Multi-mode Low-latency Software-defined Error Correction for Data Centers

Fakhreddine Ghaffari*, Ali Akoglu†, Bane Vasić† and David Declercq*
*ETIS, UMR 8051,Univ Paris Seine, Univ Cergy-Pontoise, ENSEA, CNRS, France
{ghaffari, declercq}@ensea.fr
†Dep. of Electrical and Computer Engineering, Uni. of Arizona, Tucson, AZ, 85719 USA
{akoglu,vasic}@email.arizona.edu

***Abstract*—Flash memories are gaining prominence for utilizing in large scale data centers (DCs) due to their high memory density, low power consumption and heat dissipation, and high access speed characteristics. The rate of degradation for a flash memory is largely affected by the amount and frequency of the erase/write operations, which is a challenge in the DC context that serves dynamically changing workloads. Adaptive Error Correction Code (AECC) schemes have been introduced for changing the error correction algorithm based on the reliability state of the flash. In this study we show that hard decision (bit-flipping) and soft-decision decoding (Belief Propagation) class of algorithms for Low Density Parity Check (LDPC) decoders complement each other for utilizing in the flash based DCs in order to meet the dynamically changing reliability level. We propose a new family of ECC to improve the reliability of flash memory. Our Monte-Carlo simulations and Field Programmable Gate Array (FPGA) based hardware implementation analysis show that LDPC decoders are suitable for balancing the throughput, decoding performance and reliability requirements in DCs.**

Keywords: ***high-performance hard-decision and soft decision LDPC decoders, FPGA architecture, hardware complexity/decoding performance trade-off, low-latency LDPC decoder, data centers***

I. Introduction

Today, two types of approaches dominate the Data Centers (DCs) architecture exploration studies. First one is designing hardware architecture along with its programming environment that enables seamless integration of heterogeneous set of computing nodes (CPU, Graphics Processing Unit (GPU), Field Programmable Gate Array (FPGA), etc.) through a shared memory system accessible by each node. In this study, we take a position in favor of the second approach based on the evidence that next generation DCs will require a composable system [1] utilizing a set of flexible building blocks, which can be dynamically and automatically assembled and re-assembled to meet the changing workload needs [2]. Assembling the hardware logically based on a resource pool allows customizing the DC configuration to meet the application service level objectives. Study in [3] has shown that disaggregation of individual DC components is feasible with a high speed interconnect technology operating at Tera bit per second (Tbps) and low latency.

Several studies quantified the performance benefits of reconfigurable computing platforms based on parallelization of applications such as database search through the Catapult project [4] by Microsoft and database analytics projects by IBM [5] and Google [6]. In parallel to software optimization processes, reconfigurable architectures and their integration to the DCs have been investigated based on novel architectures by Xilinx that couple conventional DRAMs with serial-attached flash (Memcached design) [7] by Intel that couple Xeon processor with FPGA [8], and by IBM Power-8 system that relies on a novel accelerator interface [9]. From composable data center design point of view, Memcached designs are attractive as they lend themselves naturally to directly attaching FPGA to the network [10]. In the physically separated design approach, the integrity of the information stored through a hierarchy of memory units, each of which may rely on different technologies, and the reliability of the information exchanged between the heterogonous set of computing units becomes a critical concern. In the memory hierarchy from storage to main memory and cache, the underlying technology determines the capacity, latency and reliability of the memory subsystem. Alternative resistance based architectures such as spin-transfer-torque magnetic random access memory (STT-MRAM [11]), phase-change random access memory (PCRAM [12]), and resistive random access memory (RRAM [13]) are being investigated to address the issue of scaling down the charge based memories (Static Random-Access Memory (SRAM), Dynamic Random-Access Memory (DRAM), flash, Solid-State Drive (SSD)) [14]. We assume that future generation composable DCs will employ a combination of these memory technologies. Considering that the future large scale datacenters will cope with exabytes of data to serve applications such as data analytics and database search, accessibility of the entire database at a time scale that matches the computation capability of the system will be the main barrier in responding to the ever growing workload demands in an energy efficient way [15]. Advances in the field of flash management framework have played a critical role in improving the performance through algorithmic approaches that improve address mapping, wear-levelling, garbage collection, error correction and bad block management for the flash file system; and in the field of flash controller hardware architectures with multi-chip parallelism for serving data requests concurrently through parallel channels. As a result of these advances, the technology choice for storage in large scale datacenters has seen a shift towards SSDs with their superior low-power and high-throughput performance characteristics. For example, the internet search company Baidu relies on data centers with hundreds of thou-

978-1-5090-2991-4/17/$31.00 ©2017 European Union

sands of SSDs [16] with a reported order of magnitude greater throughput over the conventional hard disks. Flash based storage with its superior memory density and random access performance over hard disks is a promising technology for data analytics applications as demonstrated by several studies on nearest neighbor search, graph traversal, and string search [17].
The rest of the paper is organized as follows. In Section II we present the concept of error correction for flash memories. In Section III, we give an overview of the iterative decoding algorithms. In section IV, we show our simulation setup and hardware designs. Section V presents the hardware performance analysis. Section VI shows the decoding performance analysis, and finaly Section VII concludes the paper.

