APPROXIMATE-MIN* CONSTRAINT NODE UPDATING FOR LDPC CODE DECODING

Christopher Jones, Esteban Vallés, Michael Smith, and John Villasenor
Electrical Engineering Department, University of California, Los Angeles, CA, 90095, USA
{christop, evalles, miksmith, villa}@icsl.ucla.edu

ABSTRACT

This work introduces a technique for updating messages that originate at the constraint nodes of bi-partite graphs in Low-Density Parity-Check codes. The technique computes only two outgoing magnitudes at each constraint node and exhibits no measurable performance loss as compared to exact belief propagation which computes a unique magnitude for each departing edge from a given constraint node. The technique eliminates the need for memory based table look-up in the constraint node processing and has been implemented using only shift, add, and comparison operations. Finite wordlength results for a 'worst-case' code indicate that 6 bit quantization yields performance that is similar to that provided by full floating point computation.

I. INTRODUCTION

Low-density parity-check (LDPC) codes were proposed by Gallager in the early 1960s [1]. The structure of Gallager's codes (uniform column and row weight) led them to be called regular LDPC codes. Gallager provided simulation results for codes with block lengths on the order of hundreds of bits. However, these codes were too short for the sphere packing bound to approach Shannon capacity, and the computational resources for longer random codes were decades away from being broadly accessible.

Following the groundbreaking demonstration by Berrou *et al.* [2] of the impressive capacity-approaching capability of long random linear (turbo) codes, MacKay [3] re-established interest in LDPC codes during the mid to late 1990s. Luby *et al.* [4] formally showed that properly constructed irregular LDPC codes can approach capacity more closely than regular codes. Richardson, Shokrollahi and Urbanke [5] created a systematic method called density evolution to analyze and synthesize the *degree distribution* in asymptotically large random bipartite graphs under a wide range of channel realizations.

In his original work, Gallager also introduced several decoding algorithms. One of these algorithms has since been identified for general use in factor graphs and Bayesian networks [6] and is often generically described as Belief Propagation (BP). In the context of LDPC decoding, messages handled by a belief propagation decoder represent probabilities that a given symbol in a received codeword is either a one or a zero. These probabilities can be represented absolutely, or more compactly in terms of likelihood ratios or likelihood differences. The logarithmic operator can also be applied to either of these scenarios. Due to the complexity of the associated operator sets and wordlength requirements, the log-likelihood ratio form of the Sum-Product algorithm is the form that is best suited to VLSI implementation. However, this form still posses significant processing challenges as it employs a non-linear function that must be represented with a large dynamic range for optimal performance. The aim of this paper is to manipulate and simplify the most challenging portions of the log-likelihood version of belief propagation. Throughout the rest of the paper, unaltered log-likelihood belief propagation with be described as Full-BP.

Even Full-BP algorithms suffer performance degradation as compared to the optimum ML decoder for a given code. This is due to the fact that bipartite graphs representing finite-length codes without singly connected nodes are inevitably non-tree-like. Cycles in bipartite graphs compromise the optimality of belief propagation decoders. The existence of cycles implies that the neighbors of a node are not in general conditionally independent (given the node), therefore graph separation does not hold and Pearl's polytree algorithm [6] (which is analogous to Full-BP decoding) inaccurately produces graph a-posteriori probabilities. Establishing the true ML performance of LDPC codes with length beyond a few hundred bits is generally viewed as an intractable problem. However, code conditioning techniques [7] can be used to mitigate the non-optimalities of iterative decoders and performance that approaching the Shannon capacity is achievable even with the presence of these decoding non-idealities.

The next section provides background on LDPC representation with bi-partite graphs and matrices over GF(2). Section III is presented in several parts. In the first part, our modified version of Full-BP is introduced and its performance is contrasted to that of Full-BP. Next, we provide a discussion of the steps taken in the derivation of this technique. At the end of Section III a complexity comparison between the proposed technique, Full-BP, and a previously proposed reduced complexity technique [8] is provided. Section IV gives a discussion of finite precision issues and provides performance data for several quantization schemes. Section V describes the LDPC codec that has been developed at UCLA based on the constraint update technique of this paper. Concluding remarks are made in section VI.

