L+\tau-1]}$, to bring the encoder to the zero state, i.e., the partial syndrome $\mathbf{p}_{L+\tau-1}$ should be zero. This means that the code sequence must satisfy the condition

$$\mathbf{v}_{[0,L+\tau-1]}\mathbf{H}_{[0,L+\tau-1]}^{\mathrm{T}} = \mathbf{0}_{[0,L+\tau+m_s-1]}. \tag{15}$$

Equation (15) can be split into two conditions

$$\begin{aligned} \mathbf{v}_{[0,L-1]}\mathbf{H}_{[0,L-1]}^{\mathrm{T}} &= [\ \mathbf{0}_{[0,L-1]} \mid \mathbf{p}_L] & (16)\\ \mathbf{v}_{[L,L+\tau-1]}\mathbf{H}_{[L,L+\tau-1]}^{\mathrm{T}} &= [\ \mathbf{p}_L \mid \mathbf{0}_{[0,\tau-m_s-1]}]. & (17) \end{aligned}$$

The first condition is equivalent to (8) and the second condition defines a linear system of $(c-b)\cdot\tau$ equations for finding the tail $\mathbf{v}_{[L,L+\tau-1]}$.

Note that some of the equations in (17) may be linearly dependent and therefore redundant. Let us suppose that we have excluded from matrix $\mathbf{H}_{[L,L+\tau-1]}^{\mathrm{T}}$ all dependent columns, obtaining a $c\tau\times\psi$ matrix $\mathbf{F}(L)$, where $\psi \le \min[(m_s+\tau)(c-b), c\tau]$. Now we can rewrite (17) as

$$\mathbf{v}_{[L,L+\tau-1]}\mathbf{F}(L) = [\ \mathbf{p}_L^* \mid \mathbf{0}], \tag{18}$$

where $\mathbf{p}_L^*$ is the partial syndrome vector $\mathbf{p}_L$ with omitted redundant components and $\mathbf{0}$ is a zero vector with length such that the length of $[\ \mathbf{p}_L^* \mid \mathbf{0}]$ is ψ. The rank of $\mathbf{F}(L)$ is full, it has a right inverse $\mathbf{F}^{-1}(L)$, and we can rewrite (18) as

$$\mathbf{v}_{[L,L+\tau-1]} = [\ \mathbf{p}_L^* \mid \mathbf{0}\ ]\mathbf{F}^{-1}(L). \tag{19}$$

Equation (19) defines the termination tail $\mathbf{v}_{[L,L+\tau-1]}$. Using this result, it is straightforward to express the subblocks of the tail as a linear combination of the partial syndrome components

$$\mathbf{v}_t = \sum_{i=1}^{m_s} \mathbf{p}_{L,i}\ \mathbf{f}_{i,t}(L),\ L \le t \le L+\tau-1, \tag{20}$$

where the $\mathbf{f}_{i,t}(L)$ are $(c-b)\times c$ matrices, given by the inverse $\mathbf{F}^{-1}(L)$, that are calculated in advance from $\mathbf{H}_{[L,L+\tau-1]}^{\mathrm{T}}$ and $\mathbf{p}_{L,i}$ are the contents of the shift register of the partial syndrome encoder at time $t = L$ (Figure 1).

Equation (20) shows that for the tail calculation, it is sufficient to save the partial syndrome former components $\mathbf{p}_{L,i}$, $i = 1, ..., m_s$, at time L and to use them in the calculation of the tail. If the LDPC convolutional code is periodic with period T and if the information block length L is chosen to be a multiple of T, i.e., $L = nT, n \in \mathcal{Z}^+$, then $\mathbf{f}_{i,t}(L)$ is independent of L and the same terminating circuitry can be used for different block lengths.

For the simulations of LDPC convolutional codes reported in Section VIII, we constructed the syndrome former matrices randomly. We selected only codes having tail lengths $\tau = 2(m_s + 1)$. Codes requiring longer tails were rejected. It is interesting to note that we only had to reject about half of all randomly generated codes.

From a theoretical point of view, it is important to study the asymptotic behavior of the tail length τ, when the syndrome former memory goes to infinity. In [15] and [16], an ensemble $\mathcal{C}_1(m_s, J, 2J)$ of $(m_s, J, 2J)$-LDPC convolutional codes with syndrome formers composed of random permutation matrices was considered. The following theorem confirms our empirical observations that rate $R = 1/2$, $(m_s, J, 2J)$-LDPC convolutional codes can always be chosen such that the tail length τ does not exceed $2 \cdot (m_s + 1)$.

Theorem 4.1: If $m_s \rightarrow \infty$, almost all codes in $\mathcal{C}_1(m_s, J, 2J)$ can be terminated with a tail of length $\tau = 2 \cdot (m_s + 1)$ (proof given in [15]).

V. Decoding of LDPC Convolutional Codes

There are several ways to decode an LDPC convolutional code. In one approach, we terminate the encoded sequence to form a frame of pre-determined length. The received word is then decoded by a message-passing algorithm, such as belief-propagation [1], similar to the decoding of LDPC block codes.