II. Error Correction for Flash Memories

A. Overview of hardware architectures

In a flash architecture an array of memory chips are coupled with a dedicated ASIC or embedded processor based controller, which manages the generation of voltage waveforms for reading, writing and erasing memory cells. Lifetime of a flash memory depends on the effectiveness of the erase and write operations that are managed by the Flash Translation Layer (FTL) through complex operations such as garbage collection, address mapping, and wear leveling [18]. Algorithms for each of these processes are either customized for the types of workloads that will be primarily served by the data center based on their memory access characteristics or designed for worst-case scenarios as a general purpose solution. Customized solutions lack the flexibility of responding to the dynamically changing read/write demands of the applications, whereas general purpose solutions can not guarantee satisfying their performance requirements in terms of resource usage, effective storage usage, power consumption, throughput and reliability [19]. Next generation standards for flash memory are expected to include probes for the controller so that new and emerging reading or writing methods can be incorporated [20]. Ability to customize the flash memory controller on a need basis based on the performance requirements or application memory access patterns of the datacenter workloads dynamically through tunable controller architecture parameters has been discussed with cases studies in the literature [16, 21, 22] under the commonly used "Software-defined flash" term. FPGAs pose as a seductive tool in this context to deliver capability on-the-fly by morphing hardware configurations dynamically with software and by implementing power efficient application-specific controller algorithms. Recently several architectures have been introduced with a direct interface between the FPGA and the flash chip array, in which FPGA is utilized as a data compression engine in order to reduce the amount of data traffic on the storage and as a near-data parallel computation engine [23] for accelerating data analytics tasks [21, 22, 24, 25] with successful deployment in commercial data centers [16].

Flash storage is read/written by pages, erased by blocks and must be erased prior to an in-place rewrite. In the SDF architectures, multiple FPGAs are utilized to exploit the channel level parallelism. For example, in [16] architecture has four Spartan-6 FPGAs each supporting 11 flash channels and two flash chips per channel with page size of 8KB and block size of 8MB, in which a Virtex-5 FPGA acts as the bridge between PCIe and each channel. Similarly in [22], the architecture is formed of three FPGAs each supporting 17 flash channels with four flash chips per channel. Instead of using a dedicated interface, this architecture allows each of the three FPGAs acting as a bridge between the PCIe and the channels. In [21], similar to [16], SDF architecture uses an Xilinx Virtex-7 FPGA as the bridge between the host Xeon processor and the channels through PCIe. The controller functionality is implemented through two Artix-7 FPGAs with four flash chips per FPGA.

B. Reliability challenge for SDF in datacenters

The speed of degradation and the Bit Error Rate (BER) of a flash memory are affected by the amount and frequency of the erase/write operations [26]. A workload that repeatedly writes into the same block results with rapid aging of that block. Several fault tolerance techniques have been introduced to address this problem by either reducing such operations or wear leveling through redistribution of those operations over all blocks evenly. Even though the application of fault tolerance techniques implemented through the flash controller improves the lifetime of the flash, they do not resolve the change in reliability level of the flash over time. In this context, there are studies on utilizing error correction techniques as a controller task [21][19]. As the probability of errors increases over time due to degradation, adaptive Error Correction Code (ECC) schemes with Bose-Chaudhuri-Hocquenghem (BCH) architecture have been introduced in the literature for balancing the resource requirements, error correction performance and hardware performance [27, 28]. As the number of errors to correct increases, the number of parity bits are increased proportionally to increase the ECC strength in this adaptive model. The study in [19] extends this tradeoff analysis to studying the benefits of combining optimized write algorithms with runtime-adaptable ECC schemes.

Given that FPGAs are now becoming an interesting candidate for implementing flexible and programmable controllers, and based on the assumption that the underlying fast interconnect technology is available for the future generation of data centers to support composable architectures, our aim is to utilize the FPGA resources for error correction in the disaggregated memory hierarchy of the datacenter to offer Error Correction as a Service (ECaS) in the DC software stack. We propose a software defined error correction scheme managed by the DC middleware, where the latency and reliability characteristics of the specific memory subsystem determines the type of error correction method to utilize. We benefit from the massive amount of fine grained parallelism offered by the FPGA architecture that ideally matches with the bitwise intensive error correction operations and take advantage of the reconfigurability of the FPGA

for the middleware to determine error correction mode and manage the available pool of FPGAs. For example memory unit that requires frequent reads/writes cannot tolerate soft decision based error correction methods as they require large amount of FPGA resources and offer much lower throughput than hard-decision based methods. In that case, the middleware would assign a light weight decoder to execute on the FPGA for a high throughput and energy efficient error correction service. On the other hand, memory subsystem with less frequent accesses can tolerate the low-latency decoding algorithm for error correction. Based on the location, a multi-mode error correction can be implemented. We experimentally show that the latency efficiency can be insured through the software defined ECaS for the DCs.

