BIT-FLIPPING POST-PROCESSING FOR FORCED CONVERGENCE DECODING OF LDPC CODES

Ernesto Zimmermann, Prakash Pattisapu†, Gerhard Fettweis**

*Dresden University of Technology, Vodafone Chair Mobile Communications Systems, 01062 Dresden, Germany
†STMicroelectronics Private Ltd, Plot 18, Sector 16A, Noida - 201301,UP, India
email: zimmere@ifn.et.tu-dresden.de

ABSTRACT

The recently proposed *forced convergence* technique allows for reducing the decoding complexity of Low-Density Parity-Check Codes (LDPCC) at only slight deterioration in performance. The basic idea is to restrict the message passing during LDPC decoding to the nodes that still significantly contribute to the decoding result. In this paper, we propose to add a bit-flipping post-processor to the forced convergence decoder in order to alleviate some problems of this novel technique, namely the error floors observed when aiming for high reduction in decoding complexity. Our results show that combining a hard decision bit-flipping with the forced convergence approach enables to almost retain original error correction performance while further reducing decoding complexity.

1. INTRODUCTION

Low-Density Parity-Check Codes have been developed by Gallager in the early sixties [1] and were then largely forgotten until their rediscovery by MacKay and Neal [2] in the late nineties. In recent years LDPCC emerged as a promising candidate technology for forward error correction (FEC) in future wireless systems, due to their near-capacity error correction performance [3, 4, 5].

LDPCC are defined in terms of a sparse parity check matrix $\mathbf{H}$, where codes with a fixed number of non-zero elements per row and column of $\mathbf{H}$ are referred to as *regular* LDPCC. Their irregular counterparts (i.e., codes featuring a non-uniform weight of rows and/or columns) have been shown to be superior in terms of error correction performance [4]. The different algorithms used for LDPC decoding (sum-product decoding, min-sum decoding and variations on the topic) all iteratively approximate the maximum likelihood solution to the decoding problem, i.e., the task of finding the codeword with minimum Euclidean distance from the received signal. This is done by passing messages on the edges between the nodes of the bipartite "Tanner graph", an equivalent representation of the parity check matrix of the considered LDPCC. The average complexity of the message passing algorithm (MPA) is hence essentially the product of the following factors:

1. the number of operations per node,
2. the average number of iterations, and
3. the number of nodes involved in message passing.

Forced convergence (FC) decoding [6, 7] aims at reducing complexity by decreasing the last factor – the number of active nodes. Message passing is restricted to the nodes that are expected to significantly contribute to the decoding result in the following iterations, i.e., nodes that have not yet converged to a final solution. One drawback of this approach is the introduction of an error floor whenever the threshold used to identify converged nodes is chosen too low.

In this paper, we propose the usage of a bit-flipping algorithm after FC decoding, which allows for correcting the residual errors introduced by the deactivation of nodes. The combination of forced convergence with bit-flipping post-processing thus enables a further reduction in decoding complexity at almost no deterioration in FER versus SNR performance. The remainder of this paper is structured as follows: Section 2 describes the standard and forced convergence LDPC decoding algorithm, as well as details of the post-processing procedure. Simulation results are presented in Section 3 before we draw conclusions in Section 4.

2. LOG DOMAIN LDPC DECODING

The Tanner graph [8] of a LDPC code contains two types of nodes – the variable nodes v_i, representing the bits of the codeword (usually on the left hand side) and the check nodes c_j, representing the constraints imposed by the parity checks (usually on the right hand side). Two nodes v_i and c_j are connected, iff $h_{j,i} \neq 0$, that is, the code bit i is checked by the jth check.

At the outset of the decoding process, each variable node i is initialized with the corresponding soft output F_i from the channel detector. Decoding is then done by exchanging extrinsic information between variable and check nodes in the Tanner graph. During the first half-iteration, each variable node v_i sends its "belief" of being in a certain state, given the input from all adjacent check nodes $c_{j'}, j' \neq j$ (denoted by $Q_{i,j}$) to check node c_j. During the second half-iteration, each check node c_j sends its probability of being satisfied (denoted by $R_{j,i}$), given the belief of v_i and taking the messages $Q_{i',j}$ of all other adjacent variable nodes $v_{i'}$, $i' \neq i$ into account to v_i. It is easily shown that the exchanged (log domain) messages are then defined as:

$$Q_{i,j} = F_i + \sum_{j',j' \neq j} R_{j',i} \tag{1}$$

$$R_{j,i} = \prod_{i',i' \neq i} \operatorname{sign}\big(Q_{i',j}\big)\, \Phi\Bigg(\sum_{i',i' \neq i} \Phi\big(|Q_{i',j}|\big) \Bigg), \tag{2}$$

where we define

$$\Phi(x) = \log \frac{e^x + 1}{e^x - 1}, \qquad x > 0. \tag{3}$$

The message passing is repeated until all checks are satisfied (successful decoding) or a maximum number of iterations is reached and decoding failure must be declared.

From the above formulations it is obvious that in message passing, the major source of computational complexity lies in the check node decoder, as the calculation of the check node messages requires the repeated execution of the non-linear function Φ. It is, however, important to note two properties of this function: firstly, $\Phi(\Phi(x)) = x$, and, secondly, $\Phi(x)$ is decreasing very fast as a function of x. Hence the sum in (2) is clearly dominated by the biggest term $\Phi\big(|Q_{i',j}|\big)$ which corresponds to the minimum value of $|Q_{i',j}|$. Keeping the first property in mind, this motivates for the introduction of the *Min-Sum Algorithm* (MSA), where the calculations in the check nodes are approximated by taking only the incoming message with the lowest reliability into account:

$$\tilde{R}_{j,i} \quad = \quad \prod_{i',i'\neq i} \operatorname{sign}\big(Q_{i',j}\big) \; \min_{i',i'\neq i} |Q_{i',j}|, \tag{4}$$

Calculating the check node messages then only requires very simple MAC operations. However, it has been shown in [9] that this kind of approximation generally overestimates check node messages, and that in order to substantially reduce the loss incurred by min-sum decoding it is effective to scale all messages sent by the check nodes with a factor α:

$$\overline{R}_{j,i} \quad = \quad \alpha\tilde{R}_{j,i} = \frac{E\big[R_{j,i}\big]}{E\big[\tilde{R}_{j,i}\big]}\tilde{R}_{j,i}. \tag{5}$$

For our considered code, $\alpha = 0.8125 = \frac{13}{16}$ showed to be a reasonable choice. We will refer to this modified algorithm as the *normalized* min-sum algorithm (NMSA) and include it in our performance assessment.

2.1 Forced convergence

This technique, introduced in [6], reduces decoding complexity by exploiting the fact that a large number of variable nodes converge to a strong belief after very few iterations, i.e., these bits have already been reliably decoded and we can skip updating their messages in subsequent iterations. A threshold rule is used to identify such nodes. This can be done at almost zero overhead, since in a practical implementation one will most certainly calculate the following "aggregate" messages before deriving the messages sent out over the different node edges:

$$\begin{aligned} B_i &= F_i + \sum R_{j,i} \\ C_j &= \sum \Phi\big(|Q_{i',j}|\big), \end{aligned} \tag{6}$$

where the magnitude of B_i and $\Phi(C_j)$ is the confidence of variable node i and check node j, respectively, to be in a specific state (0 or 1). Whenever $|B_i| > t_v$ or $|C_j| < t_c$ we "deactivate" the respective node for the following iterations. Note that t_c is an upper bound since it operates on the sum *before* applying the transform Φ. This approach is easily extended to (normalized) min-sum decoding where t_c serves as a lower bound since output messages are based on the incoming message with lowest reliability (i.e., magnitude). To avoid freezing nodes that have converged to a wrong decision, check nodes that do not fulfill their parity check during tentative decoding are reactivated, as well as the connected variable nodes.

Choosing t_v and t_c appropriately thus allows for trading computational complexity for decoding performance. Note that for a practical implementation, one may store a vector containing the indices of the active nodes to avoid checking during message passing at each node whether it is still active, or not. The main drawback of this technique is the introduction of further bit errors whenever the thresholds are not appropriately chosen. This effect has been analyzed in detail by using extrinsic information transfer (EXIT) charts in [7]. The observed error floors can be explained by the fact that the inactivation of nodes prevents the magnitudes of the extrinsic messages during message passing to reach the high values needed for successful decoding, i.e., the decoding process gets stuck.

