Memory Efficient LDPC Code Design for High Throughput Software Defined Radio (SDR) systems

Yuming Zhu and Chaitali Chakrabarti
Department of Electrical Engineering
Arizona State University, Tempe, AZ 85287
Email: {yuming,chaitali}@asu.edu

***Abstract*— Low-Density Parity-Check (LDPC) codes have been adopted in the physical layer protocol of many communication systems because of their superior performance. A direct implementation of the LDPC decoder on an existing platform, such as a software defined radio (SDR), is likely to be inefficient. Our approach is to design the LDPC code in a way that takes into account the constraints imposed by the existing architecture, without compromising the communication performance. In this paper, a procedure for architecture-aware LDPC code design which minimize the number of global memory accesses in a memory constrained system is derived. The procedure is built on top of existing super-code based LDPC code design. The proposed code construction procedure also results in reduction in the number of iterations and thereby increases the throughput significantly.**

***Index Terms*— channel coding, throughput (communication systems), memory access, computer architecture.**

I. Introduction

Low-Density Parity-Check (LDPC) codes have attracted the interest of the coding community because of their BER performance being close to the Shannon limit [1], [2]. Consequently, there has been a lot of work on the implementation of LDPC decoders [3], [4], [5], [6], [7], [8]. In order to increase the efficiency of the implementation, recently, the focus has been to keep the decoder design in mind when designing the LDPC codes. The architecture-aware LDPC code design studies in [3], [6], [8] have been targeted for special-purpose VLSI implementations. In this paper, we address the problem of efficient LDPC code design for an existing hardware platform, such as a multi-processor software defined radio (SDR) system.

A typical SDR platform consists of multiple processing units (PU) and multiple global storage units as shown in Fig. 1. The PUs and the global storage units communicate with each other via an interconnection network. Each PU consists of a local memory, a processing element (PE), which consists of a scalar unit and a single-instruction multiple-data (SIMD) unit, and an application specific element (ASE). The combination of a scalar unit and a SIMD unit enables a large class of algorithms to be mapped very efficiently as shown in [9]. The ASEs are typically included to enhance the performance of some target protocols. The software control unit coordinates all the operations in the system.

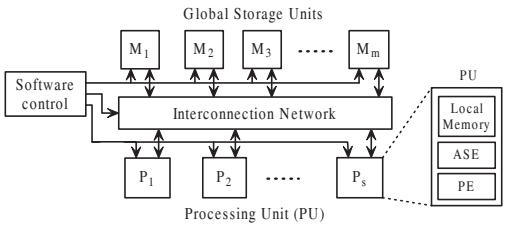

Fig. 1. Software defined radio (SDR) platform.

Recently, we presented a general design flow for systematically designing LDPC codes to exploit the characteristics of an existing multi-processor architecture [10]. We showed how the constraints imposed by the interconnection network can be translated into constraints during the code design phase, resulting in a LDPC code that can be mapped into the target architecture very efficiently.

In this paper, we study the problem of designing LDPC codes for a memory constrained architecture. This problem is motivated by the fact that while large block sizes (which result in large memory requirement) in LDPC codes are required to achieve superior performance, the size of the local memory in a PU is relatively small. As a result, there is a large volume of data that is transferred between the global memory and PU through the SDR interconnection network resulting in possible timing delays due to routing conflicts.

Super-code based decoding of LDPC codes has been shown to be an effective way of reducing the memory requirement of LDPC decoders [6], [11]. Here the LDPC code is viewed as the concatenation of multiple super-codes, and the iterative decoding process of the LDPC codes is divided into the decoding of each super-code (inner iteration that is repeated Q_1 times), and passing information among the super-codes iteratively (outer iteration that is repeated Q_2 times). Since Q_2 is proportional to the number of global memory accesses, it seems imperative that Q_2 should be minimized. However, simulation results show that for the same equivalent iteration number $\tilde{Q} \triangleq Q_1 \cdot Q_2$, decreasing Q_2 degrades the BER performance.