C. ECC for Flash memory based on LDPC Codes

While flash memories are rapidly gaining prominence due to their low power consumption and heat dissipation, as well as high access speeds, they still face a scaling problem due to their high costs. Flash memory cell sizes must shrink significantly in order to reduce the cost and increase capacity, which causes severe and unavoidable degradation in the reliability of the stored bits, making current state-of-the-art error-correction coding solutions based on BCH codes inadequate to meet the desired reliability requirements. Therefore, there is a strong need in the market for better error-correction algorithms that can provide reliability enhancements at reduced complexity and cost. In the latest Grand Challenges document, the International Technology Roadmap for Semiconductors (ITRS) identifies the development of methods for ensuring reliability of flash memories as critical [29].

In this paper we discuss our findings in error-correction solution based on Low-Density Parity-Check (LDPC) codes and iterative decoding algorithms that address the requirements of the next-generation flash memory systems. Following the bit-level error models utilized in [30], a binary symmetric channel (BSC) with the crossover probability of α is adopted for the bits stored in memories in this paper. The parameter α is the upset probability of Flash memory. LDPC codes provide significant improvement in error correction compared to the BCH codes which are currently used as ECC in flash memory systems. As a result, the most recent flash controllers have started to replace the BCH codes with LDPC codes but at the expense of increased complexity [31, 32]. The main limitation of the existing solutions comes from the high decoding complexity required to achieve the level of error correction needed for flash memories. The high complexity, which translates to power-hungry memory controllers, is due to specific high-density LDPC code structures and complex iterative decoders that consume significant hardware resources. Our codes and decoders are developed using a unique design approach, and are able to provide significant gains in reliability and speed, together with a reduced-complexity implementation. We note that presentations on LDPC codes dominated ECC sessions at all Flash Memory Summits since 2012 (see our recent presentations and references therein [33, 34]).

The known problem of LDPC codes is that they suffer from an abrupt degradation in their error-rate performance known as the error floor. This is one of the main challenges as flash-memory SSDs require extremely low error-rates below 10^{-10}-10^{-12}. In order to combat this problem, current state-of-the-art decoder solutions use computationally intensive decoders as well as post-processing techniques to improve the reliability but suffer from added costs of higher complexity, power consumption, and latency. In this paper we present decoder based on extremely low-complexity decoding algorithms known as bit-flipping (BF). The BF algorithms use a single bit precision of node values. The check node unit is an XOR while the variable node unit requires only majority logic Boolean function.

The BF algorithms significantly reduce the required hardware resources due to their simple computation units, but lead to a non-negligible performance loss compared to BP or MS algorithms. In order to improve the performance while keeping the low complexity, many modifications of BF decoders have been introduced. Wadayama *et. al* [35] proposed a BP algorithm that in addition to local computation also use involved *global* computation. A rule for flipping a value of a bit is based on the *energy function* of that variable.

Global minimization of the energy functions coupled with the gradient descent algorithm was shown to greatly improve convergence. This makes the algorithm, known as *Gradient Descent Bit-Flipping* (GDBF), superior in decoding performance over the BF. Despite its performance, the GDBF still exhibits error floor. We have analyzed thoroughly weaknesses of the GDBF, and found a way to significantly improve it. Our method [36] called *Probabilistic Gradient Descent Bit-Flipping* (PGDBF) introduces random perturbation of messages. Similar to the probabilistic bit flipping algorithm (PBFA) [37] in which a variable node value is flipped with some probability $p < 1$ instead of being flipped by default, our algorithm also uses randomness, but unlike the PBFA, it is based on global computations needed for the gradient descent. Surprisingly, if a decoder is implemented carefully, the hardware complexity overhead required to perform these global computations is compensated by simplicity of the node update computations, resulting in overall complexity significantly lower than that of BP-like algorithm such as min-sum of the same or lower throughput [38].

III. Iterative Decoding Algorithms

An LDPC code is defined by a sparse parity-check matrix H with size (M, N), where $N > M$. A codeword is a vector $\mathbf{x} = (x_1, x_2, \ldots, x_N) \in \{0,1\}^N$ which satisfies $H\mathbf{x}^T = 0$. We denote by $\mathbf{y} = \{y_1, y_2, \ldots, y_N\} \in \{0,1\}^N$ the output of a BSC channel, in which the bits of the transmitted codeword $\mathbf{x}$ have been flipped with crossover probability α. The graphical representation of an LDPC code is a bipartite graph called Tanner graph (Figure 1) composed of two types of nodes, the Variable Node Unit (VNUs) v_n, $n = 1, \ldots, N$ and the Check Node Unit (CNUs) c_m, $m = 1, \ldots, M$. In the Tanner graph, a

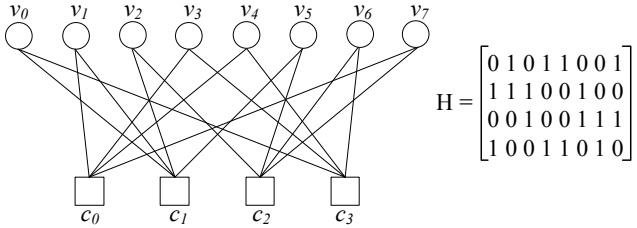

Figure 1: Tanner graph (left) and its parity check matrix (right).