II. LOW-DENSITY PARITY-CHECK CODES

Like turbo codes, LDPC codes belong to the class of codes that are decodable primarily via *iterative* techniques. The demonstration of capacity approaching performance in turbo codes stimulated interest in the improvement of Gallager's original LDPC codes to the extent that the performance of these two code types is now comparable in AWGN. The highly robust performance of LDPC codes in other types of channels such as partial-band jamming, quasi-static multi-input multi-

Funded by Office of Naval Research Contract N00014-01-C-0016

0-7803-8140-8/03/$17.00 (C) 2003 IEEE

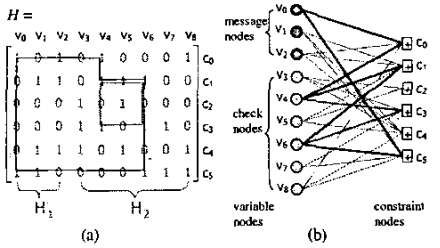

Fig. 1. Matrix and graph descriptions of a (9, 3) code. A length 4 and a length 6 cycle are circumscribed with bold lines.

ouput (MIMO) Rayleigh fading, fast MIMO Rayleigh fading, and periodic fading is evidenced in [9] and [10].

LDPC codes are commonly represented as a bipartite graph (see Fig. 1b). In the graph, one set of nodes, the *variable* nodes, correspond to the codeword symbols and another set, the *constraint* nodes, represent the constraints that the code places on the variable nodes in order for them to form a valid codeword. *Regular* LDPC codes have bipartite graphs in which all nodes of the same type are of the same degree. A common example is the (3,6) regular LDPC code where all variable nodes have degree 3, and all constraint nodes have degree 6. The regularity of this code implies that the number of constraint nodes (which is the same as the number of parity check bits) equals exactly half the number of variable nodes such that the overall code is rate 1/2.

The parity check matrix H of a linear binary $(n,\ k)$ systematic code has dimension $(n-k) \times n$. The rows of H comprise the null space of the rows of the code's $k \times n$ generator matrix G. H can be written as,

$$H = \left[\begin{array}{cc} H_1 & H_2 \end{array} \right], \tag{1}$$

where H_1 is an $(n-k) \times k$ matrix and H_2 is an $(n-k) \times (n-k)$ matrix. H_2 is constructed to be invertible, so by row transformation through left multiplication with H_2^{-1}, we obtain a systematic parity check matrix H_{sys} that is range equivalent to H,

$$H_{sys} = H_2^{-1}H = \left[\begin{array}{cc} H_2^{-1}H_1 & I_{n-k} \end{array} \right]. \tag{2}$$

The left-hand portion of which can be used to define a null basis for the rows of H. Augmentation of the left-hand portion of the systematic parity check matrix H_{sys} with I_k yields the systematic generator matrix,

$$G_{sys} = \left[\begin{array}{cc} I_k & \left(H_2^{-1}H_1\right)^T \end{array} \right]. \tag{3}$$

The rows of G_{sys} span the codeword space such that $G_{sys}H^T = G_{sys}H_{sys}^T = 0$. It should be noted that although the original H matrix is sparse, neither H_{sys} nor G_{sys} is sparse in general. G_{sys} is used for encoding and the sparse parity matrix H is used for iterative decoding. A technique that manipulates H to obtain a nearly lower triangular form and allows essentially linear time (as opposed to the quadratic time due to a dense G matrix) encoding is available and was proposed by [11].

The matrix and graph descriptions of an $(n = 9, k = 3)$ code are shown in Fig. 1. Structures known as cycles, that affect decoding performance, are shown by (bold) solid lines in the figure. Although the relationship of graph topology to code performance in the case of a specific code is not fully understood, work exists [7] that investigates the effects of graph structures such as cycles, stopping sets, linear dependencies, and expanders.

