A Memory Efficient Serial LDPC Decoder Architecture

Abhiram Prabhakar and Krishna Narayanan
Department of Electrical Engineering,
Texas A&M University,
College Station, TX-77843
Email: {pabhiram,krn}@ee.tamu.edu

We present a memory efficient serial low density parity check(LDPC) decoder that implements a modified Sum Product Algorithm(SPA). The modification is similar to the approximate min constraint presented in [1] but differs in hardware implementation to suit a serial architecture. Our main contribution is the proposed architecture that exploits the min constraint to reduce the storage of extrinsic messages which forms the bulk of the hardware. In the proposed design the least reliable bit to check input along with the check sum are the only quantities stored in the decoder. Extrinsic message memory reduction increases with the rate of the code and up to 68% savings is achieved for a rate 9/10 code. Simulation results show that the proposed changes do not degrade the bit error rate performance.

Keywords-LDPC decoder, SPA, Memory efficient decoding, Serial hardware decoder.

I. Introduction

Recently, a number of LDPC decoder architectures have been proposed [3]- [8]. These architectures have implemented the widely used Sum Product Algorithm with various approaches to satisfy throughput and hardware requirements. The need for large hardware resources is a significant problem in the implementation of an LDPC decoder. Mostly, serial architectures that require lesser hardware than parallel implementations are used. In [4] and [5] a memory efficient turbo decoding algorithm for LDPC codes is proposed. In [11] the issues relating to the implementation of a min-sum LDPC decoder is explored. In [2] an offset min-sum algorithm offering tradeoff between performance and complexity is explored to save extrinsic message memory. In [1] an approximate min constraint for check node update is proposed. The approximation is exploited by the decoder to reduce hardware for check node computation units without any noticeable degradation in bit error rate performance. We observe that the bulk of hardware requirement for the serial LDPC decoder lies in the memory used for storing the extrinsic values(check to bit or bit to check) and hence attempt to reduce it. The proposed modification to the SPA explained in section III is identical to the one proposed in [1] but we implement the approximation in log-tanh domain to facilitate our serial decoder architecture. The proposed decoder stores only few bit to check messages which is very efficient when compared to architectures proposed in [6], [7] which store all the bit to check messages and also check to bit messages. When compared to the $\lambda - min$ serial decoder [2] our design stores lesser number of extrinsic values and requires lesser memory to store intermediate partial sums. The proposed decoder is explained in section IV.

II. Decoding of LDPC codes

Let us assume a BPSK modulation scheme where a 1 is mapped to $+1$ and 0 is mapped to -1 and an Additive White Gaussian Noise(AWGN) channel. The received channel values are converted to log-likelihood ratios given by, $L_{ch}(c_i) = \frac{-2}{\sigma^2} r_i$, where σ^2 is the variance of the noise added in the channel. Let us define the function, $\psi(x) = \log(\tanh(\frac{|x|}{2}))$.

Each decoding iteration has a bit node update and a check node update. Let $L_c^q(i)$ represent the check to bit message along the i^{th} edge connected to the n^{th} check node during the q^{th} iteration(we will not explicitly use the index n to denote quantities associated with the n^{th} node as the operations are identical at all nodes). Similarly, let $L_b^q(i)$ represent the bit to check value along the i^{th} edge connected to the n^{th} bit node.

Enforcing the parity check constraint on the incoming bit to check values, the check node update is given by,

$$|L_c^q(i)| = \psi^{-1}\left(\sum_{k=1,k\neq i}^{k=t} \psi(L_b^{q-1}(k))\right), \forall i \quad (1)$$

$$sign(L_c^q(i)) = \prod_{k=1,k\neq i}^{k=t} sign(L_b^{q-1}(k)) \quad (2)$$

Where t is the degree of the check node and index k refers to the k^{th} edge connected to the check node. For serial implementation in hardware, Eq. 2 is divided into two steps. First, we find two quantities $M1$ and $S1$.

$$M1 = \sum_{k=1}^{k=t} \psi(L_b^{q-1}(k)) \qquad S1 = \prod_{k=1}^{k=t} sign(L_b^{q-1}(k)) \quad (3)$$

Note that by definition of ψ and $M1$ are always negative. Next, we find $M2(i)$ and $S2(i)$ for all the edges $i = 1$ to t

according to

$$M2(i) = M1 - \psi(L_b^{q-1}(i)) \qquad S2(i) = S1 \times sign(L_b^{q-1}(i)) \tag{4}$$

Now,

$$L_c^q(i) = S2(i) \times \psi^{-1}(M2(i)) \tag{5}$$

The soft output for the n^{th} bit is given by,

$$L_{soft}^n = L_{ch}(c_n) + \sum_k L_c^q(k) \tag{6}$$

The bit node update is given by,

$$L_b^{q+1}(i) = L_{ch}(c_n) + \sum_{k \neq i} L_c^q(k) \tag{7}$$

Alternatively, the bit node update can be written as,

$$L_b^{q+1}(i) = L_{soft}^n - L_c^q(i) \tag{8}$$

The n^{th} bit is decoded as 1 if $L_{soft}^n < 0$, else as 0.