VNU v_n is connected to a CNU c_m if $H(m,n) = 1$. Let us also denote $\mathcal{N}(v_n)$ the set of CNUs connected to the VNU v_n, with a connection degree $d_{v_n} = |\mathcal{N}(v_n)|$, and denote $\mathcal{N}(c_m)$ the set of VNUs connected to the CNU c_m, with a connection degree $d_{c_m} = |\mathcal{N}(c_m)|$. We will study in this paper only regular $d_v = 3$ and $d_v = 4$ LDPC codes for BSC channel.

A. Gradient Descent Bit Flipping Concept

The GDBF algorithm has been introduced by Wadayama *et al*. in [35]. This algorithm is derived from a gradient descent formulation and it consists of finding the most suitable bits to be flipped in order to maximize a pre-defined objective function. The GDBF algorithm shows an error correction capability superior than most known BF algorithms while still maintaining a lower hardware complexity compared to the soft decision decoders. In [36], we proposed to incorporate a probabilistic feature in the flipping step, inspired from the probabilistic BF algorithms of [37]. In the PGDBF decoder, all the bits that satisfy the gradient descent condition are not flipped by default, but instead, only a randomly chosen fraction p of them are flipped. Interestingly, this small modification of the GDBF algorithm led to a large performance improvement, with error correction capability approaching the soft-decision message passing decoders [36]. An efficient hardware (HW) implementation of the PGDBF decoder is proposed in this paper, which minimizes the resource overhead needed to implement the random perturbations and the maximum finder of the PGDBF.

B. Probabilistic Gradient Descent Bit Flipping Decoder (PGDBF)

A BF decoder is defined as an iterative update of the variable node values over the decoding iterations. We denote in this paper by $v_n^{(k)}$ the value of the VN v_n at the k-th iteration. We correspondingly denote by $c_m^{(k)}$ the binary value of the CN c_m parity check node at iteration k, which indicates whether the fact that the m-*th* parity-check equation is satisfied or not. The BF decoding process is terminated when all CNs values are satisfied or a maximum number of iteration It_{max} is reached. The CN calculation in BF algorithms can be written as

$$c_m^{(k)} = \bigoplus_{v_n \in \mathcal{N}(c_m)} v_n^{(k)},$$

where $\bigoplus$ is the bit-wise exclusive-OR (XOR) operation.

For the n-*th* VN calculation, the update rule uses the information on satisfiability of the neighboring CN $\mathcal{N}(v_n)$ to keep or flip the value of $v_n^{(k)}$. In the case of GDBF algorithms, a function called *inversion function* or *energy function* is defined for each VN, and is used to evaluate whether the value $v_n^{(k)}$ should be flipped or not. The original GDBF has been proposed for the AWGN channel [35] and the energy function was defined as in (1), where γ_n is the Log-Likelihood Ratio (LLR) received from AWGN channel.

$$\Lambda_{v_n}^{(k)} = (1 - 2\, v_n^{(k)})\gamma_n + \sum_{c_m \in \mathcal{N}(v_n)} (1 - 2\, c_m^{(k)}) \qquad (1)$$

For the GDBF on the AWGN channel, the energy function is real valued, and has a unique minimum, corresponding to the bit with lowest reliability. In [35], two modes for the bit-flipping rule at iteration k are proposed: either only the bit having smallest energy function is flipped (single flip), or a group of bits having energy function lower than a predefined threshold are flipped (multiple flips).

For the BSC channel, the energy function can be modified with the following equation:

$$E_{v_n}^{(k)} = v_n^{(k)} \bigoplus y_n + \sum_{c_m \in \mathcal{N}(v_n)} c_m^{(k)} \qquad (2)$$

In this case, the energy function is an integer, varies from 0 to $(d_{v_n} + 1)$, and the bits which have the maximum value $E_{max}^{(k)} = \max_n E_{v_n}^{(k)}$ are flipped. Due to the integer representation of the energy function, many bits are likely to have the maximum energy, leading to the multiple flips mode. Let us use an indicator variable to indicate the VNs which have the maximum energy at iteration k, i.e. $I_n^{(k)} = 1$ if $E_{v_n}^{(k)} = E_{max}^{(k)}$, and $I_n^{(k)} = 0$ otherwise. The fact that the number of bits to be flipped cannot be precisely controlled induces a negative impact to the convergence of the GDBF, as the analysis of [36] shows. To avoid this effect, the PGDBF has been proposed with the following modification: instead of flipping all the bits with maximum energy function value, only a random fraction of those bits are flipped. The random fraction is fixed to a pre-defined value $p_n^{(k)}$, which could be different for each VN and each iteration. In this work, we restrict ourself to the case for which $p_n^{(k)}$ are constant for all iterations and all VNs, denoted $p_0 \in [0, 1]$ hereafter.