An analysis of the results show that the LDPC codes constructed based on [11] have the same degree distribution for all super-codes, and thus contain a large number of "degree-one" nodes. The bit-to-check information from these "degree-one" nodes does not improve during the inner iterations, resulting in the performance degradation of super-code based decoding. Our approach has been to remove the "degree-one" nodes in the super-codes in the code design stage and thereby improve the performance. Simulation results show that use of these codes increases the convergence speed significantly. In fact, the LDPC code generated by this procedure achieves performance comparable to that in [11] with half the number of iterations. Thus this procedure not only reduces the number of global memory accesses but also increases the overall throughput significantly.

The rest of this paper is organized as follows. The memory-aware LDPC code design is presented in Section II. The memory requirement and memory traffic is analyzed for super-code based decoding. Next, the code design constraints that result in enhanced performance are proposed, and the results validated by simulations. The paper is concluded in Section III.

II. Memory-Aware LDPC code design for SDR

In this section, we assume that in the SDR architecture, only one PU is assigned for LDPC decoding, and that the other PUs are used to perform other kernels such as MIMO, OFDM, etc. The size of the local memory in a PU is relatively small. Thus to implement LDPC codes with large block sizes, a large volume of data has to be transferred between the global and local memories. This can cause reduction in the overall system throughput due to conflicts in the

interconnection network of the SDR platform. Thus our goal is to minimize the number of global memory accesses for a local memory constrained system without compromising the BER performance.

A. Super-code based LDPC decoding

The memory required in the iterative decoding process can be reduced by decomposing the code into super-codes [6], [11]. Fig. 2 shows the decomposition of a LDPC code into two super-codes. The parity check matrices of super-codes (H_b^1 and H_b^2) are sub-matrices of the original H_b matrix. The H_b matrix is a $m_b \times n_b$ block matrix whose elements are either circulant sub-matrices (represented by '1') or zero sub-matrices (represented by '0') with dimension of $Z \times Z$.

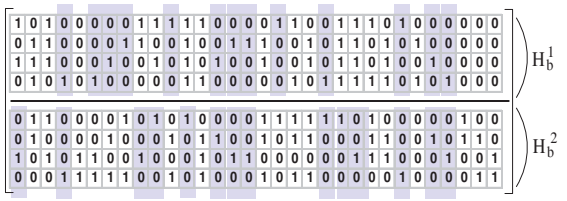

Fig. 2. Decomposition of the LDPC code into two super-codes with block parity check matrix of H_b^1 and H_b^2 respectively.

The decoding algorithm of LDPC code based on super-codes is summarized in Alg. 1, which is similar to the one described in [11] for the case when there are two supercodes. There are two levels of iteration: each super-code is decoded with fixed number of iterations (inner iteration). The output LLR values from one super-code serve as the input to the next super-code. Thus the super-codes are scheduled to decode one after the other and in an iterative fashion (outer iteration). Let C denote the number of super-codes, each of which has a block parity check matrix $H_b^i, i \in [1, C]$. The outer iteration number is represented by Q_2 and the inner iteration number of the i-th super-code is Q_1^i.

The AA-LDPC codes in [6] is equivalent to the case $C = m_b$, $Q_1^i = 1$, $\forall i \in [1, C]$. The scheme shown in [11] is equivalent to the case $C = 2$ and $Q_1^i = Q_1^j, \forall i, j \in [1, C]$.

Algorithm 1 Super-code based iterative decoding

1: {Initialization:}
Set iteration number $i = 0$, and for $\forall n \in [1, N], \forall m \in M(n)$ set
$$E_{m,n} = 0, \qquad L_n = I_n$$
2: **while** $i \leq Q_2$ and parity check equation not met **do**
3: **for all** $j \in [1, C]$ **do**
4: Load local memory with L_n values for super-code H_b^j from global memory (Scheme-II only).
5: Load Check-to-Bit information $E_{m,n}$ for all edges in H_b^j
6: **for all** $k \in [1, Q_1^j]$ **do**
7: Perform one iteration of BP decoding for H_b^j
8: **end for**
9: Save updated $E_{m,n}$ values to global memory.
10: Save updated L_n values to global memory (Scheme-II only).
11: **end for**
12: **end while**
13: Output the soft information or hard decision of L_n, $\forall n \in [1, N]$ (Scheme-I only).