Semi-infinite (non-terminated) LDPC convolutional codes can also be iteratively decoded using a message-passing algorithm. Although the corresponding Tanner graph has an infinite number of nodes, the distance between two variable nodes that are connected to the same check node is limited by the memory of the code. This allows continuous decoding with a decoder that operates on a finite window sliding along the received sequence, similar to a Viterbi decoder with finite path memory [13]. The decoding of two variable nodes that are at least $(m_s + 1)$ time units apart can be performed independently, since the corresponding bits cannot participate in the same parity-check equation. This allows the parallelization of the I iterations by employing I independent identical processors working on different regions of the Tanner graph simultaneously. A pipeline decoder that is based on this idea was introduced by Jiménez Feltström and Zigangirov in [10]. The operation of this decoder on the Tanner graph for a simple time-invariant rate $R = 1/3$ LDPC convolutional code with $m_s = 2$ is shown in Figure 2.

When the first c received symbols of the transmitted sequence enter the pipeline decoder, it takes $I \cdot (m_s + 1)$ time units for this c-tuple to reach the output of the decoder where the decision on these symbols is made. Once this initial decoding delay of the pipeline decoder has elapsed, the decoder produces a continuous output stream, i.e., at each time unit, c newly received values enter the decoder and c decoded bits leave it. The I processors perform the I decoding iterations simultaneously. At each time unit, the i-th processor begins by activating the first column of $(c - b)$ check nodes in its operating region and then proceeds to activate the last column of c variable nodes that are about to leave the operating

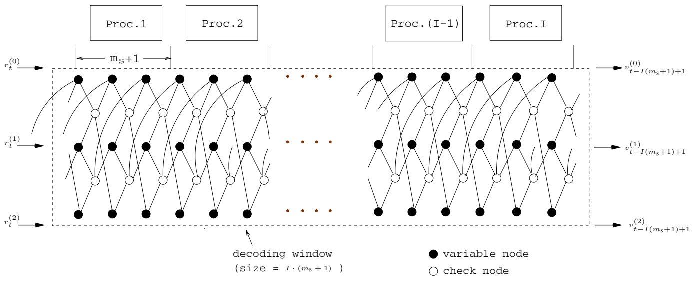

Fig. 2. Tanner graph of R=1/3 LDPC convolutional code and illustration of pipeline decoding.

region. A check node activation is the step where the check node collects all the incoming messages from its neighboring variable nodes, calculates the outgoing messages, and sends the new messages to each neighboring variable node. Correspondingly, during a variable node activation, all the incoming messages to a variable node from the neighboring check nodes are collected and the new outgoing messages are calculated. Then the new messages are sent to each neighboring check node.

VI. Improvements to the Pipeline Decoder

To insure the independence of consecutive decoding iterations, the operating regions of neighboring processors do not overlap. As noted above, this results in a fairly long initial decoding delay of $I \cdot (m_s + 1)$ time units. Also, since each variable node must store a channel value and one storage element is needed for each edge in the Tanner graph, a total memory of $(J + 1) \cdot I \cdot (m_s + 1) \cdot c = (J + 1) \cdot I \cdot \nu_s$ storage elements is required. Both quantities are linear functions of the number of iterations I and the processor separation $(m_s + 1)$. In this section, we present several reduced complexity decoding strategies to decrease the initial decoding delay and the memory requirements without significantly reducing the performance of the decoder.

These strategies can also be used with a circular memory topology, similar to the one used in [17], which is more suitable for decoding terminated LDPC convolutional codes. This circular memory topology is used in the simulations of terminated LDPC convolutional codes reported in Section VIII.[2]

A. A Stopping Rule for the Pipeline Decoder

For LDPC block codes, the decoded sequence is a valid codeword if and only if all of the parity-check equations are satisfied after some number of iterations, or, equivalently, all components of the syndrome vector are zero. When using belief-propagation decoding, it almost never occurs that the decoder converges to a different codeword after further iterations. This suggests stopping decoding once the decoder output converges to a codeword. This is a well-known tool that saves both time and power consumption in the decoding process.

However, this stopping rule cannot be used in the continuous pipeline convolutional decoder because of the finite size

[2]All simulation results presented in this paper assume the use of the sum-product algorithm.

operating window, semi-infinite size parity-check matrix, and the undetermined length of the input sequence. For such an LDPC convolutional code decoder, it would be cumbersome to check whether all the parity-check equations are satisfied at the completion of decoding. In this section, we present a way of allowing the individual processors to determine when to stop and restart.

Since decoding is performed over a finite size window, instead of trying to determine whether the entire decoded sequence is a codeword or not, one can consider segments of the decoded sequence and determine whether they match a segment of a valid codeword, i.e., whether the corresponding components of the syndrome vector are zero. This control of the processors is realized by using individual counters for each processor. If the c-tuple that enters the operating region of the i-th processor is observed to satisfy the first column of check nodes in this operating region, i.e., the corresponding component of the syndrome vector is zero, the i-th counter is incremented by one. If the counter value exceeds a stopping parameter P, the processor enters the sleep mode and stops processing. (The stopping condition is still checked in this mode.) When a c-tuple that does not satisfy the first column of check nodes enters the decoder, the counter is reset to zero and the processor resumes operation. The complexity overhead the stopping rule introduces is quite small compared to the total complexity of the tasks the processor performs at each time unit.

