USING INPUT/OUTPUT QUEUES TO INCREASE LDPC DECODER PERFORMANCE

Esa Alghonaim, Aiman El-Maleh and Adnan Al-Andalusi

King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia

ABSTRACT

The paper presents a novel approach to increase the performance and/or throughput of iterative belief propagation (BP) decoding of low density parity check (LDPC) codes. The proposed approach is based on utilizing the decoder idle time by introducing two queue s: one at the decoder input and the other at the decoder output. At the presence of an input queue, the decoder runs extra iterations beyond the maximum allowable iterations as long as the input queue is not full. The function of the output queue is to preserve decoder timing, guaranteeing frames to be decoded within a fixed time similar to a conventional LDPC decoder, making it practical for real time applications.
Simulation results for a rate ½ (1024,512) progressive edge-growth (PEG) LDPC code show that the proposed approach can increase the decoder performance up to 69% keeping the same throughput, or doubling the throughput while keeping performance almost the same.

Index Terms— LDPC codes, Belief Propagation iterative decoding, Error correction coding.

1. INTRODUCTION

One of the leading families of error-correcting codes is known as Low Density Parity Check (LDPC) codes, which were first introduced by Gallager in [1]. LDPC codes have been found to rival other state-of-the-art coding families (such as Turbo Codes), and demonstrate performance that can asymptotically achieve the information-theoretic limits, while at the same time having the distinct advantage of low-complexity, near-optimal soft message-passing iterative decoding based on the Belief Propagation (BP) algorithm (also known as the Sum-Product algorithm) [2,3].

One of the advantageous features of LDPC codes is that error detection is always achieved as a by-product of the BP decoding, without the need of extra cyclic redundancy check-CRC codes (unlike other FEC families, like turbo, for example). This is because decoder failures, after reaching a maximum pre-set number of iterations (usually in the range of 100), are always known.

The selection of the value of the decoder maximum iterations has a direct effect on both decoder error correction performance and decoder throughput. Increasing maximum iterations increases error correction performance but decreases throughput, and vice versa. Figure 1 shows an example of a PEG (1024,512) decoded with 64 maximum iterations. It can be seen that at the point 2.5dB, decoder is active 10% of the time and becomes idle waiting for the next frame for 90% of the time. In other words, at point 2.5dB, on the overage, frames are decoded within 6.4 iterations out of the maximum 64 iterations.
In this research we utilize the inherent decoder feature of error detection to allow for decoding iterations beyond the maximum iterations without affecting decoder throughput. We accomplish this target by introducing two queues: one at the decoder input and one at the decoder output.
Several previous works implemented similar technique to increase LDPC decoder performance. In [4], the authors showed that adding additional buffering at the input of an iterative decoder increased the throughput of the decoder. They implemented a buffer at the input of a turbo decoder with additional cyclic redundancy check (CRC) as a stopping criterion. In [5], an input buffer is used at the input of an LDPC decoder to decrease the overall complexity of the decoding circuit. In [7], they did a similar approach as in [4] but for an LDPC decoder. They did the analysis and simulation using the decoder for the next generation satellite digital video broadcasting (DVB-S2) as a case study.
According to our knowledge, all previous works are based on implementing input buffers (queue) to increase the throughput or decrease the decoder complexity. Their approach suffers from a major problem: the big variance of decoding time, which makes their approach not practical, especially for real time applications, such as DVB-S2.
In this paper, we could solve this problem by introducing another queue, same size as the input queue, at the output of the LDPC decoder. Given a conventional LDPC decoder with a pre-set number of maximum decoding iterations, using input/output queues, frames are ensured to be decoded within the maximum decoding iterations. It should be also mentioned that although the added output queue has same size as the input queue, its cost is less than the input queue. The reason is in the size of

elements of each queue: input queue elements are numbers (integer or floating-point), while output queue elements are binary numbers. For example, for a 6-bit quantized LDPC decoder, output queue has a cost equal to 1/6 of the input queue.

The rest of the paper is organized as follows. First, in Section 2, a brief review of LDPC codes and their iterative decoding are introduced. In Section 3, we describe the proposed decoder. Experimental results are given in Section 4, and final conclusions in Section 5.