B. Memory Analysis

The data that needs to be stored for iterative belief propagation (BP) decoding process includes the log-likelihood ratio (L_n) and the check-to-bit information ($E_{m,n}$). In a PU with SIMD datapath as in [9], all the check-node and bit-node operations are performed in parallel. The memory required for decoding a circulant matrix based LDPC code with BP algorithm is of size

$$\text{MEM}_{Total}^{SIMD} \cong N_p \cdot (n_b + m_b \cdot d_c) \tag{1}$$

where n_b is the number of block columns, m_b is the number of block rows, P is the parallel factor of the SIMD unit, $N_p = \lceil \frac{Z}{P} \rceil$, Z is the dimension of the circulant sub-matrices and d_c is the maximum row weight (the equality holds for LDPC code with single check node degree).

For super-code based decoding, only the L_n and $E_{m,n}$ related to the super-code currently being decoded are required to be stored in local memory. Consequently, the local memory size is smaller. We discuss two storage schemes:

- Scheme-I: The local memory keeps a complete copy of all L_n values and the check-to-bit information of the super-code to be decoded. The global memory does not need to store the L_n values and only needs to store the check-to-bit information.
- Scheme-II: The local memory only stores the L_n values for the block columns which will be accessed by the super-code to be decoded. The local memory size can be greatly reduced at the expense of an increase in the global memory size as well as the number of global memory accesses.

The memory requirement for both schemes are summarized below:

$$\text{MEM}_{Local}^{SIMD} = \begin{cases} N_p \cdot \max_{i \in [1,C]} \left(n_b + m_b^i \cdot d_c\right), & \text{Scheme-I} \\ N_p \cdot \max_{i \in [1,C]} \left(n_b^i + m_b^i \cdot d_c\right), & \text{Scheme-II} \end{cases} \tag{2}$$

$$\text{MEM}_{Global}^{SIMD} = \begin{cases} N_p \cdot m_b \cdot d_c, & \text{Scheme-I} \\ N_p \cdot (n_b + m_b \cdot d_c), & \text{Scheme-II} \end{cases} \tag{3}$$

where m_b^i is the number of block rows in H_b^i and n_b^i is the number of block columns that are not all-zeros in H_b^i. For Scheme-I, the m_b^i values are usually set equal for all super-codes, i.e, $m_b^i = \lceil \frac{m_b}{C} \rceil$, to minimize the local memory. For Scheme-II, there exist some optimal super-code decomposition method to maximize the utilization of the memory.

It is clear that with the increase in the number of super-codes, the local memory requirement reduces. Thus the size of the available local memory can be used to determine the number of supercodes, as will be described later. Note that the reduction of local memory comes at the expense of increased memory traffic between the global memory and the PU local memory.

The number of global memory accesses, T_{mem}, is given by

$$T_{mem} = 2 \sum_{i=1}^{C} (m_b^i + \Delta n_b^i) \cdot d_c \cdot Q_2 \cdot T_1 \tag{4}$$

where T_1 is the number of cycles for transmitting Z values between global memory and local memory, Δn_b^i is the number of block columns in the super-code currently being decoded and not in the previous super-code. For Scheme-I, $\Delta n_b^i = 0$, $\forall i \in [1, C]$, and $T_{mem} = 2 m_b \cdot d_c \cdot Q_2 \cdot T_1$. Thus the number of global memory accesses can be reduced by reducing Q_2 and our goal is to achieve this without compromising on the BER performance.