Intuitively, the stopping parameter P can be chosen equal to the syndrome-former memory m_s. This insures that all the bits entering a decoding region in succeeding time intervals will lie on a valid codeword until a non-zero syndrome component is observed. Simulation results for the BER performance of the proposed stopping rule for non-terminated LDPC convolutional codes with $J = 3$, $K = 6$, $P = m_s$, and different syndrome former memories over an AWGN channel with signal-to-noise ratio (SNR) E_b/N_0 per bit have been obtained. Compared to simulation results for the conventional pipeline decoder, no performance degradation is observed.

The stopping rule performs very well for various memory lengths. For the $(512, 3, 6)$ code and a maximum allowable number of iterations $I = 100$, the average number of iterations I_{av} performed per bit at 1.00dB SNR is 64, and this value drops to 16 at 1.25dB. Although the stopping decoder does not reduce the required decoder memory, depending on the quality of the received sequence and the value of m_s, it performs many fewer iterations than the non-stopping decoder without any loss in BER performance. The realized savings in power consumption can be translated into decoding time if, instead of I different processors, a single processor performs the iterations one-by-one by hopping to successive operating regions.[3] It is also possible to use the stopping rule with $P < m_s$. However, if P is too small, the BER will increase.

B. On-Demand Variable Node Activation Schedule

For decoding LDPC block codes, the standard message-passing schedule calls for, at each iteration, all the variable nodes, and then all the check nodes, to be activated. This scheduling corresponds to the message-passing schedule of the pipeline decoder introduced in [10], in which, within a processing region, the $c - b$ check node activations are performed just after a set of c variable nodes enters the region, and the c variable node activations are performed just before a set of c variable nodes leaves the region. However, using this scheduling, the decoder does not make use of the most recent information at the variable nodes. The incoming information to a variable node is included in its outgoing messages only after an iteration is completed. In this subsection, we propose an on-demand variable node activation schedule for the pipeline decoder.

[3]This would require clocking the decoder according to the average number of iterations performed and employing a buffer at the channel output.

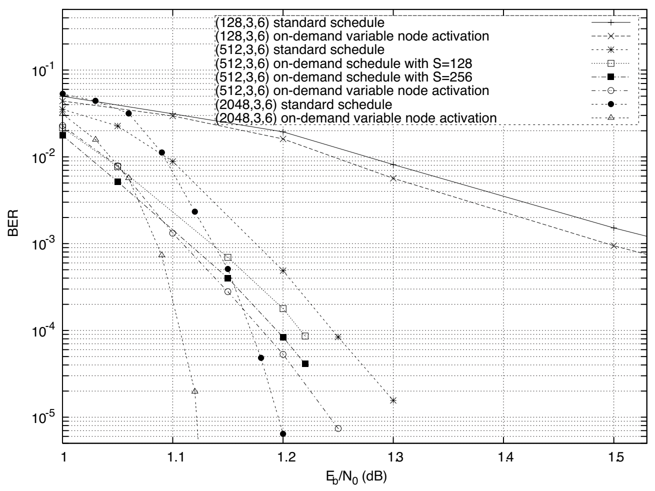

Fig. 3. Bit error rate performances of on-demand variable node scheduling and the compact decoder.

According to the proposed schedule, the activation order of the check nodes does not change, but instead of activating the variable nodes that are about to leave an operating region, we activate a variable node whenever its message is required by an active check node. Using this approach, the check nodes are able to access the most recent information available from the variable nodes. Also, the messages generated during a check node activation are immediately available for further check node activations within the same iteration. This on-demand scheduling allows the effects of message passing to flow faster through the Tanner graph.

BER performance simulation results are presented in Figure 3 for codes of various memory lengths. These results show that allowing some messages to flow faster through the Tanner graph improves performance, as long as these messages have high enough reliabilities. The $(128, 3, 6)$ code results indicate that weaker codes may not take full advantage of this scheme, whereas the higher memory order codes exhibit a $0.05 - 0.10$dB gain at high SNR.

From the computational point of view, the standard and on-demand variable node activation schedules have exactly the same complexity, since the number of computations is identical for both schemes. The check node activation complexity is also clearly the same for the two schemes. On the one hand, on-demand scheduling activates each variable node J times per iteration, which is the number of neighboring check nodes that require new information, whereas standard scheduling activates each variable node just once per iteration. But, in the standard schedule, once a variable node is activated, all J outgoing messages are calculated, whereas the variable node activation in on-demand scheduling corresponds only to a single outgoing message calculation from the variable node to the check node that has triggered the activation.

Compared to standard scheduling, the on-demand variable node activation schedule achieves a lower BER for a fixed number of iterations. This advantage can be translated into reduced delay and storage requirements by noting that the on-demand schedule requires fewer iterations to achieve a given fixed BER, thus reducing the initial decoding delay and requiring fewer processors and memory units in the decoder implementation.

C. Compact Pipeline Decoder Architecture

The requirement that the processing regions of the pipeline decoder cannot overlap follows directly from the independence of the operations performed in the standard schedule. But, once on-demand scheduling is employed, we can let neighboring operating regions overlap to decrease the total memory requirements of the decoder. As a result of the overlap, the information generated in the i-th processor's region will be immediately available to the $(i+1)$-th processor. These two processors will both have access to the information stored in the overlapping region. In such an arrangement, the BER is expected to increase, since the bits that belong to the overlapping region are not ready for the next iteration. Therefore, there will be a trade-off between the distance between neighboring processors and BER performance. We introduce the overlap by denoting the processor separation (which is equal to $(m_\mathrm{s}+1)$ in the standard decoder) as a variable S. This implementation is called the reduced-memory (compact) decoder.

