An Algorithmic Error-Resilient Scheme for Robust LDPC Decoding

Hui-Ting Li,1 Ding-Yuan, Lee1, Kun-Chih Chen,2 and An-Yeu (Andy) Wu1
1 Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan (R.O.C.)
2 Intel-NTU Connected Context Computing Center, National Taiwan University, Taipei, Taiwan (R.O.C.)

Abstract—To fit into multiple communication standards, flexible Low-Density Parity-Check (LDPC) decoding is desirable to be implemented in a chip multiprocessor (CMP) system. However, reliability issues, such as soft errors and timing errors, are severer in future advanced CMP systems when CMOS technology scale. Therefore, enhancing error resilience for a CMP system becomes an important design issue. In this paper, we propose a design methodology to achieve a robust LDPC decoding based on algorithmic error-resilient method. We firstly analyze the performance degradation caused by the soft errors which occur in the computing units (check node units and bit node units), and then explore the inherent error-tolerant characteristic of LDPC decoding algorithm. In our proposed method, we exploit some characteristic distribution or behavior in the operations of the LDPC decoding algorithm to validate the computing results. The experimental results show that the proposed algorithmic error resilience can approach the error-free decoder while facing high injected soft-error rate of $10^{-3}$ in computing units, but with only 6.07% computational overhead. To the best of our knowledge, this is the first discussion about the LDPC decoding algorithm in terms of soft errors in computing units.

Keywords—Error resilience; LDPC; Algorithm-level design

I. INTRODUCTION

As CMOS technology advances, more powerful many-core systems, such as Intel 80-core system [1], are developed and widely used in many high-performance computing applications, such as medical imaging, data mining, and machine learning. However, technology scaling will face some reliability issues. For example, the alpha particle strikes or cosmic rays result in soft errors, the crosstalk results in erratic errors, and process-voltage-temperature variation results in timing errors [2]. Those errors make the microelectronic circuits and overall systems much less reliable in future advanced CMOS technologies. Moreover, energy efficiency is another important consideration in the advanced CMOS technology. However, the technique for energy efficiency, such as voltage over-scaling (VOS), is known to occasionally produce erroneous outputs [3]. Consequently, the reliability issues must be considered in the advanced many-core system designs.

In the case of application-specific integrated circuit (ASIC), traditional techniques like N-modular redundancy (NMR) [4] and algorithmic noise-tolerance (ANT) [5] need additional hardware components, which leads to very high area overhead. However, the problem of area overhead in the many-core system can be alleviated due to the inherent core replication. In [6], Mitra et al. proposed a robust system architecture called Error Resilient System Architecture (ERSA) to achieve high degrees of error resilience for new killer applications like recognition, mining, and synthesis (RMS), which enhances the robustness of many-core system for the RMS applications.

II. BACKGROUND

A. LDPC Codes

LDPC codes are linear block codes defined by parity-check sparse binary $H$ matrices with $M$ rows and $N$ columns. The set of valid codewords $C$ satisfies...
In the decoding aspect, a parity check matrix can be mapped into a bipartite graph formed by BNUs and CNUs, which are linked by bidirectional edges also called Tanner graphs. The connection also stands for the mutual information or message passing among CNUs and BNUs in the decoding stage.

### B. Decoding Algorithm

LDPC codes can be decoded by using a parallel belief propagation decoding algorithm [11], which belongs to the iterative decoding approach. Messages are exchanged between CNUs and BNUs in the Tanner graph. For the soft decision decoding algorithm, the exchanged message is assumed to be log-likelihood ratio (LLR), with its appropriate sign, indicating the received signal should be 1 or 0. The absolute value of LLR corresponds to the belief whether a made decision is correct or not. Therefore, the sign of LLR is by far the most important message in the decoding algorithm. The characteristic of parallel computing makes LDPC decoding is suitable to being implemented in many-core system.

Assume that transmitted and received signals are $U(u_1, u_2, \ldots, u_n)$ and $V(v_1, v_2, \ldots, v_n)$, respectively. Also, $L_n$ indicates the a priori LLR of BNU$_n$, derived from the received signal, and $\Lambda_{-m \to n}$ designates the message sent from CNU$_m$ to BNU$_n$. And, $\hat{\lambda}_{m \to n}$ is the LLR sent from BNU$_n$ to CNU$_m$. The exact decoding algorithm is as follows:

#### Step 1: Initialization

Each bit node $n$ is assigned as a posteriori LLR by using

$$L_n(u_n) = \log \frac{P(u_n = 0 | v_n)}{P(u_n = 1 | v_n)} .$$

The initial values of BNUs and CNUs are

$$\hat{\lambda}_{n \to m}(u_n) = L_n(u_n) \quad \text{(Initial values of BNUs)}$$

and

$$\Lambda_{m \to n}(u_n) = 0 . \quad \text{(Initial values of CNUs)}$$

#### Step 2: Check node updating

Each CNU collects the messages from the correspondingly connected BNUs and updates the exchanged information by

$$\alpha_m = \text{XOR}\{\text{sgn}(\hat{\lambda}_{m \to n}) | n' \in N(m) \setminus n\},$$

$$\beta_m = 2 \tanh^{-1}\{\exp[\sum_{n' \in N(m) \setminus n} \log \tanh(\frac{\hat{\lambda}_{m \to n'}(u_{n'})}{2})]\} ,$$

and

$$\Lambda_{m \to n}(u_n) = \alpha_m \times \beta_m ,$$

where $N(m) \setminus n$ represents the set of BNUs connected to the check node $m$, excluding the $n$-th BNU.