2.2 Bit Flipping Post Processing

The bit flipping algorithm which was already proposed by Gallager himself [1] follows a syndrome decoding approach and can be implemented at very low complexity, provided the number of bits in error is very low. The basic idea is to use the knowledge on unsatisfied checks to iteratively correct bit errors. Towards this end, first all unsatisfied checks are identified (this is done during tentative decoding) and then the variable node connected to the most unsatisfied checks is identified. The corresponding bit value is flipped and the syndrome vector updated. This process is repeated until all checks are satisfied or a maximum number of iterations is reached. For our evaluations, we set this number to $N_{BF} = 30$. Since the bit flipping algorithm usually corrects only one or a few violated check per iteration, the bit flipping post-processing is only activated when the number of unsatisfied checks does not exceed $2N_{BF}$. Note that the number of (additional) unsatisfied checks resulting from the application of the forced convergence approach is usually very low (below 10 for the code under consideration). It is hence easily seen that in comparison to a full iteration over all variable and check nodes, the effort of bit flipping is negligible. For the remaining block errors which are not due to forced convergence, the number of unsatisfied checks is usually several hundreds – the bit-flipping will hence only be beneficial for alleviating the deteriorating effects of forced convergence, but not to enhance the decoding result for standard decoding.

3. RESULTS

3.1 Performance

For evaluation we used a (3,6) regular LDPCC of block length 4000 bits (more specifically, the 4000.2000.3.243 code from MacKay's encyclopedia of sparse graph codes [10]) with a maximum of 40 decoder iterations. Similar results have been obtained with other codes (not shown).

Figure 1 shows the performance of forced convergence decoding in conjunction with MPA, with and without bit-flipping. We have shown in [6] that by choosing the thresholds high enough, one can almost retain error correction performance at a target block error rate of 10^{-2} which appears to be reasonable in current wireless systems employing ARQ mechanisms. In this paper, we chose thresholds too low on purpose, thus introducing quite high error floors in the frame error rate. However, since the number of residual bit errors in most of the frames in error is very low, the bit flipping post-processing allows for substantially lowering the observed error floors.

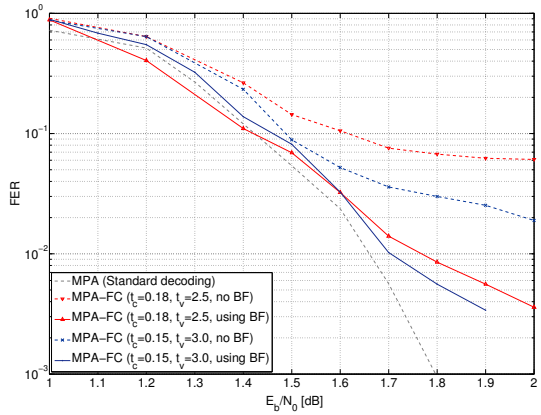

Figure 1: Performance of FC decoding in comparison to standard LDPC decoding, in terms of FER over SNR, using the original message passing algorithm. Choosing thresholds too low quickly leads to error floors unless bit-flipping post-processing is employed.

Figures 2 and 3 shows that the application of forced convergence to (normalized) min-sum decoding is equally viable. For min-sum decoding, one can even outperform the original error correction performance when thresholds are chosen high enough and residual errors corrected by bit-flipping. This effect might be explained by the fact that forced convergence gradually reduces the magnitudes of exchanged extrinsic messages – thus counteracting the deteriorating effects introduced by the min-sum approximation. When used together with the normalized min-sum algorithm, a performance within 0.2 dB of original message passing can be attained, even with a very low threshold of 2.5.

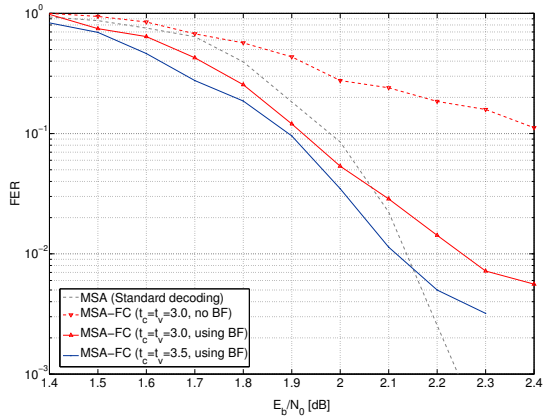

Figure 2: Performance of forced convergence LDPC decoding in combination with MinSum LDPC decoding. Using appropriate thresholds allows for even outperforming the original error performance, when bit-flipping post-processing is used.