The simulation results given in Figure 3 show that, as long as S is chosen to satisfy $S > m_\mathrm{s}/2$, the compact decoder causes negligible performance loss while reducing memory requirements and initial decoding delay by almost a factor of two. One reason for the performance loss when $S < m_\mathrm{s}/2$ is that even the non-neighboring $(i-1)$-th and $(i+1)$-th regions overlap, i.e., messages that haven't even been processed completely by the $(i-1)$-th processor are used in the $(i+1)$-th iteration.

Although the computational complexity of the proposed decoder is the same as the standard one, the storage requirement is much less. The total number of memory units required for the compact decoder is just $(I\cdot S+(m_\mathrm{s}+1-S))\cdot c\cdot(J+1)$. (The additional memory units corresponding to the $(m_\mathrm{s}+1-S)$ additional time units are required by the last processor to successfully decode the c bits leaving the decoder. This amount is usually very small compared to $I\cdot S$.)

VII. Implementation Complexity Comparisons with LDPC Block Codes

In this section, we compare several aspects of decoding LDPC convolutional codes to LDPC block codes.

A. Computational Complexity

Let C_{check} (C_{var}) denote the number of computations required for a check (variable) node update for a check (variable) node of degree K (J). Regardless of the code structure, C_{check} and C_{var} only depend on the values J and K.

For a rate $R=b/c$, (m_s,J,K)-LDPC convolutional code decoded using a pipeline decoder with I iterations/processors, at every time instant each processor activates $c-b$ check nodes and c variable nodes. The computational complexity per decoded bit is therefore given by

$$\begin{aligned} C_{bit}^{conv} &= ((c-b)\cdot C_{check}+c\cdot C_{var})\cdot I/c \qquad (21)\\ &= ((1-R)\cdot C_{check}+C_{var})\cdot I, \end{aligned}$$

which is independent of the constraint length ν_s.

Similarly, the decoding complexity for an (N,J,K)-LDPC block code is calculated as

$$\begin{aligned} C_{bit}^{block} &= (N\cdot\frac{J}{K}\cdot C_{check}+N\cdot C_{var})\cdot I/N \qquad (22)\\ &= (\frac{J}{K}\cdot C_{check}+C_{var})\cdot I\\ &= ((1-R)\cdot C_{check}+C_{var})\cdot I, \end{aligned}$$

which is again independent of the code length N. Thus, there is no difference between block and convolutional LDPC codes with respect to computational complexity.

B. Processor (Hardware) Complexity

The pipeline decoder implementation of an LDPC convolutional code operates on $I\cdot\nu_\mathrm{s}$ symbols. However, as described in the previous sections, decoding can be carried out by using I identical independent parallel processors each capable of handling only ν_s symbols. Hence, it is sufficient to design the processor hardware for ν_s symbols. For an LDPC block code of length N, the processor must be capable of handling all N symbols. Therefore, for the same processor complexity, the block length of an LDPC block code must be chosen to satisfy $N=\nu_\mathrm{s}$.

C. Storage Requirements

For the pipeline decoder, we need a storage element for each edge in the corresponding Tanner graph. Each variable node also needs a storage element for the channel value. Thus a total of $I\cdot(J+1)\cdot\nu_\mathrm{s}$ storage elements are required for I iterations of decoding. Similarly, we need $N\cdot(J+1)$ storage elements for the decoding of an LDPC block code of length N. Thus, for the same storage requirements, an LDPC block code must satisfy $N=I\cdot\nu_\mathrm{s}$.

D. Decoding Delay

Let T_s denote the time between the arrival of successive symbols, i.e., the symbol rate is $1/T_s$. Then the maximum time from the arrival of a symbol until it is decoded is given by

$$\Delta_{io}^{conv}=((c-1)+(m_\mathrm{s}+1)\cdot c\cdot I)\cdot T_s. \qquad (23)$$

The first term $(c-1)$ in (23) represents the time between the arrival of the first and last of the c encoded symbols

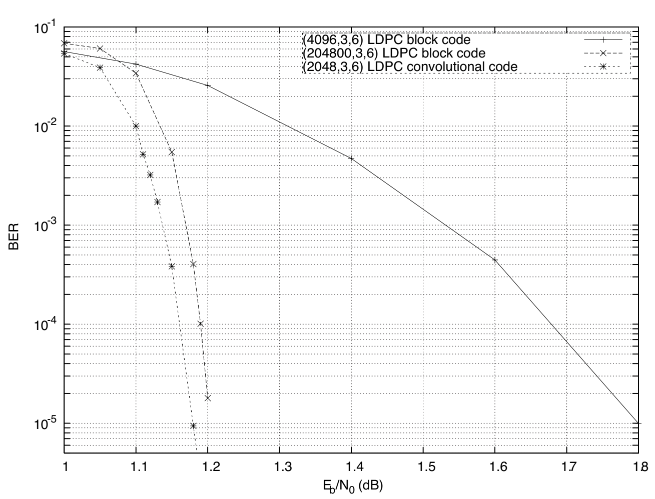

Fig. 4. Performance comparison of LDPC block and convolutional codes.