III. Reducing the Complexity of Full-BP Decoding

A. The Approximate-Min*-BP technique

Before describing the technique, we introduce notation that will be used in the remainder of the paper. On the variable node (left-hand) side of the bi-partite graph, u messages arrive and V messages depart. At the constraint node (right-hand) side of the graph v messages arrive and U messages depart. All four message types are actually log-likelihood ratios (LLRs). For instance, a message v arriving at a constraint node is actually a shorthand representation for $v = ln(p(v=0)/p(v=1))$. The constraint node *a-posteriori probability*, or U^{APP}, is defined as the constraint node message determined by the d_c variable messages that arrive at a constraint node of degree-d_c. The notation $U_j = U^{APP} \backslash v_j$ denotes the outgoing constraint message determined by all incoming edges with the exception of edge v_j. Message v_j represents *intrinsic* information that is left purposefully absent in the *extrinsic* message computation $U_j = U^{APP} \backslash v_j$. Our algorithm (called Approximate-Min*-BP, or A-Min*-BP) updates constraint messages as follows,

$$\begin{aligned}
&\text{initialize } \left\{ v_{\min} = \min_{j=1..d_c} \left(|v_j|\right), \delta^0 = \infty \right\} \\
&\text{for k =1 } \ldots d_c \\
&\quad \text{if } (k \neq min) \\
&\qquad \delta^k = \left| \Lambda^{BP^*} \left(\delta^{k-1}, v_k\right) \right| \\
&\quad \text{else:} \\
&\qquad \delta^k = \delta^{k-1} \\
&\text{end} \\
&\delta^{APP \backslash v_{\min}} = \delta^{d_c} \\
&\delta^{APP} = \left| \Lambda^{BP^*} \left(\delta^{APP \backslash v_{\min}}, v_{\min}\right) \right| \\
&\sigma^{APP} = \prod_{j=1}^{d_c} \operatorname{sgn}(v_j)
\end{aligned}$$

where δ^k is a storage variable, and Λ^{BP^*} will be defined shortly. Constraint message updates are found by applying the following additional operations on the above quantities.

$$\begin{aligned}
\underset{j=\{1..d_c\} \backslash \min}{U_j} &= \operatorname{sgn}(v_j) \operatorname{sgn}(\sigma^{APP}) \delta^{APP} \\
U_{\min} &= \operatorname{sgn}(v_{\min}) \operatorname{sgn}(\sigma^{APP}) \delta^{APP \backslash v_{\min}}
\end{aligned}$$

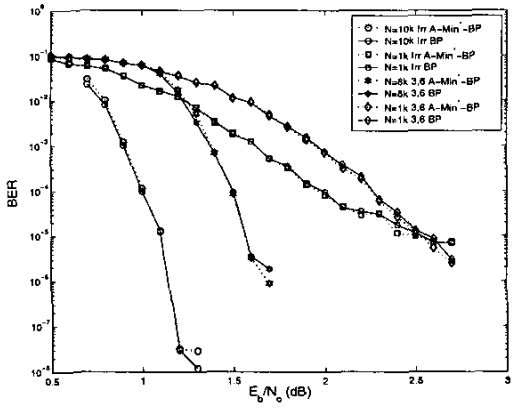

Fig. 2. A-Min*-BP decoding compared to Full-BP decoding for four different rate 1/2 LDPC codes. Length 10k irregular max left degree 20, length 1k irregular max left degree 9, length 8k regular (3,6), length 1k regular (3,6). The proposed algorithm incurs negligible performance loss. Note that rate 1/2 BPSK constrained capacity is 0.18 dB (E_b/N_o). All simulations were performed with the same initial random seed.

The above constraint node update equations are novel and will be described in further detail. Variable node updating in our technique is the same as in the case of Full-BP. Extrinsic information is similarly described as before via $V_j = V^{APP} \backslash u_j$, however the processing required to achieve these quantities is much simpler,

$$V^{APP} = \sum_{j=0}^{d_v} u_j \qquad \underset{j=1..d_v}{V_j} = V^{APP} - u_j, \tag{4}$$

where d_v is the variable node degree.