The PGDBF of [36] can be implemented using a sequence of N random bits, generated following a Bernoulli distribution $\mathcal{B}$ with parameter p_0, with different realizations at each iteration. We denote the random generator (RG) sequence at the k-*th* iteration by $R^{(k)}$. In the GDBF algorithm, a VN $v_n^{(k)}$ at iteration k is flipped (is XOR-ed by '1') when its energy function is a maximum (the indicator variable is $I_n^{(k)} = 1$) while in PGDBF, a VN $v_n^{(k)}$ is flipped *if and only if* the two conditions $I_n^{(k)} = 1$ and $R_n^{(k)} = 1$, are both satisfied. The PGDBF algorithm is presented in Algorithm 1. A

Algorithm 1: Probabilistic Gradient Descent Bit-Flipping

Initialization $k = 0$, $v_n^{(0)} \leftarrow y_n, n = 1, \ldots, N$.
$s = H\mathbf{v}^{(0)^T}$ mod 2
while $s \neq 0$ *and* $k \leq It_{max}$ **do**
  Generate $R_n^{(k)}$, $n = 1, \ldots, N$, from $\mathcal{B}(p_0)$.
  Compute $E_{v_n}^{(k)}$, $n = 1, \ldots, N$, using Eq. (2).
  $E_{max}^{(k)} = max_n(E_{v_n}^{(k)})$,
  for $n = 1, \ldots, N$ **do**
    if $E_{v_n}^{(k)} = E_{max}^{(k)}$ **and** $R_n^{(k)} = 1$ **then**
      $v_n^{(k+1)} = v_n^{(k)} \oplus 1$
    end if
  end for
  $s = H\mathbf{v}^{(k+1)^T}$ mod 2
  $k = k + 1$
end while
Output: $\mathbf{v}^{(k)}$

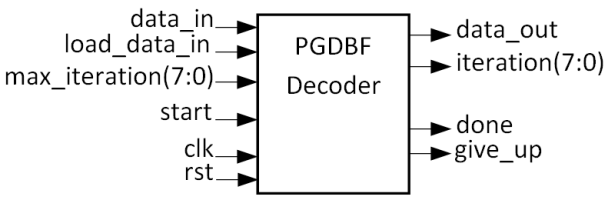

Figure 2: Top level view of PGDBF decoder.

configurable hardware model of the PGDBF decoder was designed using the VHDL Hardware Description Language (VHDL) and synthesized using Xilinx Synthesis Tool. Behavioral and post-synthesis simulations were carried out using ModelSim. The top level diagram of the PGDBF decoder as implemented is shown in Figure 2 The overall of the decoder is synchronized by clock "clk" and reset "rst" signals inputs. The maximum number of iterations is determined by the value supplied at the input "*max_iteration*". When the "*load_data_in*" input is high, the BSC channel values (bits '0' or '1') are fed into the decoder using a shift register. The decoding process is initiated by "start" signal. After decoding we have two possibilities: Either the decoder converges to codeword, in this case the "done" signal is high and the number of iterations used for decoding is obtained from the "iteration" output signal, or the maximum number of iteration is reached without converging to a codeword and in this case the "*give_up*" signal will be high. The proposed hardware architecture of the PGDBF decoder is showed in Figure 3.

IV. Simulation Setup and Hardware Design

Our simulation environment includes the GDBF, PGDBF, MinSum, and Offset MinSum implementations in C programming language. We evaluate the hardware performance and resource utilization using 1296-bit LDPC codes [39], with rate of 0.5. We design and implement each decoder on the Xilinx Virtex-6 FPGA (vc6vlx240t ff1156 target speed -3) and conduct post placement and routing analysis over hardware cost in terms of logic and register usage, and hardware performance in terms of maximum clock rate, and average throughput. Similar to other studies ([40], [41]), we calculate the system throughput using Equation 3.

$$SystemThroughput = \frac{CodeLength \times MaxClockRate}{AvgIteration \times CyclesPerIteration} \quad (3)$$

The randomness in the PGDBF decoder is generated by a Linear Feedback Shift Register (LFSR) based random number generator (RNG). Random number generators have been studied in terms of their quality and complexity in the literature extensively [42]. Our design choice favors simplicity with the objective of a light-weight decoder architecture in terms of its hardware resource requirement. Therefore, we use an LFSR based random number generator to feed a 1-bit random value per clock cycle in each VNU as shown in Figure 3. During the iterations between the CNUs and VNUs, we generate 1-bit and use a shift register of size N to feed into each of the VNUs.

V. Hardware Performance Analysis

We compare the decoding algorithms based on their hardware implementations for dv3 and dv4 using Table I. Even though modification to GDBF involved including a random number generator and an additional input to each VNU, the maximum clock rate and throughput differences between the GDBF and PGDBF designs are negligible for both codes (dv3 and dv4). However, the resource requirements in terms of register and slice LUT usage for PGDBF are higher compared to the GDBF for both codes. Register overhead of the PGDBF implementation includes the 1296 bits to store the 1-bit random number for each VNU and the 32-bit shift register to implement the RNG. The increase in resource usage is a reasonable trade-off for improving the decoding performance. Relative to both MS [43], [44] and OMS implementations, which require large number of registers and LUTs, the PGDBF is superior in terms of its maximum operational clock rate and throughput performance. Here, we note that for fair comparison of register usage we implemented the MS and OMS without BRAMs resources (only distributed memory). This implementation choice has direct impact on their maximum clock rates.

We further optimize the average number of iterations for PGDBF by applying the randomness only after a subsequent iterations, i.e. decoding by GDBF algorithm for $1 \leq i \leq 10$ and PGDBF algorithm with $10 < i \leq i_{max}$. This modification comes from the fact that some error patterns can be easily corrected by GDBF in few iterations without the strength of PGDBF (which decelerates the converging speed).