#### Step 3: Bit node updating

Each BNU collects the messages from the correspondingly connected CNUs and updates the exchanged information by

$$\hat{\lambda}_{n \to m}(u_n) = L_n(u_n) + \sum_{m' \in M(n)} \Lambda_{m' \to n}(u_n) ,$$

where $M(n) \setminus m$ represents the set of CNUs connected to the bit node $n$, excluding the $m$-th CNU. Then, iteratively execute the Step 2 and Step 3 until the predefined maximum number of decoding iterations is satisfied.

#### Step 4: Decoded-bit decision

Assume that the decoded output signal is $\hat{U} = (\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_n)$. The decision can be computed by

$$\hat{\lambda}_n(u_n) = L_n(u_n) + \sum_{m \in M(n)} \Lambda_{m \to n}(u_n)$$

and

$$\hat{u}_i = \begin{cases} 0, & \text{if } \hat{\lambda}_i(u_i) \geq 0 \forall i \in \{1, 2, \ldots, n\}, \\ 1, & \text{if } \hat{\lambda}_i(u_i) < 0 \forall i \in \{1, 2, \ldots, n\}, \end{cases}$$

where $M(n)$ represents the set of CNUs connected to the $n$-th BNU.

### III. ERROR ANALYSIS IN LDPC DECODING

To analyze the various degrees of degradation in performance when errors occur in different part of LDPC decoder, we injected random bit flips into the registers which are used to perform information updating in CNUs and BNUs, respectively. In the case of CNUs, we inject errors in the operations in (5) and the accumulators in (6). For BNUs, the errors are injected in the operations in (2), and the accumulators in (8) and (9). The errors which occur in (10) will destroy the successful message passing in all iterations and cause an inevitable error floor. Therefore, we do not consider this type of error.

Fig. 2 shows the bit error rate after decoding under different scenarios. Because sign and magnitude of LLR value are processed independently in CNUs, the sign of LLR is more vulnerable to soft errors in CNUs than in BNUs. Besides, the operations in (6) are processed in the logarithm-hyperbolic tangent domain, so that a bit flip will cause enormous and non-linear influence on the magnitude of LLR. Hence, the CNU errors result in more performance impact than the BNU errors.
IV. PROPOSED RELIABILITY-AWARE DECODING ALGORITHM

In this section, we describe the proposed algorithm-based error-resilience method in LDPC decoding algorithm. For GPU implementation, all the CNUs and BNNs are assigned in several blocks. A block is set as a fundamental computing unit of GPU. The number of threads assigned to a block should be cautiously designed to obtain efficient parallel operation [10]. To make the decoder robust respect to the soft errors which occur in computing units, we propose a reliability-aware decoding algorithm for both CNU updating and BNU updating.

A. Protection Scheme in Check Node Units (CNU)

In each check node unit, which is the more significant part of LDPC decoder, we adopt the temporal bit-level N-modular redundancy (TNBMNR) regarding the computation of the sign of LLR in (5). To achieve acceptable robustness, we set N equal to 5 as a design example in this paper. Compared with the conventional NMR, the proposed TBNMNR processes five times operations of (5) in one computing unit rather than hardware replication during each iteration and decides the result by majority voting. This method considerably reduces the error rate in the sign of LLR. Because the computation for the magnitude of LLR in (6) is much more complex, the proportion of the computation load for the sign bit in whole operation of a check node unit is very small. Based on our experimental results, the overall extra computational overhead of CNU is about 0.83%.

For the computation of the magnitude of LLR, we utilize the inherent error tolerance in LDPC decoding algorithm. The magnitude of LLR represents the degree of belief whether the passed message is correct or not, which is less important part of LLR. In addition, the messages will converge to the correct value during subsequent iterations by error-free BNNs. Therefore, erroneous magnitude in passed message has the advantage of causing near zero performance loss from the viewpoint of overall system.

Fig. 5(a) shows the comparison of the decoding performance between unprotected CNUs and protected CNUs with the proposed TBNMNR when the soft errors are injected in the CNUs for an error rate of 10^{-3}. The experimental results show that the performance is close to the results of error-free decoder even though we only protect the sign of LLR.

B. Protection Scheme in Bit Node Units (BNU)

In each bit node unit, we propose the sanity check method by identifying algorithm invariants that can be used to validate a result of computation. The algorithm invariants refer to some characteristic distribution or behavior in the operation of a specific algorithm.