B. Derivation and complexity of the A-Min*-BP technique

Derivation of the Approximate-Min*-BP constraint node update begins with the so called 'Log-Hyperbolic-Tangent' definition of BP constraint updating. In the equation below, sign and magnitude are separable since the sign of LnTanh(x) is determined by the sign of x.

$$U^{APP} = \left[\prod_{j=1}^{d_c} \mathrm{sgn}(v_j)\right] \ln\left(\frac{1 + e^{-\sum_{j=1}^{d_c} \ln\left(\frac{1+e^{-|v_j|}}{1-e^{-|v_j|}}\right)}}{1 - e^{-\sum_{j=1}^{d_c} \ln\left(\frac{1+e^{-|v_j|}}{1-e^{-|v_j|}}\right)}}\right). \tag{5}$$

This equation is highly non-linear and warrants substantial simplification before mapping to hardware. To begin, the above computation can be performed by first considering the *inner* recursion in (5),

$$\sum_{j=1}^{d_c} \ln\left(\frac{1 + e^{-|v_j|}}{1 - e^{-|v_j|}}\right). \tag{6}$$

A total of d_c table look-ups to the function $\Lambda^{lnBP} = \ln\left(\frac{1+e^{-|v_j|}}{1-e^{-|v_j|}}\right)$ followed by $d_c - 1$ additions complete the computation in (6). Furthermore, the linearity of the inner recursion allows intrinsic variable values to be 'backed-out' of the total sum before d_c *outer* recursions are used to form the d_c extrinsic outputs. To summarize, computation of all d_c extrinsic values (in (5)) follow from d_c table look-ups, $d_c - 1$ additions, d_c subtractions, and a final d_c table look-ups. The cost of computing the extrinsic sign entails $d_c - 1$ exclusive-or operations to form the APP extrinsic sign, followed by d_c incremental exclusive-or operations to back-out the appropriate intrinsic sign to form each final extrinsic sign.

Variable node computation (4) is more straightforward. However, a possible alternative to (4) is given in [12] where it is noted that codes lacking low degree variable nodes experience little performance loss due to the replacement of V_j with V^{APP}. However, codes that maximize rate for a given noise variance in an AWGN channel generally have a large fraction of degree-2 and degree-3 variable nodes [5]. Low degree nodes are substantially influenced by any edge input and V^{APP} may differ significantly from corresponding properly computed extrinsic values. We have found experimentally that using V^{APP} alone to decode capacity approaching codes degrades performance by one dB of SNR or more.

We continue toward the definition of an alternative constraint update recursion by rearranging (5) for the $d_c = 2$ case,

$$\mathrm{sgn}(v_1)\mathrm{sgn}(v_2) \ln\left(\frac{1 + e^{|v_1|+|v_2|}}{e^{|v_1|} + e^{|v_2|}}\right) = \ln\left(\frac{1 + e^{v_1+v_2}}{e^{v_1} + e^{v_2}}\right). \tag{7}$$

Two applications of the Jacobian logarithmic identity $(\ln(e^a + e^b) = \max(a,b) + \ln\left(1 + e^{-|a-b|}\right))$ [13] result in the Min* recursion that is discussed in the rest of the paper,

$$\Lambda^{BP^*}(v_1, v_2) = \mathrm{sgn}(v_1)\mathrm{sgn}(v_2) \begin{pmatrix} \min\left(|v_1|, |v_2|\right) \\ +\ln\left(1 + e^{-(|v_1|+|v_2|)}\right) \\ -\ln\left(1 + e^{-||v_1|-|v_2||}\right) \end{pmatrix}. \tag{8}$$

Note that (8) is not an approximation. It is easy to show that $d_c - 1$ recursions on Λ^{BP^*} yield exactly U^{APP} in equation (5). Furthermore, the function $\ln\left(1 + e^{-|x|}\right)$ ranges over $(ln(2), 0)$ which is substantially more manageable than the range of the function Λ^{LnBP}, $Range\left(\ln\left(\frac{1+e^{-|x|}}{1-e^{-|x|}}\right)\right) = (\infty, 0)$ from a numerical representation point of view. However, the non-linearity of the recursion (8) implies that updating all extrinsic information at a constraint node requires $d_c\,(d_c - 1)$ calls to Λ^{BP^*}. This rapidly becomes more complex than the $2d_c$ look-up operations (augmented with $2d_c - 1$ additions) required to compute all extrinsic magnitudes based on the form in (5). Again, in this earlier case intrinsic values can be 'backed-out' of a single APP value to produce extrinsic values.