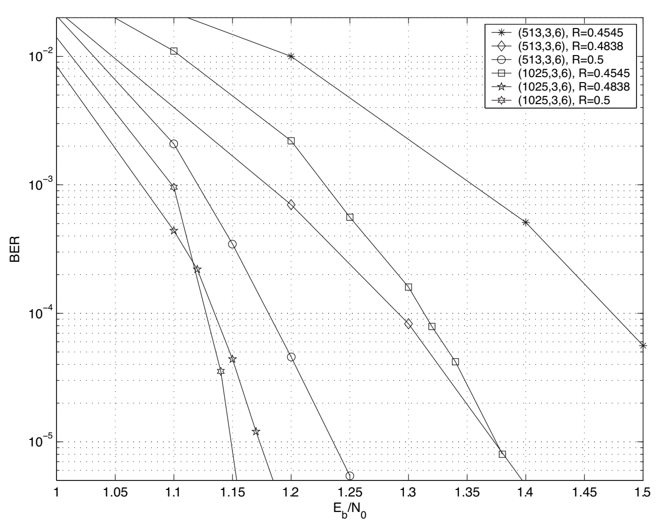

Fig. 5. Simulation results for terminated $(513, 3, 6)$ and $(1025, 3, 6)$ - LDPC convolutional codes with rates $R = 0.4545$ and 0.4838. For comparison, the performance of the unterminated codes of rate $R = 0.5$ is also shown.

output by a rate $R = b/c$ convolutional encoder in each encoding interval. The dominant second term $(m_s+1)\cdot c\cdot I$ is the time each symbol spends in the decoding window. Since c symbols are loaded into the decoder simultaneously, the pipeline decoder also requires a buffer to hold the first $(c-1)$ symbols.

With LDPC block codes, data is typically transmitted in a sequence of blocks. Depending on the data rate and the processor speed, several scenarios are possible. We consider the best case for block codes, i.e., each block is decoded by the time the first bit of the next block arrives. This results in a maximum input-output delay of $\Delta_{io}^{block} = N \cdot T_s$. [4] Thus, for equal decoding delays, the block length must satisfy $N = (c-1) + \nu_s \cdot I$, assuming the least possible delay for block codes.

In Figure 4, we plot the performance of a rate $R = 1/2$, (2048,3,6)-LDPC convolutional code with $I = 50$ iterations on an AWGN channel. Also shown is the performance of two $J = 3$, $K = 6$ LDPC block codes with a maximum number of 50 iterations. The block lengths were chosen so that in one case the decoders have the same processor complexity, i.e., $N = \nu_s$, and in the other case the same decoding delay (assuming the least possible delay for the block codes), i.e., $N = (c-1) + \nu_s \cdot I$. For the same processor complexity, the convolutional code outperforms the block code by about 0.6 dB at a bit error rate of 10^{-5}. For the same decoding delay, the convolutional and block code performance is nearly identical. Note that in the case of equal decoding delay, the memory requirements for the block code are also almost the same as for the convolutional code.

Typically, the data rates and decoding speeds are such that some buffering is required to decode block codes. There are several different decoding architectures possible in this case. However, the delay in all of these scenarios is at least as much as in the scenario we have considered earlier, so that the convolutional code would outperform the block code in each of these cases for the same decoding delay.

An LDPC block code can also be decoded by a decoder with an architecture similar to the convolutional code, by treating it as a convolutional code with memory zero. Such a decoder would consist of a buffer to store N symbols along with I processors, each performing one iteration of message-passing. In this case, $\Delta_{io}^{block} = (N - 1 + I \cdot N) \cdot T_s$ and $T_{dec}(N, I = 1) \leq N \cdot T_s$. (The block length N is analogous to c in this case.) This decoder structure for block codes is appropriate when transmitting a large amount of data ($>> N$) at high data rates.

VIII. Performance of Terminated LDPC Convolutional Codes

In our analysis of the performance of terminated LDPC convolutional codes on an AWGN channel, we are interested in the effects of the following factors: syndrome former memory, frame length, relative density of the syndrome former matrix, and code rate.

We have studied the dependence of the BER on SNR for terminated (m_s, J, K) codes with $b = 1$ and $c = 2$ when the syndrome former memory varies from $m_s = 513$ to 4093. Syndrome former matrices of density $J = 3, 4$, and 5 and $K = 2J$ were investigated. Frame lengths of $N_t = 20(m_s-1)$, $40(m_s-1)$, and $60(m_s-1)$ resulted in rates (after termination) of $R = 0.4545$, 0.4762, and 0.4838, respectively. The maximum number of iterations was 200, and a stopping rule was employed to reduce the average number of iterations.

In Figure 5, we present terminated LDPC convolutional code simulation results for $m_s = 513$ and 1025, $J = 3$, $K = 6$, and frame lengths $N_t = 20(m_s-1)$ and $60(m_s-1)$. For comparison purposes, simulation results for unterminated transmission are also provided. We see that, for each value of m_s, the code performance stays within a range of approximately 0.2-$0.3dB$ for the different values of N_t. This ensures that a single LDPC convolutional code can be used

[4]Note that the block decoder does not need any buffering under such conditions. Let $T_{dec}(N, I)$ denote the time required to perform I iterations of message-passing decoding on a block of N symbols. Thus we require $T_{dec}(N, I) \leq T_s$, i.e., this scenario requires extremely fast processors or very low data rates. By contrast, $T_{dec}(c, I = 1) \leq c \cdot T_s$ for convolutional codes.