Observing the updating function of a bit node unit in (8), we can find that there is significant data reuse in each summation operation. Fig. 3 illustrates an example of the n-th BNU. The first output $\delta_{n\rightarrow m1}$ is the summation of $L_o$ and all the inputs, $A^i$, except $A_{m1\rightarrow n}$, the second output $\delta_{n\rightarrow m2}$ is the summation of $L_o$ and all the inputs, $A^i$, except $A_{m2\rightarrow n}$. Therefore, the difference between $\delta_{n\rightarrow m1}$ and $\delta_{n\rightarrow m2}$ should be the difference between $A_{m1\rightarrow n}$ and $A_{m2\rightarrow n}$, and we can easily find the characteristic by

$$\lambda_{n\rightarrow m1} - \lambda_{n\rightarrow m2} = (L_o + \Lambda_{m1\rightarrow n} + \Lambda_{m3\rightarrow n} + \Lambda_{m4\rightarrow n}) - (L_o + \Lambda_{m1\rightarrow n} + \Lambda_{m3\rightarrow n} + \Lambda_{m4\rightarrow n}) = \Lambda_{m2\rightarrow n} - \Lambda_{m1\rightarrow n} \quad (11)$$

Obviously, the difference between any two outputs of a bit node unit is related to the inputs of this bit node unit. More specifically, a rational absolute difference between two outputs should be smaller than the maximum difference between two inputs of this BNU. Therefore, we can validate each output by exploiting the property of significant data reuse. In the proposed sanity check method, we detect the occurrence of soft error in each iteration of a BNU by the steps shown in Fig. 4. We firstly search the maximum and minimum value among all the outputs of BNU in step 1. Next, we define a threshold value equal to the difference between the maximum value and minimum value, which are obtained in the first step. After the BNU updates all its outputs, we can detect the occurrence of errors by checking that if any difference between two outputs is larger than the threshold value in step 3. If there is a detected error, the BNU will re-compute the overall function in (8). To avoid that BNNs are caught in an infinite loop in re-computation, we set the maximum re-computation limit to 3 in this paper.

---

**Fig. 3** The n-th BNU update the exchanged information for (a) CNU$_{m1}$ and (b) CNU$_{m2}$.

**Fig. 4** The pseudo code of the proposed sanity check method in BNU.
Fig. 5 Comparison of decoding performance under a soft error rate of \(10^{-3}\) between (a) unprotected CNU and protected CNU with TBMR while errors only occur in CNU, (b) unprotected BNU and protected BNU with sanity check while errors only occur in BNU, and (c) overall performance while errors occur in both CNU and BNU.

Fig. 5(b) shows the comparison of the decoding performance between unprotected BNU and protected BNU with the proposed sanity check method when the soft errors are injected in the BNUs for an error rate of \(10^{-3}\). As a result, we can detect almost all the soft errors except there are more than three soft errors in one iteration of a BNU, which is almost impossible when the error rate is \(10^{-3}\). Therefore, the proposed sanity check effectively makes the BNUs more robust. Based on our experimental results, the extra computational overhead of BNU for error detection is about 18.3%.

V. SIMULATION RESULTS

To evaluate the proposed methods, we simulated the LDPC decoding at various block sizes and SNR. The simulation is based on an AWGN channel with BPSK modulation, which is a widely used channel modulation [12]. We calculated the raw bit error rate (BER) rather than directly used the SNR to represent the actual information of channel. In this section, we only show the results for an LDPC decoder with a block length of \(M = 200\) and \(N = 400\). For other block sizes, the trends are similar. To simplify the problem, the number of iterations in decoding algorithm is set to 5. The soft errors were injected into the computing units of both CNU and BNU as what we have discussed in Section III.

Fig. 5(c) shows the decoding performance under different scenarios: 1) unprotected LDPC decoder, 2) protected BNU by using the proposed sanity check, 3) protected CNUs with the proposed TBMR, 4) protected LDPC decoder by using the proposed two methods, and 5) error-free LDPC decoder. If we induce errors in the CNUs and BNUs without any protections, we can see an obvious impact on the decoding performance even through LDPC decoding is inherently error tolerant. In the case of protecting only CNUs, there is a performance limitation of coded BER at around \(10^{-4}\) because of the soft errors which pollute the sign of LLR in CNUs. As for protecting only CNUs, which is more the influential part of LDPC decoding algorithm, there seems to be a great improvement in the decoding performance, but an error floor still exists. Hence, a comprehensive error handling is really important. By combining the two proposed methods, the decoding performance approaches the error-free decoder for a high soft error rate of \(10^{-3}\) in computing units. Based on the experimental results, the overall computational overhead is only 6.07%.

VI. CONCLUSIONS

In this paper, we propose a design methodology for error-resilient LDPC decoding. Through the proposed temporal computing redundancy and sanity check, the LDPC decoding can approach error-free decoder for a high soft error rate of \(10^{-3}\) in computing units with low computational overhead. To the best of our knowledge, this is the first discussion about the LDPC decoding in terms of soft errors in computing units.

ACKNOWLEDGEMENT

This work was supported by the Ministry of Science and Technology of Taiwan under MOST 103-2220-E-002-003.

REFERENCES