Instead of using the recursion in (8) to implement Full-BP we propose that this recursion be used to implement an approximate BP algorithm to be referred to as *Approximate-Min*-BP* (A-Min*-BP). The algorithm works by computing the proper extrinsic value for the minimum magnitude (least reliable) incoming constraint edge and assigning the U^{APP}

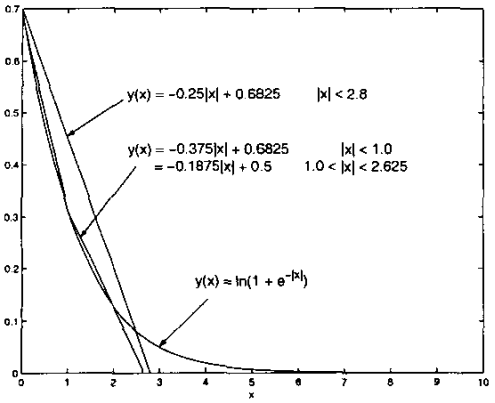

Fig. 3. Non-linear function (Λ^{BP^*}) at the core of proposed algorithm and single / double line approximations to the function that can easily be implemented in combinatorial logic.

magnitude in conjunction with the proper extrinsic sign to all other edges.

To provide intuition as to why this hybrid algorithm yields good performance, note first that a constraint node represents a single linear equation and has a known 'solution' if no more than one input variable is unknown. Consider the following two scenarios. First, if a constraint has more than one unreliable input, then all extrinsic outputs are unreliable. Second, if a constraint has exactly one unreliable input, then this unknown input can be solved for based on the extrinsic reliability provided by the 'known' variables. In this second case all other extrinsic updates are unreliable due to the contribution of the unreliable input. The approximation in the suggested algorithm assigns less accurate magnitudes to would-be unreliable extrinsics, but for the least reliable input preserves exactly the extrinsic estimate that would be produced by Full-BP.

We next show that U^{APP} always underestimates extrinsics. Here the notation U_{mn} represents the extrinsic information that originates at constraint node m and excludes information from variable node n. Rearrangement of (5) (with standard intrinsic/extrinsic notation included [14]) yields the following,

$$|U^{APP}| = \ln \frac{1 + \prod_{n' \in N(m)} \frac{1-e^{-|v_{mn'}|}}{1+e^{-|v_{mn'}|}}}{1 - \prod_{n' \in N(m)} \frac{1-e^{-|v_{mn'}|}}{1+e^{-|v_{mn'}|}}} \tag{9}$$

$$\frac{1-e^{-|U^{APP}|}}{1+e^{-|U^{APP}|}} = \left(\prod_{n' \in N(m) \setminus n} \frac{1-e^{-|v_{mn'}|}}{1+e^{-|v_{mn'}|}} \right) \left(\frac{1-e^{-|v_{mn}|}}{1+e^{-|v_{mn}|}} \right). \tag{10}$$

Note first that the function $g(x) = \frac{1-e^{-|x|}}{1+e^{-|x|}}$ (a product of which comprises the RHS of (10)) ranges over $(0, 1)$ and is non-decreasing in the magnitude of x. The first (parenthesized) term on the right-hand side of (10) equals the extrinsic value U_{mn} under the operator $g(\cdot)$, i.e. $g(U_{mn})$. The second term scales this value by the intrinsic reliability $g(v_{mn})$. Hence, the monotonicity and range of $g(x)$ ensure that $|U^{APP}| < |U_{mn}|$. We provide the inverse function, $g^{-1}(x) = \ln \frac{1+x}{1-x}$, for reference.

Underestimation in A-Min*-BP is curtailed by the fact that the minimum reliability $g(v_{\min})$ dominates the overall product that forms U^{APP}. This term would have also been included in the outgoing extrinsic calculations used by Full-BP for all but the least reliable incoming edge. The outgoing reliability of the minimum incoming edge incurs no degradation due to underestimation since the proper extrinsic value is explicitly calculated. Outgoing messages to highly reliable incoming edges suffer little from underestimation since their corresponding intrinsic $g(v_{mn})$ values are close to one. The worst case underestimation occurs when two edges 'tie' for the lowest level of reliability. In this instance the dominant term in (10) is squared. An improved version of A-Min*-BP would calculate exact extrinsics for the two smallest incoming reliabilities. However, the results in Fig. 2, where the algorithm (using floating point precision) is compared against Full-BP (using floating point precision) for short and medium length regular and irregular codes, indicate that explicit extrinsic calculation for only the minimum incoming edge is sufficient to yield performance that is essentially indistinguishable from that of Full-BP.