TABLE I
COMPUTATIONAL AND MEMORY REQUIREMENTS OF THE PROPOSED DECODERS.

Implementation Type	Memory Requirement	Computational Complexity
Standard	$I \cdot (J+1) \cdot (m_s+1) \cdot c$	$I \cdot (c-b) \cdot C_{check} + I \cdot c \cdot C_{var}$
On-demand	$I \cdot (J+1) \cdot (m_s+1) \cdot c$	$I \cdot (c-b) \cdot C_{check} + I \cdot c \cdot C_{var}$
On-demand/compact	$I \cdot (J+1) \cdot S \cdot c + (J+1) \cdot (m_s+1-S) \cdot c$	$I \cdot (c-b) \cdot C_{check} + I \cdot c \cdot C_{var}$
On-demand/stopping	$I \cdot (J+1) \cdot (m_s+1) \cdot c$	$I_{av} \cdot (c-b) \cdot C_{check} + I_{av} \cdot c \cdot C_{var} + I \cdot C_{stop}$

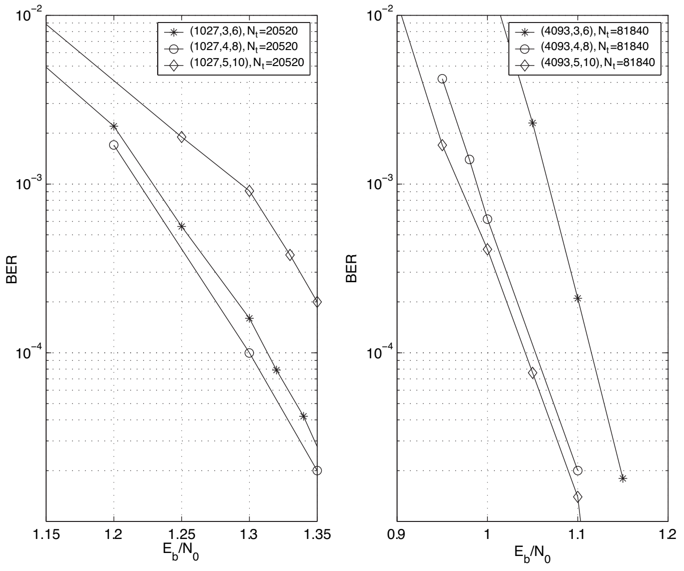

Fig. 6. Simulation results for rate $R = 0.4545$ terminated (m_s, J, K)-LDPC convolutional codes with frame length $N_t = 20(m_s - 1)$.

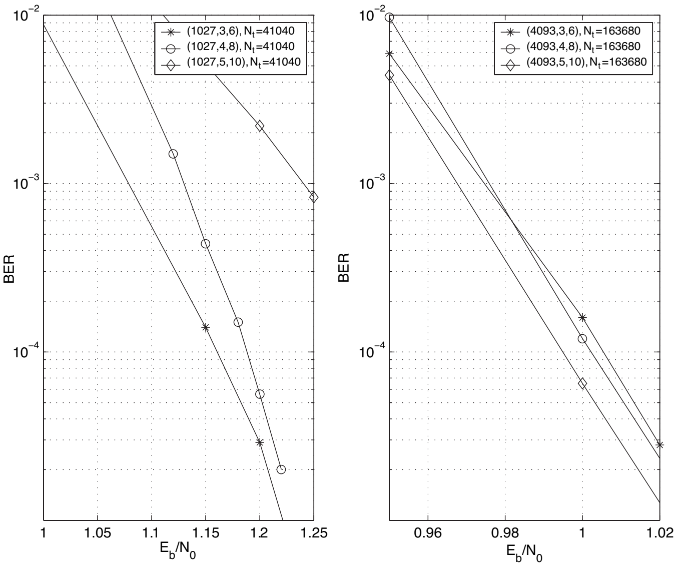

Fig. 7. Simulation results for rate $R = 0.4762$ terminated (m_s, J, K)-LDPC convolutional codes with frame length $N_t = 40(m_s - 1)$.

to achieve very good BER performance over a wide range of frame lengths. This is a property of LDPC convolutional codes that is not shared by their block code counterparts. As a result, in applications that require different frame sizes, LDPC block codes can be inefficient, since a new code must be constructed each time a change in frame size is required. LDPC convolutional codes, on the other hand, are much more robust to changes in frame size.

During the simulations, we observed that the average number of iterations for decoding the $(513, 3, 6)$ and $(1025, 3, 6)$ codes depends only slightly on the frame length N_t and decreases with increasing syndrome former memory m_s. At $E_b/N_0 = 1.0dB, 1.4dB$, and $1.8dB$, the average number of iterations was approximately $160, 40$, and 20, respectively.