The number of decoding cycles is

$$T_{dec} = \left(\sum_{i=1}^{C} m_b^i \cdot Q_1^i \right) \cdot Q_2 \cdot T_2 \tag{5}$$

where T_2 is the number of cycles for one iteration of decoding for a block row in H_b (including bit node processing and check node

processing). The Q_1^i can be independently chosen for different super-codes; for example, it can be chosen proportional to the block row number m_b^i. However, for simplicity of control and performance comparison, we choose $Q_1^i = Q_1$, $\forall i \in [1, C]$. In this case, $T_{dec} = m_b \cdot Q_1 \cdot Q_2 \cdot T_2$. Thus reduction in Q_2 results in a reduction in T_{dec} and thereby improves the throughput.

In the next section, we will study the effect of reducing Q_2 and propose a method to compensate for the performance degradation in super-code based iterative decoding.

C. Study of BER Performance

In the method proposed in [11], all super-codes have the same degree distribution and the degree distribution for super-codes are derived directly from the overall degree distribution. However, the problem with this method is that it results in a large number of degree-one bit nodes in the super-codes. It can be proved that the bit-to-check information from such bit nodes do not improve during the Q_1^i iterations of decoding for super-code H_b^i.

Theorem 2.1: The bit-to-check information does not improve during iterative decoding for super-code if the bit node is of degree-one in the super-code.

Proof: Let bit node n' be a degree-one node in super-code H_b^i and the check nodes connected to it be $m \in M^i(n')$. Without loss of generality, assume that the current outer iteration is $s \in [0, Q_2 - 1]$, and that the inner iteration is $t \in [0, Q_1^i - 1)$. We denote each variable with a two-tuple superscript (s, t) to show the relationship among the different iterations.

The current bit-to-check information

$$L_{n',m}^{(s,t)} = L_{n'}^{(s,t)} - E_{n',m}^{(s,t)} \tag{6}$$

After the check node processing, the new check-to-bit information $E_{n',m}^{(s,t+1)}$ is obtained, and thus the new LLR value is

$$L_{n'}^{(s,t+1)} = L_{n',m}^{(s,t)} + E_{n',m}^{(s,t+1)} \tag{7}$$

For the next inner iteration, we have

$$L_{n',m}^{(s,t+1)} = L_{n'}^{(s,t+1)} - E_{n',m}^{(s,t+1)} = L_{n',m}^{(s,t)} \tag{8}$$

Thus the $L_{n',m}$ values do not get updated in successive iterations if the corresponding variable node n' is degree-one in the super-code. ■

Since the bit-to-check information of the degree-one bit nodes do not change during the inner iterations, they can only be updated in the outer iteration. Thus the errors in such bit nodes are not prone to be corrected during the process of iterative decoding. This is especially bad for the cases where they happen to be erasure bits (i.e., $I_n \approx 0$).

For LDPC codes designed without the knowledge of super-code configurations, it is inevitable that there be many such degree-one bit nodes in super-codes. In Fig. 2, these bit nodes are shown shaded.

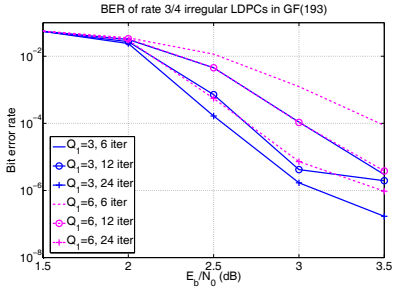

Fig. 3. BER performance for different iteration numbers. The LDPC code is the one in Fig. 2 with $C = 2$. The iteration number in legends are the equivalent iteration number, i.e., $Q_1 \times Q_2$.

Fig. 3 shows the effect of different iteration numbers on the BER performance for two configurations ($Q_1 = 3$ and $Q_1 = 6$). For the same number of equivalent iterations, $\tilde{Q} = Q_1 \cdot Q_2$, a larger value of Q_1 (and thus smaller value of Q_2) results in lower performance. Further more, in the relatively low SNR range (as in the range that we have simulated), when the number of iterations is small, the performance is strongly dependent on Q_2. For example, the case $Q_1 = 3, \tilde{Q} = 6$ and the case $Q_1 = 6, \tilde{Q} = 12$, both correspond to $Q_2 = 2$ and have comparable performance. Thus in the low SNR region, increasing Q_1 does not result in enhanced performance and so additional techniques have to be employed to compensate for the performance degradation caused by choice of low Q_2.

