2017 IEEE 35th International Conference on Computer Design

LDPC-Based Adaptive Multi-Error Correction for 3D Memories

Mihai Lefter[1], George Voicu[1], Thomas Marconi[1], Valentin Savin[2], and Sorin Dan Cotofana[1]

[1]Computer Engineering Laboratory, Delft University of Technology, The Netherlands
[2]CEA-LETI, MINATEC, Grenoble, France
{*m.lefter,g.r.voicu, t.marconi, s.d.cotofana*}*@tudelft.nl, valentin.savin@cea.fr*

***Abstract*—In this paper we introduce a novel error resilient memory architecture potentially applicable to a large range of memory technologies. In contrast with state of the art memory error correction schemes, which rely on (extended Hamming) Error Correcting Codes (ECC), we make use of Low Density Parity Check (LDPC) codes due to their close to the Shannon performance limit error correction capabilities. To allow for a cost-effective implementation we build our approach on top of a 3D memory organization which inherently fast and customizable wide-I/O vertical access allows for a smooth transfer of the required LDPC long code-words to/from an error correction dedicated die. To make the error correction process transparent to the memory users, e.g., processing cores, we propose an online memory scrubbing policy that performs the LDPC-based error detection and correction decoupled from the normal memory operation. For evaluation purposes we consider 3D memories protected by the proposed LDPC mechanism with various data width codes implementations. Simulation results indicate that our proposal clearly outperforms state of the art ECC schemes with fault tolerance improvements by a $4710\times$ factor being obtained when compared to extended Hamming ECC. Furthermore, we evaluate instances of the proposed memory concept equipped with different LDPC codecs implemented on a commercial 40nm low-power CMOS technology and evaluate them on actual memory traces in terms of error correction capability, area, latency, and energy. Our results indicate that the LDPC protected memories offer substantially improved error correction capabilities, when compared to state of the art extended Hamming ECC, being able to assure clean runs for memory error rates $\alpha < 3 \times 10^{-2}$, which demonstrate that our proposal can potentially successfully protect system on a chip memory systems even in very harsh environmental conditions.**

I. Introduction

Technology shrinking and increased integration factor allow, on one hand, for continuous Integrated Circuits (ICs) performance improvements. On the other hand, ICs are more prone to different defect types during the manufacturing process [1] and to in field degradations [2]. Multi-Bit Upsets (MBUs) become much more frequent [3], [4] with a maximum MBU bit multiplicity of over 100 bits being predicted for 32 and 22nm SRAM generations [5]. Thus, traditional Single-Error Correction (SEC) ECCs with column interleaving [6] cannot any-longer mitigate this large amount of MBUs [7] and powerful but cost effective techniques to detect and correct multiple memory errors are becoming crucial for future SoC related developments [8].

Three dimensional stacked ICs (3D-SICs) based on Through-Silicon-Via (TSV) interconnects [9] further boost increased transistor density and performance [10]–[12] while facilitating dependable computing [12]–[14]. Various 3D memory designs have been proposed ever since the technology was introduced [15]–[18]. In particular for polyhedral memories [16], TSVs bundles are distributed across the entire memory footprint. This enables a bandwidth amount and a wide interconnect width not achievable in planar counterparts, which opens new avenues for memory error correction [19].

In this paper we propose a novel memory error correction mechanism which takes advantage of the 3D memories flexible, powerful, and wide data access capabilities. Our approach relies on performing Low Density Parity Check (LDPC) encoding/decoding [20] on large codewords, which can be quickly transferred over the 3D memory TSVs to a dedicated die, on which the actual error correction and detection is performed. An important component of the mechanism consists of an online memory scrubbing policy, which enables transparent error detection and correction. In case the scrubbing process cannot keep the pace in cleaning the memory bit-flips induced by harsh environmental conditions or aging effects, faulty data might be read by the processing units, which may result in unpredictable application behavior. To handle this issue, we propose an Operating System (OS) controlled adaptation mechanism which increases the memory access time but allows for memory integrity preservations even in extremely aggressive environments.