In Figures 6 and 7, we present simulation results for terminated $(m_s, J, 2J)$ codes with $b = 1$, $c = 2$, $m_s = 1027$ and 4093, different syndrome former densities $J = 3$, 4, and 5, and frame lengths $N_t = 20(m_s - 1)$ and $40(m_s - 1)$. The results show that the $(1027, 3, 6)$ and $(1027, 4, 8)$ codes perform almost identically, whereas the $(1027, 5, 10)$ code performs much worse. However, with memory $m_s = 4093$, the $(J, K) = (5, 10)$ codes have the best performance, followed by the $(4, 8)$ and $(3, 6)$ codes. This improved performance with increasing J when m_s is large is consistent with the results of [18], where the AWGN channel iterative decoding thresholds of an ensemble of $(m_s, 3, 6)$, $(m_s, 4, 8)$, and $(m_s, 5, 10)$ codes have been calculated as 0.46 dB, 0.26 dB, and 0.21 dB, respectively. Interestingly, this result is in contrast to the threshold behavior of regular LDPC block codes, where the minimum threshold for rate $R = 1/2$ codes is achieved by $(3, 6)$ codes.

IX. CONCLUSIONS

In this paper, we have discussed several implementation aspects of decoding LDPC convolutional codes, such as systematic encoding, code termination, and a sliding window decoder operating on the infinite size Tanner graph. Several improvements to the original pipeline decoder architecture were presented. Specifically, an on-demand variable node activation technique was shown to improve performance, whereas a stopping rule and compact decoder architecture decrease either storage requirements or computational complexity with negligible performance loss. The computational and memory requirements of the proposed schemes per c decoded bits are given in Table I, where I is the maximum allowed number of iterations, I_{av} is the average number of iterations performed by the stopping decoder, and C_{check}, C_{var}, and C_{stop} denote the check node activation, variable node activation, and stopping rule complexities, respectively. Check node activation requires complex calculations of floating point numbers, variable node activation is just a summation of floating point numbers, and the stopping rule consists of making hard decisions on floating point numbers and then performing a binary XOR operation, which results in a complexity ordering of $C_{check} >> C_{var} >$

C_{stop}. The compact decoder reduces memory requirements by almost half when S is chosen as $m_s/2$ with minimal performance loss, and the stopping decoder significantly reduces the required number of iterations, especially at high SNR. These results show that the pipeline decoder architecture is an efficient way to decode LDPC convolutional codes when the proposed reduced complexity decoding strategies are employed. We also investigated terminated transmission of LDPC convolutional codes, which is important for many practical applications. Simulation results show that a single LDPC convolutional code can be used to achieve very good BER performance over a wide range of frame lengths. We also demonstrated that regular LDPC convolutional codes with larger syndrome former matrix density outperform codes of smaller density, consistent with the results of [18].

Finally, we provided a rudimentary comparison of LDPC block and convolutional codes. Simulation results indicate that, for equal processor (hardware) complexity, LDPC convolutional codes outperform their block code counterparts, whereas for equal decoder delay and memory requirements, the performance is about the same.

References

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21-28, Jan. 1962.

[2] N. Wiberg, "Codes and decoding on general graphs," Ph.D. thesis, Linkoping University, Sweden, 1996.

[3] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron, Lett.*, vol. 32, pp. 1645-1646, Aug. 1996.

[4] Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and more," *IEEE Trans. Inform. Theory*, vol. IT-47, pp. 2711-2736, Nov. 2001.

[5] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. IT-47, pp. 599-618, Feb. 2001.

[6] S. Y. Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of sumproduct decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans. Inform. Theory*, vol. IT-47, pp. 657-670, Feb. 2001.

[7] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. IT-47, pp. 619-637, Feb. 2001.

[8] S. Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. L. Urbanke, "On the design of low-density parity-check codes within 0.0045 db of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, pp. 58-60, Feb. 2001.

[9] R. M. Tanner, "Error-correcting coding system," U.S. Patent # 4,295,218, Oct. 1981.

[10] A. Jimenez-Feltstrom and K. Sh. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. Inform. Theory*, vol. IT-45, pp. 2181-2191, Sept. 1999.

[11] S. Bates and G. Block, "A memory based architecture for low-density parity-check convolutional decoders," in *Proc. IEEE International Symposium on Circuits and Systems*, Kobe, Japan, May 2005.

[12] S. Bates, Z. Chen, and X. Dong, "Low-density parity check convolutional codes for Ethernet networks," in *Proc. IEEE Pacific Rim Conference on Communications*, in *Proc. Computers and Signal Processing*, Victoria, B.C., Canada, Aug. 2005.

[13] S. Lin and D. J. Costello, Jr., Error Control Coding. Englewood Cliffs, NJ: Prentice-Hall, 2nd ed., 2004.

[14] R. Johannesson and K. Sh. Zigangirov, *Fundamentals of Convolutional Coding*. IEEE Press, 1999.

[15] A. Sridharan, Design and analysis of LDPC convolutional codes," Ph.D. thesis, University of Notre Dame, Notre Dame, Indiana, U.S.A., 2005.

[16] A. Sridharan, D. Truhachev, M. Lentmaier, D. J. Costello, Jr., and K. Sh. Zigangirov, "Distance bounds for an ensemble of LDPC convolutional codes," *IEEE Trans. Inform. Theory*, vol. IT-53, pp. 4537–4555, Dec. 2007.

[17] A. Schaefer, A. Sridharan, M. Moerz, J. Hagenauer, and D. J. Costello, Jr., "Analog rotating ring decoder for an LDPC convolutional code," in *Proc. IEEE Information Theory Workshop*, Paris, France, pp. 226-229, Apr. 2003.