III. Proposed Modifications

We propose to modify $M2(i)$ given by Eq. 4. Among the edges $1, 2, ...t$ connected to a check node, let the j^{th} edge have the lowest magnitude of L_b. That is $j = \arg\min_i |L_b(i)|$. Then $M2(j)$ is given by,

$$\begin{aligned} M2(j) &= M1 - \psi(L_b^{q-1}(j)) && (9)\\ M2(i) &= M1 \quad \text{for all other edges}(i \neq j) && (10) \end{aligned}$$

Our proposed modification is identical to the one proposed in [1] but our serial decoder implementation exploits this approximation to save extrinsic memory and is computed in the log-tanh domain. The approximation as defined in [1] is computed directly on LLR values and this would require more memory than our proposed design for a serial decoder and the reason is explained in section IV. If eq(9) was used in computing $M2(i)$ for all the edges of a check node then this would be the normal SPA, a serial implementation of which would need memory to store $M1$ for each check node and memory for storing all the bit to check values to compute $M2(i)$ and thereby the L_c values. Due to the proposed modification and the proposed serial decoder architecture that exploits the modification, we need memory to store $M1$ and only the least reliable bit to check value for each check node. From the bit error performance plots in Fig. 1, we can see that the proposed modification to the SPA algorithm leads to no noticeable loss in performance and performs better than min-sum decoding which is another possible approximation to reduce the memory requirement. The reason for this behavior is the the under estimation of check to bit messages and we shall explain this in the following section.

The following can be noted about the $\psi(x)$ function - (i) $\psi(x)$ is always negative, (ii) $|\psi(x)|$ is a decreasing function of $|x|$ and finally, (iii) the function $\psi(x)$ has a large slope for $0 < |x| < 2$ (equivalently, the function $\psi^{-1}(x)$ has a shallow slope in the range $0 < \psi^{-1}(x) < 2$ and hence, is quite insensitive to x in that range. Let us call this as the 'flat region'). Due to (i), $M1 = \sum \psi(L_b(i))$ is always negative (henceforth, we shall refer to quantities $M1$ and $M2$ only by their magnitudes). Due to (ii), we can see that highly reliable inputs to the check i.e. the inputs with large value of $|L_b|$ contribute little to $M1$ (since $\psi(L_b)$ is very small) and the less reliable L_b inputs contribute larger values to $M1$. Hence $M1$ will be small when all the bit to check values are reliable and large when some of them are unreliable. In our proposed modification we do not subtract $\psi(L_b(i))$ from $M1$ in finding $M2(i)$ except for the least reliable edge. Since $\psi(L_b(i))$ is small for the reliable edges, the approximation error in not subtracting $\psi(L_b(i))$ is small. Further note that if at least one of the incoming edges is unreliable, the overall value of $M1$ will be in the flat region and, hence, any small approximation error will not cause a large error in $|L_c(i)| = \psi^{-1}(M2(i))$. If all the incoming edges are reliable, then M1 will be small and the approximation error will be higher, but interestingly, when the incoming edges are all reliable, the outgoing messages are also fairly reliable and, hence, such an approximation error will not significantly affect the iterative decoding algorithm.