D. Code optimization for super-code based decoding

To improve the performance of the configurations with small Q_2, we propose a mechanism to reduce the number of degree-one nodes in super codes. We impose an additional constraint in the LDPC code construction procedure, which is given below. The resulting LDPC code is referred to as the *optimized* code in the rest of the paper.

- Divide H_b into c parts, each of which corresponds to a super-code. The number c is chosen as the smallest number that enables the local memory to accomodate all the L_n and $E_{m,n}$ values of one super-code.
- If $m_b^i \geq 2$ and there exists degree-one nodes in the ith super-code, perform exchange operation [10] to make the degree at least 2 or even 0. The only exceptions are the degree-one nodes that are critical in maintaining a certain structure in the parity check matrix. Note that multiple passes may be requires since it may not be possible to remove all the degree-one nodes in one pass.
- Minimize the outer iteration number Q_2 as long as the BER performance is acceptable.

Fig. 4. Parity check matrix of an LDPC code optimized for super-code based decoding.

Fig. 4 shows the H_b matrix after applying the exchange operation on the H_b matrix shown in Fig. 2. Almost all the degree-one nodes in super-codes have been removed. There are only two degree-one nodes left in H_b, which are purposely kept unchanged to maintain the lower triangle shape of the H_b matrix.

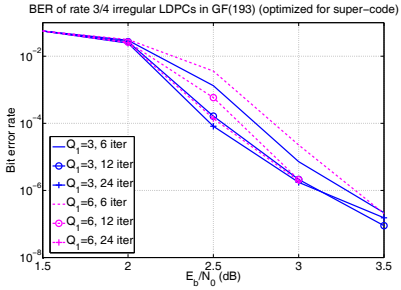

Fig. 5. BER performance for different iteration numbers. The LDPC code is the one in Fig. 4 with $C = 2$. The iteration number in legends are the equivalent iteration number, i.e., $Q_1 \times Q_2$.

Next, we analyze the performance of the optimized LDPC codes. Fig. 5 shows the effect of different iteration numbers on the BER

performance for two configurations ($Q_1 = 3$ and $Q_1 = 6$). A comparison with Fig. 3 shows that the optimized LDPC codes for super-code based decoding converge much faster. Thus fewer iterations are required to achieve the same performance. Further more, the performance is no longer strongly dependent on Q_2 even for small Q_2 values. This is quite different from the random case described in Fig. 3.

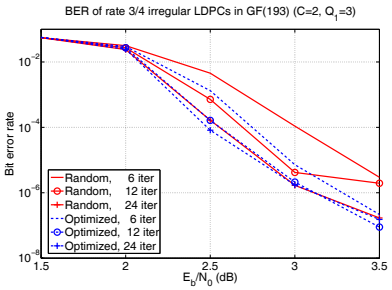

Fig. 6. BER performance of the random and optimized LDPC codes with $Q_1 = 3$ for different iteration number.

E. Trade-off analysis

Fig. 6 and Fig. 7 compare the BER performance of the original and optimized codes with different iteration numbers for $Q_1 = 3$ and $Q_1 = 6$ respectively. In both cases, the optimized LDPC codes can achieve the same performance as the original codes with approximately one half the number of iterations. The number of memory acceses is reduced by a factor of 2. The number of decoding iterations is also reduced by a factor of 2. The overall throughput of the decoder is $(n_b - m_b) \times Z/(T_{mem} + T_{dec})$, which is approximately inversely proportional to the number of outer iterations Q_2. Thus use of the optimized code increases the throughput of the super-code based decoder by approximately a factor of 2.

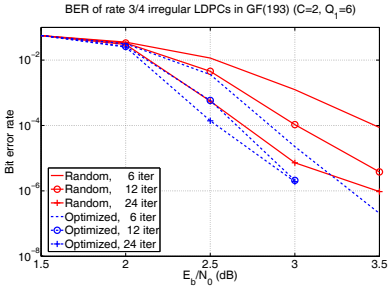

Fig. 7. BER performance of the random and optimized LDPC codes with $Q_1 = 6$ for different iteration number.