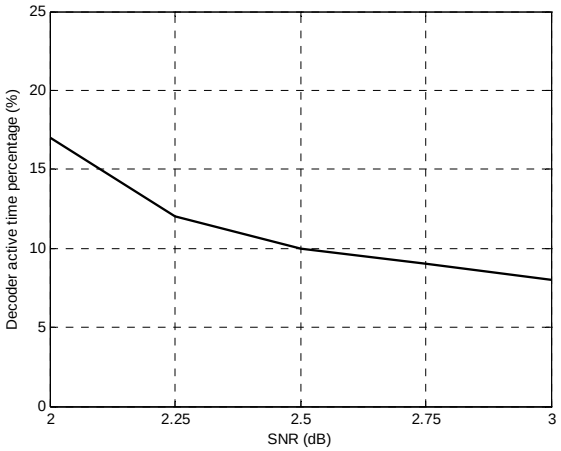

Figure 1: Example illustrates decoder active time percentage for a PEG code (1024,512) with 64 maximum iterations

2. OVERVIEW OF LDPC CODES

LDPC codes are a class of linear block codes that use a sparse, random-like parity-check matrix [1, 2]. LDPC codes can also be represented by bi-partite factor graphs having two types of nodes: *variable bit nodes* and *check nodes*, interconnected by edges whenever a given information bit appears in the parity check equation of the corresponding check bit, see figure 2. The iterative message-passing belief propagation algorithm [2,3] is used for decoding LDPC codes, and is shown to achieve optimum performance when the underlying code graph is cycle-free.

In this section, we review the belief propagation (BP) algorithm used for decoding LDPC codes presented in Gallager's work [1]. It is also called sum-product algorithm (SPA). Assume a binary (*N*,*K*) LDPC code is described by a sparse parity check matrix (called H matrix) of size $M \times N$, where M is the number of parity-checks corresponding to the parity-check nodes in a bipartite graph, and *N* is the number of variable nodes corresponding to the encoded symbols.

Before discussing the BP algorithm, we introduce some terms that will be used throughout the discussion of the algorithm [8]:

- For the j^{th} row in an H matrix, the set of column locations of the 1's is given by $R_j = \{i : h_{ji} = 1\}$. The set of column locations of the 1's, excluding location *i* is given by $R_{j\setminus i} = \{i' : h_{ji'} = 1\} \setminus \{i\}$.
- For the i^{th} column in an H matrix, the set of row locations of the 1's is given by $c_i = \{j : h_{ji} = 1\}$. The set of row locations of the 1's, excluding the location j is given by $c_{i\setminus j} = \{j' : h_{j'i} = 1\} \setminus \{j\}$
- $q_{ij}(b)$: Message (extrinsic information) to be passed from variable node v_i to check node f_j regarding the probability that $c_i = b, b \in \{0,1\}$, as shown in Figure 2(a). It equals the probability that $c_i = b$ given extrinsic information from all check nodes, except node f_j.
- $r_{ji}(b)$: Message to be passed from check node f_j to variable node v_i, which is the probability that the j^{th} check equation is satisfied given bit $c_i = b$ and the other bits have separable (independent) distribution given by $\{q_{ij'}\}_{j' \neq j}$, as shown in Figure 2(b).
- $Q_i(b)$ = the probability that $c_i = b$, $b \in \{0,1\}$
- $L(c_i) \equiv \log \frac{\Pr(x_i = +1 \mid y_i)}{\Pr(x_i = -1 \mid y_i)} = \log \frac{\Pr(c_i = 0 \mid y_i)}{\Pr(c_i = 1 \mid y_i)}$
- $L(r_{ji}) \equiv \log \frac{r_{ji}(0)}{r_{ji}(1)}$ and $L(q_{ij}) \equiv \log \frac{q_{ij}(0)}{q_{ij}(1)}$
- $L(Q_i) \equiv \log \frac{Q_i(0)}{Q_i(1)}$

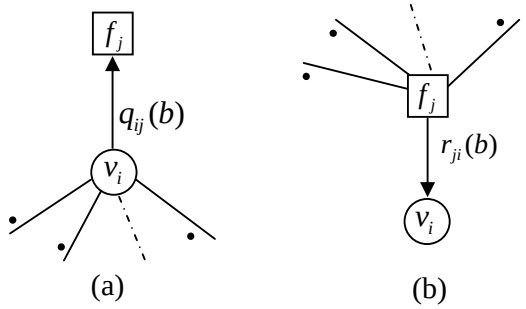

Figure 2 (a) Variable-to-check message, (b) Check-to-variable message.

The BP algorithm involves one initialization step and three iterative steps as shown below:

Initialization step: Set the initial value of each variable node signal as follows: $L(q_{ij}) \equiv L(c_i) = 2y_i / \sigma^2$, where σ^2 is the variance of noise in the AWGN channel.