VI. LDPC Decoding Performance Analysis

Figure 4 shows the FER performance of Bit Flipping (BF), GDBF, PGDBF, MS and OMS based on regular

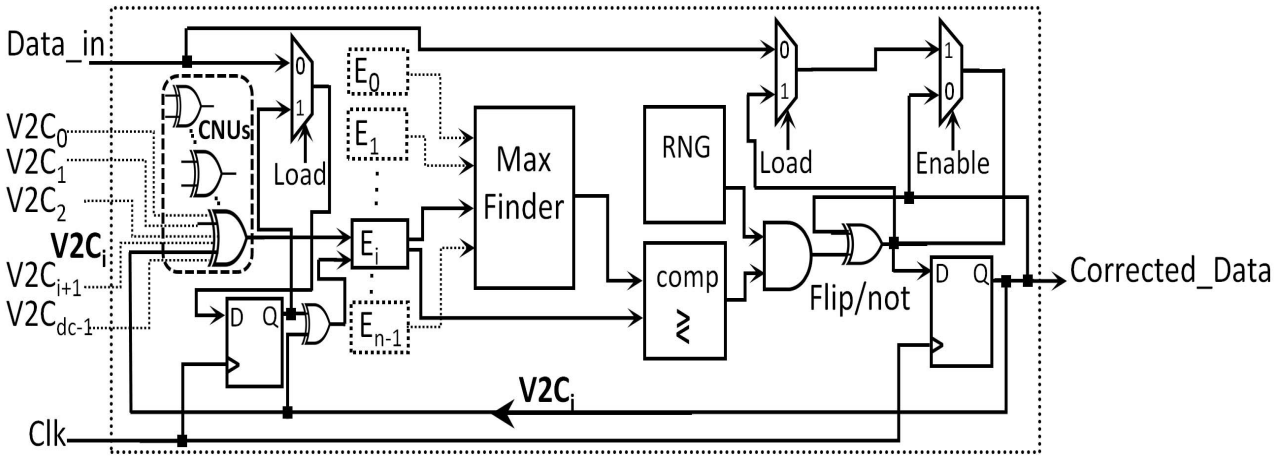

Figure 3: Overall PGDBF decoder hardware architecture

Table I: Resource usage, clock rate and throughput comparison based on FPGA implementations of GDBF, PGDBF, MS, and OMS for dv3 and dv4. (* 3 layers for dv3 and 4 layers for dv4)

	1-bit register		Slice LUTs		FMax (MHz)		Average Throughput (Mbps) (2 clock cycles /layer for MS and OMS) (1 clock cycle and 20 iterations for GDBF and PGDBF	
Design	dv3	dv4	dv3	dv4	dv3	dv4	dv3	dv4
GDBF	2613 (0%)	2613 (0%)	12331 (0%)	13228 (0%)	110	93	7128 (0%)	6026 (0%)
PGDBF	**3942 (+50.86%)**	**3942 (+50.86%)**	**13653 (+10.72%)**	**14625 (+10.56%)**	**106**	**90**	**6868 (-3.64%)**	**5832 (-3.22%)**
MS*	37610 (14.4 x)	42794 (16.37 x)	80721 (6.54 x)	84165 (6.36 x)	55.5	55.5	600 (/11.8)	450 (/13.4)
OMS*	37610 (14.4 x)	42794 (16.37 x)	86445 (7 x)	95721 (7.23 x)	54.9	53.6	593 (/12)	434 (/13.88)

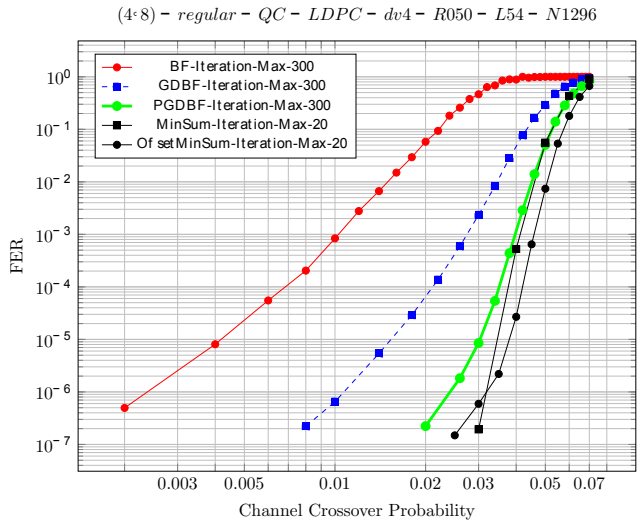

Figure 4: Performance comparison between BF, GDBF, PGDBF ($p = 0.9$), Quantized MS and Quantized Offset Min-Sum (OMS) with offset factor of 1: FER vs. probability of error introduced to each bit of the 1296-bit codeword with dv= 4, dc= 8, and Rate = 0.5

LDPC code with rate of 0.5. In general, PGDBF decoder is performing much better than BF and deterministic GDBF decoder and approaching to soft decision decoder (MS and OMS). The gap between PGDBF and GDBF in the error floor region quantifies the dramatic improvement (more than two orders of magnitude at $\alpha = 0.02$) achieved by utilization of hardware friendly and simple random number generator with a negligible loss in throughput performance.