The proposed algorithm is similar to the Offset-Min-BP algorithm of [8] where the authors introduce a scaling factor to reduce the magnitude of extrinsic estimates produced by Min-BP. The Min-BP algorithm finds the magnitude of the two least reliable edges arriving at a given constraint node (which requires $d_c - 1$ comparisons followed by an additional $d_c - 2$ comparisons). The magnitude of the least reliable edge is assigned to all edges except the edge from which the least reliable magnitude came (which is assigned the second least reliable magnitude). For all outgoing edges, the proper extrinsic sign is calculated. As explained in [14] these outgoing magnitudes overestimate the proper extrinsic magnitudes because the constraint node update equation follows a product rule (10) where each term lies in the range $(0, 1)$. The Min-BP approximation omits all but one term in this product. To reduce the overestimation, an offset (or scaling factor) is introduced to decrease the magnitude of outgoing reliabilities. The authors in [8] use density evolution to optimize the offset for a given degree distribution and SNR. The optimization is sensitive to degree sequence selection and also exhibits SNR sensitivity to a lesser extent. Nevertheless, using optimized parameters, performance within 0.1 dB of Full-BP performance is possible.

By way of comparison, A-Min*-BP improves performance over Min-BP because the amount by which U^{APP} underestimates a given extrinsic is less than the amount by which Min-BP overestimates the same extrinsic. Specifically, the former underestimates due to the inclusion of one extra term in the constraint node product while the latter overestimates due to the exclusion of all but one term in the product. A direct comparison to Offset-Min-BP is more difficult. However, a simple observation is that in comparison to Offset-Min-BP, A-Min*-BP is essentially 'self-tuning'.

The range and shape of the non-linear portion (Λ^{BP^*}) of the A-Min*-BP computation are well approximated using a single,

	Full-BP	A-Min*-BP	Offset-Min-BP
Table LookUps	$2d_c$	$d_c - 1$	0
Comparisons	0	$d_c - 1$	$2d_c - 3$
Additions	$2d_c - 1$	0	d_c
XORs	$2d_c - 1$	$2d_c - 1$	$2d_c - 1$
Tot Table Lookups	80000	35000	0
Tot Comparisons	0	35000	65000
Tot Additions	75000	0	40000
Tot XORs	75000	75000	75000
Tot Ops	230,000	145,000	180,000
Performance	Reference	No Loss	0.1dB Loss

Fig. 4. Complexity comparison for three constraint update techniques. Full-BP and A-Min*-BP have essentially the same performance. The simplest technique, Offset-Min-BP, experiences about a 0.1dB loss [12]. Numerical values are shown for a rate 1/2 code with: $n - k = 5000$, and average right degree d_c=8.

or at most a 2-line, piecewise linear fit, as shown in Fig. 3. All of the fixed precision numerical results to be presented in section IV use the 2-line approximation (as do the floating point results in Fig. 2). Hence, the entire constraint node update is implemented using only shift and add computations, no look-ups to tables of non-linear function values are actually required.

The cost of constraint node updating for Full-BP (implemented using (5)), A-Min*-BP, and Offset-Min-BP are given in Fig. 4. The latter two algorithms have similar cost with the exception that $d_c - 1$ table look-up operations in A-Min*-BP are replaced with d_c additions in Offset-Min-BP (for offset adjustment). Note that use of a table is assumed for the representation of Λ^{LnBP}. While Λ^{BP^*} is well approximated using a two line piecewise fit employing power of 2 based coefficients. Variable node updating occurs via (4) for all three algorithms.

IV. Numerical Implementation

Minimum complexity implementation of the A-Min*-BP algorithm necessitates simulation of finite wordlength effects on edge metric storage (which dominates design complexity). Quantization selection consists of determining a total number of bits as well as the distribution of these bits between the integer and fractional parts *(I,F)* of the numerical representation. The primary objective is minimization of the total number of bits with the constraint that only a small performance degradation in the waterfall and error-floor BER regions is incurred. Quantization saturation levels ($Sat = 2^I$) that are too small cause the decoder to exhibit premature error-floor behavior. We have not analytically characterized the mechanism by which this occurs. However, the following provides a rule of thumb for the saturation level,

$$Sat = -\ln(p) \approx \ln\left(\frac{1-p}{p}\right) \qquad \text{where } p = e^{-Sat}.$$

This allows literal Log-Likelihood Ratio (LLR) representation of error probabilities that are as small as p. In practice, this rule seems to allow the error-floor to extend to a level that is about one order of magnitude lower than p.