The super-code oriented optimization also reduces the local memory requirement when Scheme-II is employed. The decoder for the LDPC in Fig. 4 requires 9.7% less local memory than the counterpart for the one in Fig. 2. The results for memory requirement and memory accesses are summarized in Table I. The memory requirements are normalized to the global memory requirement of Scheme-II. Four comparisons among the randomly generated code and the optimized code are carried out in terms of the BER performance and number of global memory accesses. It is clear that the optimized LDPC codes can reduce the number of memory accesses by a factor of 2 with less than 0.1 dB loss in BER performance (the difference is even lower for $Q_1 = 6$ cases).

TABLE I
COMPARISON OF THE RANDOM AND OPTIMIZED CODE IN TERMS OF MEMORY

Item		Random	Optimized
Total Memory (Scheme-I)	Global	76.5%	76.5%
	Local	60.3%	60.3%
Total Memory (Scheme-II)	Global	100%	100%
	Local	60.5%	55.1%
Comparison 1 $Q_1 = 3$	Q_2	8	4
	SNR for 10^{-5}	2.77 dB	2.82 dB
	Memory Accesses	100%	50%
Comparison 2 $Q_1 = 3$	Q_2	4	2
	SNR for 10^{-5}	2.91 dB	2.97 dB
	Memory Accesses	50%	25%
Comparison 3 $Q_1 = 6$	Q_2	4	2
	SNR for 10^{-5}	2.98 dB	2.86 dB
	Memory Accesses	50%	25%
Comparison 4 $Q_1 = 6$	Q_2	2	1
	SNR for 10^{-5}	3.40 dB	3.14 dB
	Memory Accesses	25%	13%

III. CONCLUSION

In this paper, we presented an architecture-aware LDPC code design procedure, where the architectural constraint is the size of the local memory. The objective was to reduce the number of global memory accesses without affecting the BER performance. We utilized the super-code based decoding which has been shown to reduce the memory requirement of the LDPC decoder [6], [11]. We showed that the convergence speed of the super-code based decoding can be greatly increased by removing the degree-one nodes in the super-codes. We showed that the optimized code can reduce the number of memory accesses by a factor of 2 and increase the throughput by a factor of 2 with no degradation in BER performance.

REFERENCES

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, no. 1, pp. 21–28, Jan. 1962.

[2] S.-Y. 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 Commun. Lett.*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[3] E. Boutillon, J. Castura, and F. R.Kschischang, "Decoder-first code design," in *Proc. of the 2nd Intl. Symp. on Turbo Codes & Related Topics*, Sept. 2000, pp. 459–462.

[4] C. Howland and A. Blanksby, "A 220 mW 1 Gb/s 1024-bit rate-1/2 low density parity check code decoder," in *IEEE Conf. on Custom Integrated Circuits (CICC)*, May 2001, pp. 293–296.

[5] Y. Chen and D. Hocevar, "A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder," in *Proc. of Global Telecom. Conf. (GlobeCom)*, vol. 1, Dec. 2003, pp. 113–117.

[6] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. VLSI*, vol. 11, no. 6, pp. 976–996, 2003.

[7] D. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *IEEE Workshop on Signal Processing Systems (SIPS)*, 2004, pp. 107–112.

[8] H. Zhong and T. Zhang, "Block-LDPC: a practical LDPC coding system design approach," *IEEE Trans. Circuits and Systems I*, vol. 52, no. 4, pp. 766–775, April 2005.

[9] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: a low-power architecture for software radio," in *The 33rd Ann. Int. Symp. on Computer Arch. (ISCA)*, June 2006.

[10] Y. Zhu and C. Chakrabarti, "Architecture-aware LDPC code design for software defined radio," in *IEEE Workshop on Signal Processing Systems (SIPS 2006)*, 2006.

[11] H. Sankar and K. R. Narayanan, "Memory-efficient sum-product decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 52, no. 8, pp. 1225–1230, Aug. 2004.