VII. Conclusion

In this paper we quantified the hardware cost and performance of the GDBF with probabilistic module (PGDBF decoder). We compared them to two other soft-decision based decoders (MS and OMS). We showed that with very small loss in throughput, we improved the decoding performance of the GDBF dramatically. The simulations and FPGA implementation confirm that PGDBF decoder offers very low-complexity, high throughput and good performance. These variety of LDPC decoders are a promising candidates for software defined error correction in data center.

Acknowledgments

This work was supported by the National Science Foundation under Grant ECCS-1500170 and CCF-1314147, the Indo-US Science and Technology Forum

(IUSSTF) through the Joint Networked Center for Data Storage Research (JC-16-2014-US), and the French ANR project NAND under grant agreement ANR-15CE25-0006-01.

References

[1] Hpe synergy. [Online]. Available: https://www.hpe.com/us/en/integrated-systems/synergy.html

[2] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends, "Rack-scale disaggregated cloud data centers: The dredbox project vision," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 690–695.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402967

[4] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 13–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=2665671.2665678

[5] B. Sukhwani, H. Min, M. Thoennes, P. Dube, B. Brezzo, S. Asaad, and D. E. Dillenberger, "Database analytics: A reconfigurable-computing approach," *IEEE Micro*, vol. 34, no. 1, pp. 19–29, Jan 2014.

[6] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan, "Meet the walkers: Accelerating index traversals for in-memory databases," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: ACM, 2013, pp. 468–479. [Online]. Available: http://doi.acm.org/10.1145/2540708.2540748

[7] M. Blott, K. Karras, L. Liu, K. Vissers, J. Br, and Z. Istvn, "Achieving 10gbps line-rate key-value stores with fpgas," in *5th USENIX Workshop on Hot Topics in Cloud Computing*, ser. USENIX, 2013.

[8] P. Gupta, "Xeon+fpga platform for the data center," in *The Fourth Workshop on the Intersections of Computer Architecture and Reconfigurable Logic*, 2015. [Online]. Available: http://www.ece.cmu.edu/~calcm/carl/lib/exe/fetch.php?media=carl15-gupta.pdf

[9] J. Stuecheli, B. Blaner, C. R. Johns, and M. S. Siegel, "Capi: A coherent accelerator processor interface," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 7:1–7:7, Jan 2015.

[10] M. Lavasani, H. Angepat, and D. Chiou, "An fpga-based in-line accelerator for memcached," *IEEE Computer Architecture Letters*, vol. 13, no. 2, pp. 57–60, July 2014.

[11] J. G. Zhu, "Magnetoresistive random access memory: The path to competitiveness and scalability," *Proceedings of the IEEE*, vol. 96, no. 11, pp. 1786–1798, Nov 2008.

[12] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec 2010.

[13] H. S. P. Wong, H. Y. Lee, S. Yu, Y. S. Chen, Y. Wu, P. S. Chen, B. Lee, F. T. Chen, and M. J. Tsai, "Metaloxide rram," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, June 2012.

[14] S. Yu and P. Y. Chen, "Emerging memory technologies: Recent trends and prospects," *IEEE Solid-State Circuits Magazine*, vol. 8, no. 2, pp. 43–56, Spring 2016.

[15] J. S. Vetter and S. Mittal, "Opportunities for nonvolatile memory systems in extreme-scale high-performance computing," *Computing in Science Engineering*, vol. 17, no. 2, pp. 73–82, Mar 2015.

[16] J. Ouyang, S. Lin, S. Jiang, Z. Hou, Y. Wang, and Y. Wang, "Sdf: Software-defined flash for web-scale internet storage systems," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 471–484. [Online]. Available: http://doi.acm.org/10.1145/2541940.2541959

[17] S. H. Kang, D. H. Koo, W. H. Kang, and S. W. Lee, "A case for flash memory ssd in hadoop applications," *International Journal of Control and Automation*, vol. 6, no. 1, pp. 201–210, February 2013.

[18] P.-L. Wu, Y.-H. Chang, and T. W. Kuo, "A file-system-aware ftl design for flash-memory storage systems," in *2009 Design, Automation Test in Europe Conference Exhibition*, April 2009, pp. 393–398.

[19] D. Bertozzi, S. D. Carlo, S. Galfano, M. Indaco, P. Olivo, P. Prinetto, and C. Zambelli, "Performance and reliability analysis of cross-layer optimizations of nand flash controllers," *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 1, pp. 7:1–7:24, Jan. 2015. [Online]. Available: http://doi.acm.org/10.1145/2629562

[20] Open nand flash interface specification 4.0 revision 04 02 2014. [Online]. Available: http://www.onfi.org/

[21] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu, and Arvind, "Bluedbm: An appliance for big data analytics," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015, pp. 1–13. [Online]. Available: http://doi.acm.org/10.1145/2749469.2750412

[22] J. Zhang, D. Feng, J. Gao, W. Tong, J. Liu, Y. Hua, Y. Gao, C. Fang, W. Xia, F. Fu, and Y. Li, "Application-aware and software-defined ssd scheme for tencent large-scale storage system," in *2016 IEEE 22nd International Conference on Parallel*

and Distributed Systems (ICPADS), Dec 2016, pp. 482–490.