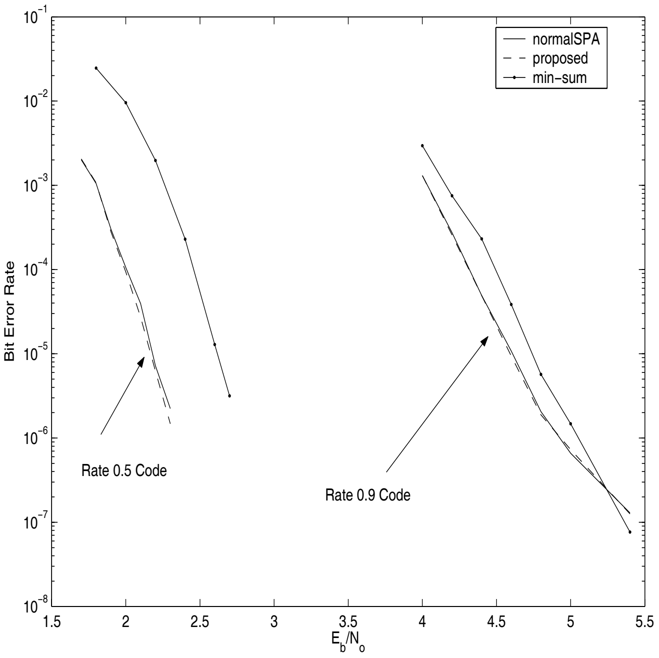

Fig. 1. E_b/N_o vs BER for a rate 0.5 and rate 0.9 code of length 2040 after 30 decoding iterations.

Note that for a given set of $L_b(i)$'s, for the least reliable edge, $M2_{prop}(j) = M2_{SPA}(j)$ and along all other edges $M2_{prop}(i) > M2_{SPA}(i)$, where $M2_{prop}$ and $M2_{SPA}$ are the $M2$ values with the proposed algorithm and with the SPA, respectively. Since higher values of $M2$ correspond to *less* reliable extrinsic messages, the proposed algorithm actually provides pessimistic estimates for $L_c(i)$ compared to the SPA. This is particularly an interesting feature of the algorithm since not subtracting the *a priori* information in the LLR domain (for example, at the bit note update) can result in more optimistic extrinsic values, which can cause error propagation. Hence, the approximation in the $\psi(x)$ domain is an important feature of this algorithm. Fig. 2 shows a plot of $L_c(i)$ values after iteration 1 for a rate 0.5 regular LDPC code at an E_b/N_0 of 2.2 dB with the all zero sequence being transmitted. As iterations progress the least reliable input to a check might change but the $L_c(i)$ values for the proposed scheme stay close to that of a normal SPA decoder (the plot is not shown due to space limitations.)

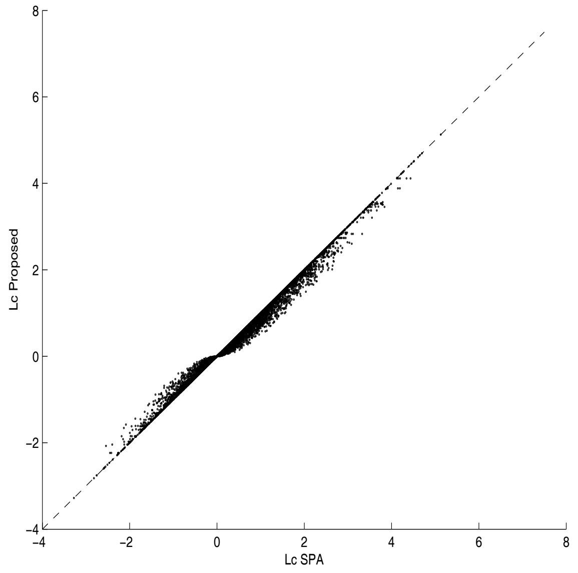

Fig. 2. Check to Bit values for the proposed scheme after iteration 1.

It is interesting to contrast the proposed algorithm with the min-sum algorithm, which is another approximation of the SPA and can be implemented with lower memory in hardware. For a min-sum decoder $M2_{minsum}(i) < M2_{SPA}(i)$. Hence, the extrinsic messages from the check node are always optimistic compared to SPA decoding($|L_{cminsum}(i)| > |L_{cSPA}(i)|$). This positive feedback appears to propagate errors in the iterative decoder resulting in a worse BER than the proposed algorithm.

We can apply the proposed modification for the least two reliable edges instead of one. This would require more memory as we need to store the bit to check values along these two edges instead of one. Since the performance of the proposed scheme is good and our objective is to minimize the hardware requirement we restrict ourself to implementing the changes only along one edge.