Iterative steps: The three iterative steps are as follows:

(I) Update check nodes as follows:

$$L(r_{ji}) = \left(\prod_{i' \in R_{j \setminus i}} \alpha_{i'j} \right) \times \phi \left(\sum_{i' \in R_{j \setminus i}} \phi(\beta_{i'j}) \right) \quad (1)$$

Where $\alpha_{i'j} = sign(L(q_{ij}))$, $\beta_{ij} = \left| L(q_{ij}) \right|$

$$\phi(x) = -\log(\tanh(x/2)) = \log \frac{e^x + 1}{e^x - 1}$$

(II) Update variable nodes as follows:

$$L(q_{ij}) = L(c_i) + \sum_{j' \in C_{i \setminus j}} L(r_{j'i}) \quad (2)$$

(III) Compute estimated variable nodes as follows:

$$L(Q_i) = L(c_i) + \sum_{j \in C_i} L(r_{ji}) \quad (3)$$

Based on $L(Q_i)$, the estimated value of the received bit ($\hat{c}_i$) is given by:

$$\hat{c}_i = \begin{cases} 1 & if \quad L(Q_i) < 0 \\ 0 & else \end{cases}$$

Stopping Criterion: The decoded vector $\hat{c}$ is used to check if $H\hat{c}$ is zero, in which case it is declared as a valid codeword. If not, iterative decoding continues until it is eventually successful, or a preset number of maximum iterations (*m*) is reached. The check equation $H\hat{c}$ has a simple implementation in hardware, each check node perform a logical XOR function on the estimated values of the connected variable nodes, for a check node j :

$$check_j = \sum_{i \in R_j} \oplus \ \hat{c}_i$$

Then a logical OR function is performed on the results of all check nodes ($check_j$) to get the final result $H\hat{c}$.

3. DECODER DESCRIPTION

Assume we are given a conventional LDPC decoder with maximum decoding iterations (*m*). Frames arrive at the input of the decoder at a fixed frame rate (*FR*). Frames are dispatched from the decoder output at the same rate (*FR*). Usually, the maximum decoding iterations value (*m*) is chosen so that the time needed for *m* iterations equals the frame inter-arrival time, which is $1/FR$.

Our proposed idea is to improve the error correction performance of the decoder and keep frame output throughput at the same fixed rate *FR*. This is accomplished by introducing two queues, both of size *q*: one queue at the input of the LDPC decoder and the other at the output of the decoder, as shown in Figure 3.

Frames are received from channel at a fixed rate (*FR*) and stored in the input queue. The LDPC decoder receives a frame from the input queue, performs decoding iterations and then stores the decoded frame at the output queue. Frames are dispatched from the output queue at the same fixed rate (*FR*) in which frames are received at the input queue.

The decoder runs over two phases: Setup phase, in which output queue is being filled with decoded frames, and normal phase in which decoded frames are dispatched from the output queue at a fixed rate (*FR*). Following is a description for each phase:

Setup phase: when the decoding process starts running, both input and output queues are empty. As soon as the input queue receives a frame, it is dispatched to the LDPC decoder which decodes it and sends the decoded frame to the output queue. This phase continues until the decoder finishes from decoding (*q*+1) frames, *q* frames at the output queue and one frame at the decoder. Two main properties of this phase are: (1) Maximum decoding iterations is set the conventional value (*m*). (2) No frames are dispatched from the output queue. At the end of this phase, state of the queues looks like the one shown in figure 4(a).

This phase takes a starting latency of *q* frames for filling the output queue plus one frame in the decoder which is a total of (*q*+1)/*FR* sec. Comparing this latency with the conventional decoder latency of 1/*FR*, this is a disadvantage for our proposal, but is acceptable in most applications such as satellite broadcasting.

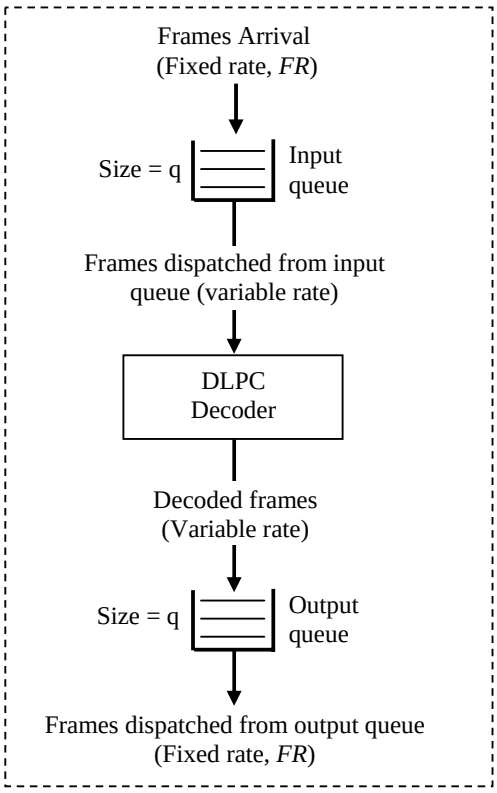

Figure 3: LDPC decoder with the input and output queues

Normal phase: In this phase, frames are dispatched from the output queue at a fixed rate *FR*. Since the input queue arrival rate has the same fixed rate *FR* as the output queue departure, the total frames in the system (decoder and queues) in any time is constant and equal to q+1 frames. At any time, there is one frame in the decoder (either being decoded or finished from decoding) and q frames distributed between input and output queues.