In the results that follow, simple uniform quantization has been employed, where the step size is given by 2^{-F}. To begin, Fig. 5 shows that low SNR performance is less sensitive to quantization than high SNR performance. A small but noticeable degradation occurs when 2 rather than 3 fractional bits are used to store edge metrics and 4 integer bits are used in both cases. In summary, 7 bits of precision (Sign, 4 Integer, 2 Fractional) are adequate for the representation of observation and edge metric storage in association with the considered code.

When power of 2 based quantization is used, the negative and positive saturation levels follow $[-2^{I-1}, 2^{I-1} - 2^{-F}]$. An alternative approach arbitrarily sets this range between a maximum and a minimum threshold and sets the step size equal to $s = 2 * MaxRange/2^{TotalBits}$. This approach to quantization is more general than the previous since the step size is not limited to powers of 2. We have found that in the low SNR regime, smaller quantization ranges are adequate, but the optimal step size remains similar to that needed at higher SNRs. Thus, operation at lower SNRs requires fewer overall bits given the general range approach to quantization. For example for $E_b/N_o = 1.0$dB, when $MaxRange = 10$ and a total of 6 bits were used, no performance degradation was observed. For higher SNR values, $MaxRange = 16$ was the best choice. This agrees with the results obtained using binary quantization with $(I, F) = (4, 2)$. The performance of this quantizer is described in Fig. 5 by the curve labeled '6bit G.R.' (or 6 bit general range) where in this case the range is set equal to (-10,10)@1.0dB;(-12,12)@1.2dB;(-16, 16)@1.4dB and a total of 6 bits (1 sign, 5 quant-bits) is used. Hence in this case the general range quantizer is equivalent to the (1,4,1) power of 2 quantizer at high SNR. At lower SNRs, the best case range was smaller than (-16,16) such that general range quantization offers an added degree of freedom in precision allocation that is useful in the context of LDPC decoding.

V. The UCLA LDPC Codec

We have implemented the above constraint update technique along with many other necessary functions in order to create a high throughput Monte Carlo simulation for arbitrary LDPC codes. The design runs on a VirtexII evaluation board from Nallatech systems and is interfaced to a PC via a JAVA API.

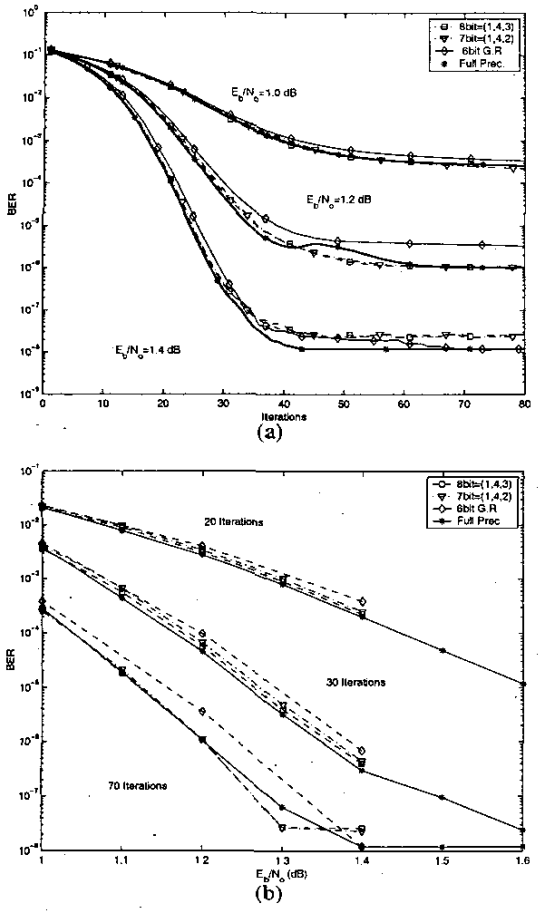

Fig. 5. (a) BER Vs. E_b/N_o, for different fixed numbers of iterations. (b) BER vs Iterations. for different fixed E_b/N_o.