IV. Proposed Serial Decoder Implementation

A serial LDPC decoder is one that computes one check update or one bit update during a clock cycle. The decoder architecture should be designed to effectively use the modification to the SPA algorithm to minimize extrinsic message storage. Depending on the architecture either bit to check extrinsic messages or check to bit extrinsic messages or both of them are stored at the decoder. We shall design our proposed serial SPA decoder to store only one type of messages. Let us first consider a serial decoder design that uses the approximate min algorithm as in [1] that directly works on bit to check messages in the LLR domain. This would require processing one check update at a time. First, the magnitude of bit to check messages at a check are compared to find the least one. Next, the algorithm finds two check to bit messages, one for the least bit to check edge and another for the other edges. The decoder would need to store these two check to bit extrinsic messages for each check. These check to bit messages would update the corresponding partial bit sums according to Eq. 6(The bit sum memory is used to store partial L_{soft} values for each bit node). The bit sum values and the stored check to bit messages would be later used to compute the bit to check messages according to Eq. 8 and hence eliminate the necessity to store bit to check messages. Hence the decoder would require memory to store the partial bit sum updates for the current iteration and memory to hold the bit sum from previous iteration to compute bit to check messages for the current iteration. The size of each bit sum memory is proportional to the length of the code and is independent of the rate of the code. Suppose we can store the check sums($M1$) instead of bit sums, then the size of this memory would be proportional to the number of checks and hence would decrease with the rate of the code. Also we would like to store only one extrinsic message per check instead of two. These requirements can be met if we compute one bit update at a time and use the bit to check messages to update partial check sums and hence distribute $M1$ computation over the length of the code. Since we do not compute one check at a time the algorithm in [1] cannot be used directly and hence we implement the min constraint in the log-tanh domain. In the log-tanh domain once $M1$ has been computed for all the checks, $M2$ can be computed easily by subtracting the bit to check message from $M1$. In the LLR domain the check to bit message cannot be computed from a quantity that includes all the bit to check messages. Hence, implementation of the min constraint in the log-tanh domain as proposed in this paper becomes a necessity for any decoder design that does not compute all the check to bit messages for a check node together. In our design $M1$ values for all the check nodes are computed first and simultaneously, only the least reliable bit to check input to each check node is stored. After this is done, $M2$ values are computed using equation 9 or 10.

Fig. 3 shows the block diagram of the proposed serial decoder for a $(3, k)$ code. The FIFO stores the sign of bit to check messages (L_b) from the previous iteration and simultaneously, the $(S1, M1)$ values for the checks in which the bit participates are updated and stored in memory blocks(RAMs). The memory blocks in addition to $(S1, M1)$ also store the least reliable bit to check input L_{min} and Cnt. Cnt identifies the edge corresponding to L_{min} and is used for generating the select signal in the multiplexer to select between $M1$ or the subtracted value for $M2$. The comparator compares the least reliable bit to check message stored in the RAM for a check node with the incoming value and updates L_{min}. It can be seen that L_b values update the corresponding $M1$ values one at a time and the final $M1$ values are obtained only after all the L_b values have updated the corresponding $M1$ values. Hence, the computation of $M1$ values is distributed over the length of the code and not done one at a time. Once $M1$ is computed for all the check nodes, Cnt and L_{min} can be used to compute $M2$ according to eq(9) or eq(10). Two sets of memory banks can be used so that when partial $M1$ values are updated in one memory bank $M2$ values can be formed from the other memory bank that holds the $M1$ values from previous iteration. In a normal serial SPA decoder we need to have stored both the sign and magnitude of all the incoming L_b messages in the FIFO to compute $M2$ and hence there is a significant reduction in memory requirement for the proposed design. The extra hardware required for comparator, multiplexer, mux select etc are small as only one of them is there in each path. In our previous work [8], we have shown that the serial architecture can be easily scaled to a partly parallel architecture with various levels of parallelization. The approximate min scheme in log-tanh domain can be easily implemented on each parallel path. The proposed technique can also be easily incorporated into a partly parallel LDPC decoder [8] [9] to save extrinsic memory.

V. Results and Conclusions

Here we present some examples of savings in extrinsic memory for the proposed decoder design. Let us consider a design where the LLR and ψ values have 5 bit precision(1 for sign and 2 for magnitude and 2 for decimal. Let $(S1, M1)$ have 6 bits precision. We first look at a $(2040, 204, 3, 30)$ rate 0.9 code. In the proposed decoder the FIFO stores only the sign of L_b messages and hence require only $2040 \times 3 = 6120$ bits. The RAMs, in addition to $M1$ have to store L_{min} which is 3 bits and Cnt which is 5 bits(each check has 30 incoming L_b values and hence $\log_2(30)$) for each check node and hence require $204 \times 8 = 1632$ bits. The total memory required is 7752 bits. A normal SPA decoder with similar architecture would need $2040 \times 3 \times 4 = 24480$ bits for the FIFO. Hence there is a 68% reduction in extrinsic memory for the proposed design. For a rate 0.5 code of same length a 50% reduction is achieved. Most of the architectures listed in the references require memory in the order of the number of extrinsic values plus memory in the order of the length of the code/number of checks. When compared to the serial λ-min decoder [2], our serial decoder requires even lesser memory. We store only one extrinsic messages and identifier per check whereas [2] stores λ extrinsic messages and identifiers per check. Also the partial check sum memory which decreases with the rate of the code is much less when compared to the partial bit sum memory used in the λ-min decoder. In our design the reduction in memory increases with the rate of the code since the number of checks decrease. Simulation results for codes of different rates showed no noticeable loss in performance and hence we have presented a novel memory efficient serial LDPC decoder.