Because the average number of decoding iterations required per a frame is less than the conventional maximum iterations (m), most of the time the state of input/output queues is: input queue is empty and output queue is full, as shown in figure 4(a). When the decoder is in this state, an arrived new frame can use m iterations (the conventional value) plus extra iterations until the input queue becomes full. Since time for filling input queue is equal to the time for $m \times q$ iterations, the maximum possible iterations a frame can use is given by: $m + m \times q$ iterations. When a frame consumes iterations more than m, received frames are stored in the input queue, and the input queue starts growing at the same rate *RF* as the output queue is decaying, see figure 4(b). In this state, the maximum allowed decoding iterations for a given frame is given by $m \times (1 + q - n)$, where n is the number of frames in the input queue just after the start of frame decoding.

The worst case is when the input queue is full (saturated), as shown in figure 4(c). In this state, the maximum possible decoding iterations is equal to the conventional value m.

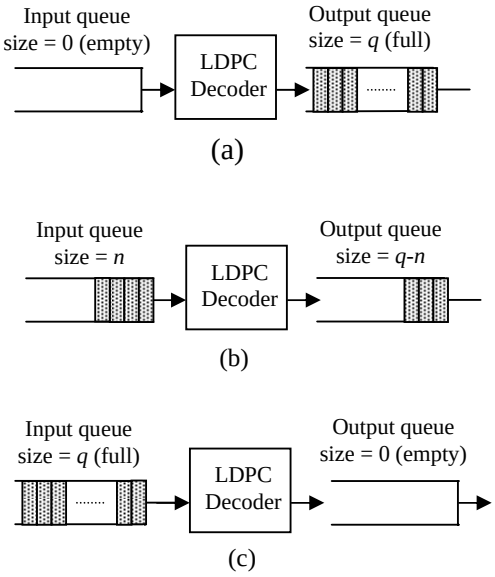

Figure 4: Three different states of input/output queues

4. EXPERIMENTAL RESULTS

In order to verify the effectiveness of the proposed idea, extensive simulations have been performed. Since simulation of LDPC is time consuming, especially at high SNR, a parallel computing simulation platform was developed to run the LDPC simulations on 170 processing nodes on a departmental LAN network.

First, we studied the effect of the maximum iterations value (m) on the performance of an LDPC decoder. We did a simulation for a PEG matrix [6] of size (1024, 512) using different maximum decoding iterations m, specifically: 16, 32, 64, 128, 256, 512 and 1024 iterations. The performance results are shown in figure 5. It is clear from the figure that the performance difference between two values of m is large when m is small. For example, the performance difference between m=16 and m=32 is much higher than the performance difference between m=512 and m=1024. Another important observation is that increasing the value of m has a more significant effect on the performance as the value of SNR increases. On the other hand, as SNR increases, average decoding iterations decreases, this increases decoder idle time and allows for efficient utilization of the proposed input/output queues scheme.

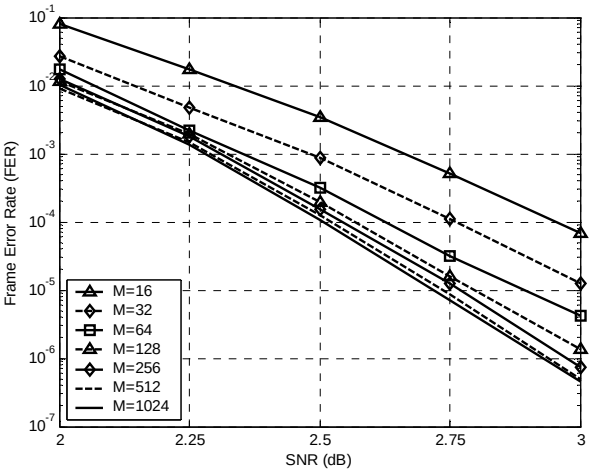

Figure 5: Effect of maximum iterations (M) on LDPC decoder performance

Figure 6 shows the effect of the proposed input/output queues scheme on LDPC decoder performance. Focusing on the two curves: m=64 without queues (conventional decoder) and m=64 with queue size of 1, it is obvious that the performance is increased remarkably without any effect on the throughput. At SNR=3dB, the conventional FER is decrease by 69% when using input/output queues of size 1 each.