[18] A. Sridharan, M. Lentmaier, K. Sh. Zigangirov, and D. J. Costello, Jr., "Terminated LDPC convolutional codes with thresholds close to capacity," in *Proc. IEEE Intl. Symposium on Information Theory*, Adelaide, Australia, pp. 1372-1376, Sept. 2005.

Ali Emre Pusane was born in Istanbul, Turkey in 1978. He received his B.Sc. and M.Sc. degrees in Electronics and Communications Engineering from Istanbul Technical University, Istanbul, Turkey in 1999 and 2002, respectively. He received his M.Sc. degree in Electrical Engineering in 2004 and another M.Sc. degree in Applied Mathematics in 2006, both from the University of Notre Dame, Notre Dame, IN, where he recently received his Ph.D. degree in Electrical Engineering. His research interests include error control coding, coded modulation, and information theory.

Alberto Jiménez Feltström was born in Málaga, Spain, on June 28, 1964. He received the Ph.D. and T.Lic. degree in telecommunications and the M.S. in Electrical Engineering from the University of Lund (Sweden) in 2006, 1997 and 1995, respectively.

From 1997-2000, he was working in Ericsson Mobile Communications (Sweden) with audio processing algorithm development. Later, in the period 2001-2006 he was with AT4 wireless (Spain) where he was responsible for the development of the physical layers of cellular test equipments for GSM/GPRS/EDGE/UMTS/WCDMA. Currently he is working with DS2 (Spain) making research and development for the physical layer of the next generation of broadband power-line communications devices. His current interests are in the area of channel coding, digital modulation and detection theory.

Arvind Sridharan received the B.Tech degree in electrical engineering from the Indian Institute of Technology, Madras, India in 1999 the M.S. and Ph.D degrees in electrical engineering from the University of Notre Dame, Notre Dame, IN, in 2001 and 2005, respectively. He is currently a member of the Coding and signal procesing group at Seagate Technology in Longmont, Colorado. His research interests include coding theory, iterative algorithms and information theory.

Michael Lentmaier was born in Ellwangen, Germany. He received the Dipl.-Ing. degree in electrical engineering from University of Ulm, Ulm, Germany in 1998, and the Ph.D. degree in telecommunication theory from Lund University, Lund, Sweden in 2003. As a Postdoctoral Research Associate he spent 15 months with the coding research group at University of Notre Dame, Indiana, and four months in the Department of Telecommunications and Applied Information Theory at University of Ulm. From January 2005 to December 2007 he was with the Institute of Communications and Navigation at the German Aerospace Center (DLR), Oberpfaffenhofen, Germany. He is now with Vodafone Chair Mobile Communications Systems, Technische Universitat Dresden, Dresden, Germany. His research interests include coding theory, with emphasis on iterative decoding of block and convolutional codes, and sequential Bayesian estimation techniques with applications to multipath mitigation in navigation receivers.

Kamil Sh. Zigangirov was born in the U.S.S.R. in 1938. He received the M.S. degree in 1962 from the Moscow Institute for Physics and Technology, Moscow, U.S.S.R., and the Ph.D. degree in 1966 from the Institute of Radio Engineering and Electronics of the U.S.S.R. Academy of Sciences, Moscow, U.S.S.R.

From 1965 to 1991, he held various research positions at the Institute for Problems of Information Transmission of the U.S.S.R. Academy of Sciences, Moscow, first as a Junior Scientist, and later as a Main Scientist. During this period, he visited several universities in the United States, Sweden, Italy, and Switzerland as a Guest Researcher. He organized several symposia on information theory in the U.S.S.R. In 1994, he received the Chair of Telecommunication Theory at Lund University, Lund, Sweden. In 2003 and 2004, he has been a Visiting Professor at the University of Notre Dame, Notre Dame, IN, and at the University of Alberta, Edmonton, AB, Canada. His scientific interests include information theory, coding theory, detection theory, and mathematical statistics. In addition to papers in these areas, he published a book on sequential decoding of convolutional codes (in Russian) in 1974. With R. Johannesson, he co-authored the textbook *Fundamentals of Convolutional Coding* (Piscataway, NJ: IEEE Press, 1999). His book *Theory of CDMA Communication* was published by IEEE Press in 2004.

Daniel J. Costello, Jr. received the Ph.D. in Electrical Engineering from the University of Notre Dame in 1969, after which he joined the Illinois Institute of Technology as an Assistant Professor. In 1985 he became Professor at the University of Notre Dame and later served as Department Chair. In 1999, he received the Humboldt Research Prize from Germany, and in 2000 he was named Bettex Professor of Electrical Engineering at Notre Dame.

Dr. Costello has been a member of IEEE since 1969 and was elected Fellow in 1985. Since 1983, he has been a member of the Information Theory Society Board of Governors, and in 1986 he served as President. He also served as Associate Editor for *IEEE Transactions on Communications* and *IEEE Transactions on Information Theory*, and as Co-Chair of the ISIT conferences in Kobe, Japan (1988), Ulm, Germany (1997), and Chicago, IL (2004). In 2000 he was selected by the IT Society as a recipient of a Third-Millennium Medal.

Dr. Costello's research interests are in error control coding and coded modulation. He has more than 300 technical publications and has co-authored a textbook entitled *Error Control Coding: Fundamentals and Applications*, the 2nd edition of which was published in 2004.