A block diagram is provided in Fig. 6. The Gaussian noise generator developed by the authors in [15] is instantiated next to the decoder so as to avoid a noise generation bottleneck. This block directly impacts the overall value of the system as a Monte Carlo simulator for error-floor testing as good noise quality at high SNR (tails of the Gaussian) is essential. Since the LDPC decoding process is iterative and the number of required iterations is non-deterministic, a flow control buffer can be used to greatly increase the throughput of the overall system.

Through the use of JAVA as an soft interface to the board, we have been able to facilitate the initiation and monitoring of simulations from remote locations. Researchers around the world have successfully uploaded their own LDPC codes for testing on the "UCLA Monte Carlo System".

VI. Conclusion

A reduced complexity decoding algorithm that suffers little or no performance loss has been developed and is justified both theoretically and experimentally. Finite wordlengths have been

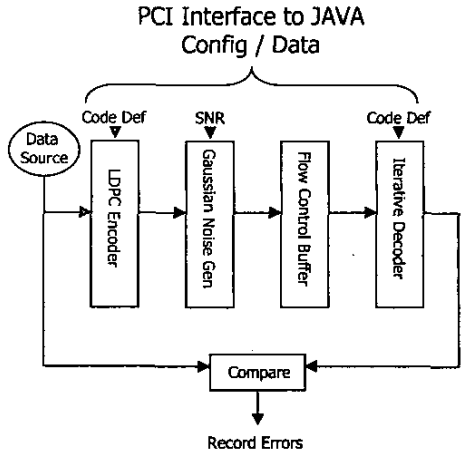

Fig. 6. Architecture block diagram.

carefully considered and 6 to 7 bits of precision have been shown to be adequate for a highly complex (a length 10,000 d_{lmax} = 20 irregular LDPC) code to achieve an error floor that is code rather than implementation limited.

References

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

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *Proc. IEEE Int. Conf. Commun.*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[3] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.

[4] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Inform. Theory*, vol. 47, pp. 585–598, Feb. 2001.

[5] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity approaching irregular low density parity check codes," *IEEE Trans. on Inform. Theory*, vol. 47, pp. 618–637, Feb 2001.

[6] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA: Morgan Kaufmann, 1998.

[7] T. Tian, C. Jones, J. Villasenor, and R. D. Wesel, "Characterization and selective avoidance of cycles in irregular LDPC code construction," *International Conference on Communications (ICC) 2003*.

[8] J. Chen and M. Fossorier, "Density evolution for BP-based decoding algorithms of LDPC codes and their quantized versions," in *Proc. IEEE Globecom*, Taipei, Taiwan, Nov. 2002.

[9] C. Jones, A. Matache, T. Tian, J. Villasenor, and R. Wesel, "LDPC Codes - Universally Good for Wireless Channels," in *Proceedings of the Military Communications Conference*, Oct 2003.

[10] C. Jones, R. Tian, A. Matache, R. Wesel, and J. Villasenor, "Robustness of LDPC Codes on Periodic Fading Channels," in *Proceedings of GlobeCom*, Nov 2002.

[11] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. on Inform. Theory*, vol. 47, pp. 638–656, Feb 2001.

[12] J. Chen, A. Dholakia, E.Eleftheriou, M. Fossorier, and X.-Y. Hu, "Near optimum reduced-complexity decoding algorithms for LDPC codes," in *Proc. IEEE Int. Sym. Inform. Theory*, Lausanne, Switzerland, Jul. 2002.

[13] E.Eleftheriou, T. Mittelholzer, and A. Dholakia, "A reduced-complexity decoding algorithm for low-density parity-check codes," *IEE Electron. Letters*, vol. 37, pp. 102–104, Jan. 2001.

[14] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of LDPC codes," *IEEE Trans. on Comm.*, vol. 50, no. 3, Mar. 2002.

[15] L. Dong-U, J. Villasenor, and W. Luk, "A hardware Gaussian noise generator for exploring channel code behavior at very low bit error rates," in *In proceedings of FCCM*, May 2003.