The second (m=128, no queues) and third (m=64, q=1) curves in Figure 6 indicate that we could implement the proposed input/output queues to increase the LDPC decoder throughput keeping almost the same error correcting performance.

Finally, Figure 7 illustrates the effect of increasing the queues size on the performance of the LDPC decoder. It is obvious from the figure that the decoder performance increases as the size of queues increases. However, increasing the queues size has two negative impacts: increasing hardware cost and increasing initial latency time. A good balance in the choice of the values for m

and *q* could provide the best trade-off between cost and performance.

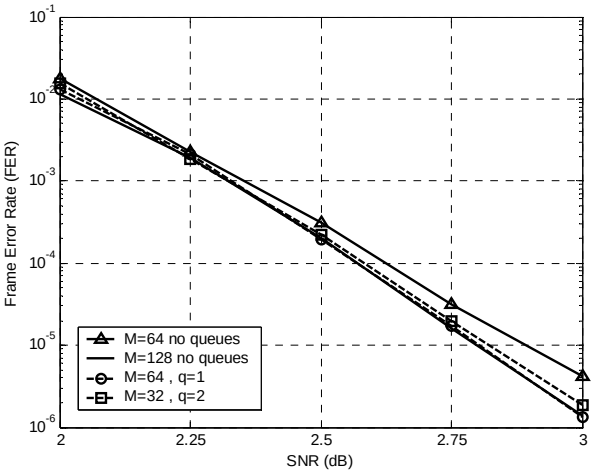

Figure 6: Effect of the proposed input/output queue scheme on LDPC decoder performance

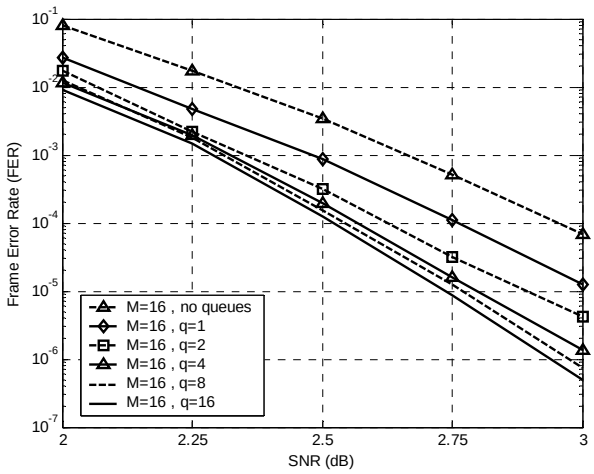

Figure 7: LDPC decoder performance using different input/output queues sizes

5. CONCLUSION

In this paper, we introduced the concept of input/output queues to increase the performance and/or throughput of an LDPC decoder. Unlike previous approaches, our proposed idea ensures that the frames are decoded within a maximum time interval, similar to the conventional decoders. This property makes our proposal practical for real time applications, such as digital video broadcasting. However, our proposal requires an initial latency time proportional to the input/output queues size, which is acceptable in most practical applications.

ACKNOWLEDGMENT

The authors thank King Fahd University of Petroleum & Minerals for support of this work.

REFERENCES

[1] R. G. Gallager, “Low Density Parity-Check Codes”. MIT Press, Cambridge, MA, 1963.

[2] D.J.C. MacKay, “Good error-correcting codes based on very sparse matrices,” *IEEE Trans. Inform. Theory*, vol.45, pp.399-431, March 1999.

[3] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, San Mateo: CA, Morgan Kaufmann, 1988.

[4] J. Vogt and A. Finger, "*Increasing throughput of iterative decoders*", ELECTRONICS LETTERS, 7th June 2001, Vol. 37, No. 12

[5] Gabriella Bosco, Guido Montorsi and Sergio Benedetto, , *"Decreasing the Complexity of LDPC Iterative Decoders"*, IEEE COMMUNICATIONS LETTERS, VOL. 9, NO. 7, JULY 2005

[6] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, “Progressive edge-growth Tanner graphs”, in *Proc. IEEE GLOBECOM 2001*, San Antonio, TX, Nov. 2001, pp. 995–1001.

[7] Massimo Rovini and Alfonso Martinez, *"On the Addition of an Input Buffer to an Iterative Decoder for LDPC Codes"*, Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th

[8] William Ryan, “A Low-Density Parity-Check Code Tutorial, Part II - The Iterative Decoder”, ECE dept. The University of Arizona, April 2002.