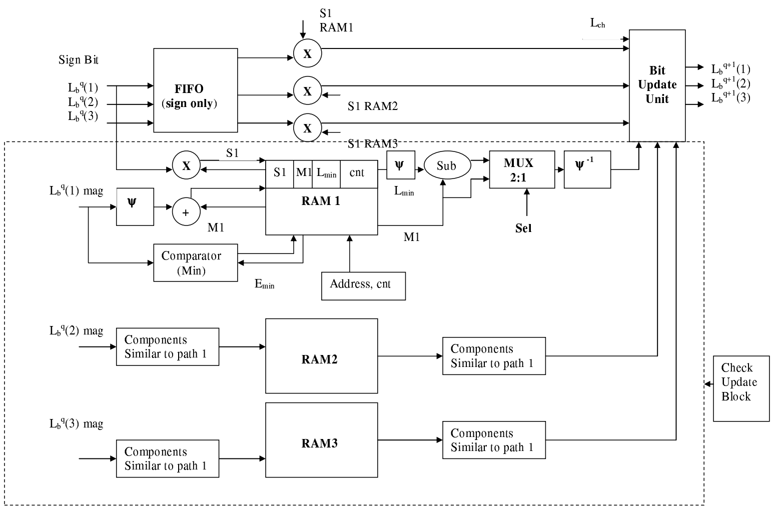

Fig. 3. Block diagram of the proposed Serial LDPC decoder.

References

[1] C. Jones, E. Valles, M. Smith, J. Villasenor. Approximate-Min Constraint Node Updating for LDPC code design. *IEEE conference on Military Communications, 2003. MILCOM 2003*, 13-16 Oct 2003, pages:57-162.

[2] F. Guilloud, E. Boutillon, J.L. Danger. λ-Min Decoding Algorithm of Regular and Irregular Codes. *Proceedings of the 3nd International Symposium on Turbo Codes & Related Topics*,Brest,France, Sept 2003.

[3] S. Sivakumar. VLSI Implementation of Encoder and Decoder for Low Density parity check codes. *Masters Thesis,Texas A&M University*,December 2001.

[4] M.M. Mansour, N.R. Shanbhag. Memory efficient turbo decoder architecture for LDPC codes. *IEEE workshop on Signal Processing Systems, 2002(SIPS'02)*, 16-18 Oct 02, pages:159-164.

[5] H. Sankar, K. R. Narayanan. Memory-Efficient Sum-Product Decoding of LDPC Codes. *To appear in IEEE Tansactions on Communications.*

[6] E. Yeo, P. Pakzad, B. Nikolic, V. Anantharaman. High throughput Low-Density Parity-Check decoder architectures. *IEEE Global Telecommunication Conference,2001. GLOBECOM'01*, vol:5, pages:3019-3024.

[7] T. Zhang, K.K. Parhi. A 54 MBPS (3,6)-regular FPGA LDPC decoder. *IEEE Proc. of SIPS*, pp.127-132, 2002.

[8] A. Selvarathinam, G. Choi, K. Narayanan, A. Prabhakar, E. Kim. A massively scalable architecture for low-density parity-check codes. *Circuits and Systems 2003, ISCAS' 03*, Volume 2, 25-28 May 2003. Pages:II-61-II64 vol.2.

[9] K. Gunnam, G. Choi, M. Yeary. An LDPC decoding shedule for memory access reduction. *Acoustics, Speech, and Signal Processing, 2004. ICASSP 04*, vol:5, 17-21 May 2004, pages:173-176.

[10] Y. Chen, D.Hocevar. A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder. *IEEE Global Telecommunciations Conference, 2003. GLOBECOM'03*, vol:1, 1-5 Dec.2003, pages:113-117.

[11] F. Zarkeshvari, A.H. Banihashemi. On implementation of min-sum algorithm for decoding low-density parity-check(LDPC) codes. *IEEE Global Telecommunications Conference,2002. GLOBECOM'02*, vol:2, Nov 17-21, 2002, pages:1349-1353.