3.2 Complexity

Summarizing the statements in section 2, it is possible to calculate the node complexity of the different LDPC decoder architectures compared in this paper. The complexity for calculating the variable node messages is obviously $\gamma_{v,i} = 2\overline{d}_v\,\overline{n}_{v,i}$, where $\overline{d}_v$ is the average variable node degree and $\overline{n}_{v,i}$ the average number of active variable nodes at iteration step i (we similarly define $\overline{n}_{c,i}$). For the check node messages, the respective value is $\gamma_{c,i} = (5+2c_\Phi)\overline{d}_c\,\overline{n}_{c,i}$ for message passing and only $\tilde{\gamma}_{c,i} = 5\overline{d}_c\,\overline{n}_{c,i}$ for min-sum decoding, where c_Φ is the complexity of calculating $\Phi(x)$, which we assume to be 10 MAC operations. For NMSA the check node complexity is $\tilde{\gamma}_{c,i} = 5(\overline{d}_c+1)\,\overline{n}_{c,i}$, taking into account 5 MAC operations (3 shift, 2 add) for the scaling of the outgoing messages. To these values, 1 MAC operation is added for checking the nodes' "belief" against the threshold. The complexity of tentative decoding is obviously $\gamma_{d,i} = (2\overline{d}_c+2)N_c\Pr(N_i >= i)$, where $\Pr(N_i >= i)$ denotes the probability of reaching iteration step i and N_c is the number of check nodes. Again, the value is somewhat higher for forced convergence to take the reactivation of nodes into account. Taking the sum over all iterations and dividing by the number of information bits yields the total complexity in MAC operations per received information bit:

$$\Theta_D = \frac{1}{N_v - N_c}\sum_i(\gamma_{v,i}+\gamma_{c,i}+\gamma_{d,i}) = \Theta_V + \Theta_C + \Theta_{TD}, \quad (7)$$

where N_v is the number of variable nodes. This value is obviously a function of the average number of iterations and hence the SNR. The amount of computations for the bit flipping can be upper bounded by

$$\gamma_b \leqslant N_{BF}(2N_{BF}d_c + 1 + d_v(2d_c+2)), \quad (8)$$

where we used the fact that we execute at maximum N_{BF} loops with maximum $2N_{BF}$ unsatisfied checks. Under our settings, the maximum additional overhead per information bit $\gamma_b/(N_c - N_v)$ is only 6 MAC operations, which confirms our notion that the additional effort is negligible.

One "active node profile" illustrating the development of $\overline{n}_{c,i}$ and $\overline{n}_{c,i}$ over the number of iterations is depicted in Figure 4 at a target FER of 10^{-2} (bit SNR of 1.7 dB). The curves are in fact the product of two factors: the number of

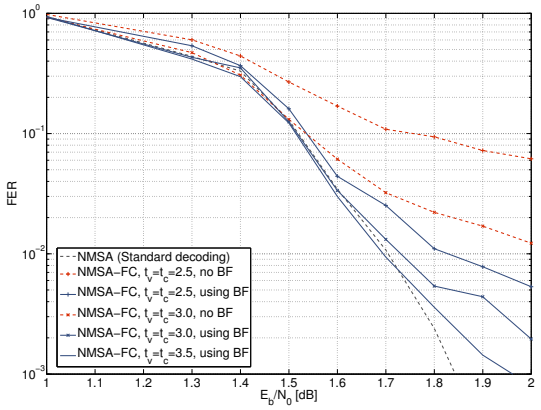

Figure 3: Performance of forced convergence LDPC decoding in combination with Normalized MinSum LDPC decoding.