[23] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho, J. Jeong, and D. Chang, "Biscuit: A framework for near-data processing of big data workloads," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 153–165. [Online]. Available: https://doi.org/10.1109/ISCA.2016.23

[24] A. De, M. Gokhale, R. Gupta, and S. Swanson, "Minerva: Accelerating data analysis in next-generation ssds," in *Proceedings of the 2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, ser. FCCM '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 9–16. [Online]. Available: http://dx.doi.org/10.1109/FCCM.2013.46

[25] S.-W. Jun, M. Liu, K. E. Fleming, and Arvind, "Scalable multi-access flash store for big data analytics," in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, ser. FPGA '14. New York, NY, USA: ACM, 2014, pp. 55–64. [Online]. Available: http://doi.acm.org/10.1145/2554688.2554789

[26] K. Zhao, K. S. Venkataraman, X. Zhang, J. Li, N. Zheng, and T. Zhang, "Over-clocked ssd: Safely running beyond flash memory chip i/o clock specs," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2014, pp. 536–545.

[27] T. H. Chen, Y. Y. Hsiao, Y. T. Hsing, and C. W. Wu, "An adaptive-rate error correction scheme for nand flash memory," in *2009 27th IEEE VLSI Test Symposium*, May 2009, pp. 53–58.

[28] M. Fabiano, M. Indaco, S. D. Carlo, and P. Prinetto, "Design and optimization of adaptable {BCH} codecs for {NAND} flash memories," *Microprocessors and Microsystems*, vol. 37, no. 45, pp. 407 – 419, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933113000471

[29] "Itrs grand challenges2011 edition," 2011. [Online]. Available: http://www.src.org/program/grc/about/grandchallenges/

[30] C. H. Huang, Y. Li, and L. Dolecek, "Gallager b ldpc decoder with transient and permanent errors," *IEEE Transactions on Communications*, vol. 62, no. 1, pp. 15–28, January 2014.

[31] "Lsi shield technology, advanced error correction for high-density flash memory." [Online]. Available: http://www.lsi.com/downloads/Public/LSI_WP_Shield_Tech.pdf

[32] E. Yeo, "An LDPC-enabled flash controller in 40 nm CMOS," in *Proc. Flash Memory Summit*, Santa Clara, Aug. 12–15 2012. [Online]. Available: http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2012/Proceedings_Chrono_2012.html

[33] S. Planjery, D. Declercq, B. J. Reynwar, and B. Vasić, "Efficient fpga-based architectures of finite alphabet iterative decoders for flash memories," in *Non-volatile Memories Workshop*, San Diego, CA, Mar. 1-3 2015, pp. 1–2.

[34] S. Planjery, D. Declercq, B. J. Reynwar, and B. Vasić, "Low error-floor soft-decision finite alphabet iterative decoders," in *Flash Memory Summit 2016*, Santa Clara, CA, August 9 –12 2016, pp. 1–2.

[35] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding LDPC codes," *IEEE Transactions on Communications*, vol. 58, no. 6, pp. 1610–1614, June 2010.

[36] O.-A. Rasheed, , P. Ivanis, and B. Vasić, "Fault-tolerant probabilistic gradient-descent bit flipping decoders," *IEEE Commun. Letters*, vol. 18, no. 9, pp. 1487 – 1490, September 2014.

[37] N. Miladinovic and M. Fossorier, "Improved bit-flipping decoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 51, no. 4, pp. 1594–1606, April 2005.

[38] K. Le, F. Ghaffari, D. Declercq, and B. Vasi, "Efficient hardware implementation of probabilistic gradient descent bit-flipping," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 4, pp. 906–917, April 2017.

[39] T. Nguyen-Ly, K. Le, F. Ghaffari, A. Amaricai, O. Boncalo, V. Savin, and D. Declercq, "Fpga design of high throughput ldpc decoder based on imprecise offset min-sum decoding," in *IEEE 13th Int New Circuits and Systems (NEWCAS)*, June 2015, pp. 1–4.

[40] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans on Comm*, vol. 47, no. 5, pp. 673–680, May 1999.

[41] K. Le, D. Declercq, F. Ghaffari, C. Spagnol, P. Ivanis, and B. Vasic, "Efficient realization of probabilistic gradient descent bit flipping decoders," in *Proc. of the 21st 2015 IEEE Int Symp on Circuits and Systems (ISCAS)*, May 2015, pp. 1–4.

[42] N. Dek, T. Gyrfi, K. Mrton, L. Vacariu, and O. Cret, "Highly efficient true random number generator in fpga devices using phase-locked loops," in *2015 20th International Conference on Control Systems and Computer Science*, May 2015, pp. 453–458.

[43] X. Zhang and F. Cai, "Reduced-complexity extended min-sum check node processing for non-binary ldpc decoding," in *2010 53rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2010, pp. 737–740.

[44] X. Zhao, Z. Chen, X. Peng, D. Zhou, and S. Goto, "High-parallel performance-aware ldpc decoder ip core design for wimax," in *2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2013, pp. 1136–1139.