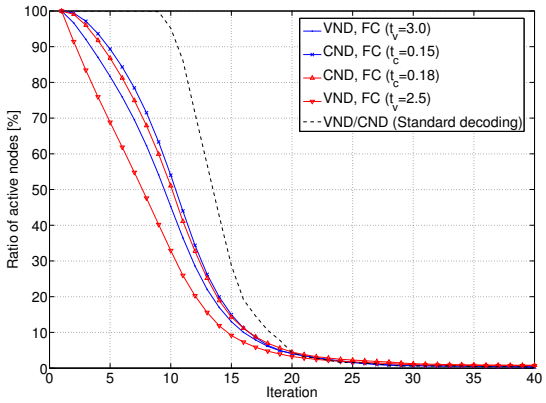

Figure 4: Average ratio of active nodes during LDPC decoding in the variable node (VND) and check node decoder (CND), for message passing decoding and a maximum number of 40 iterations. It is clearly visible how the forced convergence approach reduces the number of active nodes. Results for min-sum decoding are similar (not shown).

active variable nodes in a certain iteration and the probability of this iteration being required to decode the codeword. The decoding complexity essentially scales with the area under the graph. The curves nicely illustrate, how complexity is reduced via forced convergence in the regime of interest. The average number of active nodes decays very fast as we introduce the threshold t_v and t_c to deactivate nodes. However, the curves for FC finally show a higher "error floor" than the curves for standard LDPC decoding, which is a clear indicator that the average number of iterations is increased, due to the residual bit errors. This effect also increases the total effort required for tentative decoding, which scales with the average number of iterations.

Having obtained the necessary statistics on the average number of active nodes, we can proceed to calculate the mean complexity of the different decoder implementations, in the respective regimes of interest at FER $\approx 10^{-2}$. By determining Θ_D, the expected complexity reduction can now be easily calculated:

	MPA		MSA		NMSA	
	Std.	FC	Std.	FC	Std.	FC
Θ_D	163	95	159	83	186	116
Θ_C	2037	1515	398	226	542	389
Θ_{TD}	176	226	172	239	232	265
γ_b	–	6	–	6	–	6
Total	2376	1841	729	554	960	770
Reduction	–	23%	–	24%	–	20%

A complexity reduction between 20 and 25% can be achieved by introducing the forced convergence approach, where most of the gains come from the reduced number of active check nodes. Some of the benefits are, however, offset by the increased average number of iterations, especially in (normalized) min-sum decoding.

4. CONCLUSION

We extended the forced convergence approach with a bit-flipping post-processor to alleviate the high error floors observed when aiming for high reduction in decoding complexity. Our simulation results illustrate that a combination of these two techniques enables to further decrease decoding complexity of LPDCC while almost retaining the original error correction performance. However, a part of the achieved gains are set off by the increased average number of iterations and the overhead of the proposed technique. Reducing this effect and making the forced convergence approach more stable with respect to the selection of the deactivation thresholds constitutes an interesting path for further investigations.

REFERENCES

[1] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge,Massachusetts: M.I.T. Press, 1963.

[2] D. MacKay and R. Neal, "Near Shannon limit performance of Low-Density Parity-Check Codes," *Electronic Letters*, vol. 32, pp. 1645–1646, 1996.

[3] S. Chung, G. Forney, T. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Communications Letters*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[4] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of Capacity-Approaching irregular Low-Density Parity-Check Codes," *IEEE Transaction on Information Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[5] D. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, pp. 399–431, 1999.

[6] E. Zimmermann, P. Pattisapu, P. K. Bora, and G. Fettweis, "Reduced Complexity LDPC Decoding using Forced Convergence," in *Proceedings of the 7th International Symposium on Wireless Personal Multimedia Communications (WPMC04)*, Abano Terme, Italy, 12.-15. Sept. 2004.

[7] E. Zimmermann, W. Rave, and G. Fettweis, "Forced Convergence Decoding of LDPC Codes - EXIT Chart Analysis and Combination with Node Complexity Reduction Techniques," in *Proceedings of the 11th European Wireless Conference (EW'2005)*, Nicosia, Cyprus, 11.-13. Apr. 2005.

[8] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. on Information Theory*, vol. 27, pp. 533–547, Sept. 1981.

[9] J. Chen and M. Fossorier, "Near Optimum Universal Belief Propagation Based Decoding of Low-Density Parity Check Codes," *IEEE Transactions on Communications*, vol. 50, no. 3, pp. 406–414, Mar. 2002.

[10] D. J. C. MacKay, "Encyclopedia of sparse graph codes," Tech. Rep. [Online]. Available: http://wol.ra.phy.cam.ac.uk/mackay/codes/data.html