We evaluated our proposal by considering as a case study 3D memories protected by the proposed LDPC mechanism with various data width codes implementations. The simulation experiments indicate that our proposal outperforms state of the art extended Hamming ECC schemes in terms of error correction capability, being able to tolerate crossover probability rates (α) up to 8×10^{-2} for a targeted Word Error Rate (WER) of 10^{-6}. This translates into fault tolerance improvements by a $4710\times$ factor when compared to extended Hamming ECC. Next we evaluate instances of the proposed memory concept equipped with different LDPC codecs implemented in a commercial 40nm low-power CMOS technology on actual memory traces in terms of error correction capability, area, latency, and energy. Our results indicate that the LDPC based protection memories offer substantially improved error correction capabilities, when compared to state of the art extended Hamming ECC, being able to assure correct application execution for bit flip error rates $\alpha < 3 \times 10^{-2}$.

The outline of the paper is the following. In Section II we describe our LDPC-based error correction proposed approach. In Section III we evaluate the implications of our proposal and perform a comparison with state of the art memory error correction approaches. Section IV concludes the paper.

1063-6404/17 $31.00 © 2017 IEEE
DOI 10.1109/ICCD.2017.47

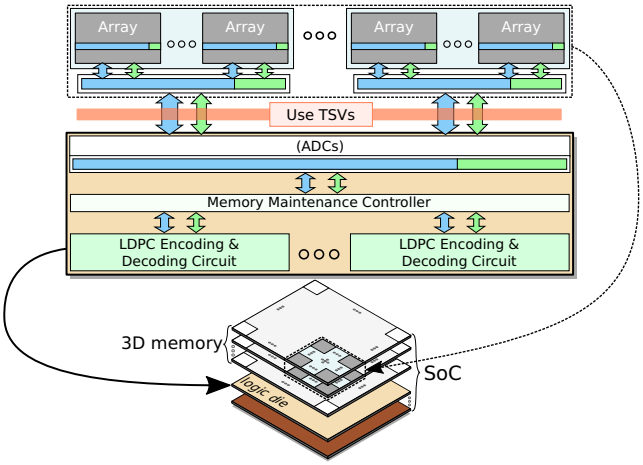

Figure 1. LDPC-based Error Correction for 3D Memories

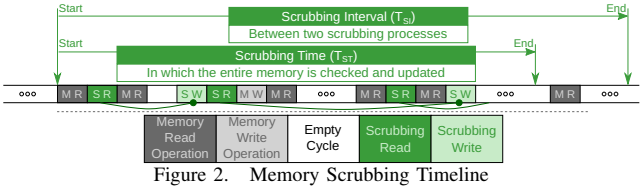

Figure 2. Memory Scrubbing Timeline

II. LDPC-based Error Correction Proposal

The proposed LDPC error protection mechanism depicted in Figure 1 consists in: (i) forming extended codewords by combining data and check bits from multiple memory sources, (ii) augmenting the 3D memory stack with extra logic dedicated to the LDPC encoding and decoding execution, (iii) employing TSVs to transfer the extended codewords on/from the dedicated Error Correction (EC) die, and (iv) utilizing an online memory scrubbing policy able to perform memory maintenance without interfering with data requests issued by the System on Chip (SoC) computation cores.

Memory maintenance is performed by means of a **scrubbing** procedure, i.e., at certain time periods (*scrubbing intervals*), the entire memory is scanned (in a time period which we call *scrubbing time*) and eventual errors are corrected. See Figure 2 for timing details. We mention that Memory Read (MR) and Memory Write (MW) operations handle memory I/O width data while Srubbing Read (SR) and Scrubbing Write (SW) operate on large data width values upper bounded by the number of TSVs in the 3D memory. The scrubbing maintenance operates in a transparent manner, i.e., in case of conflicts SR/SW accesses have lower priority than normal MR/MW accesses. Due to the 3D memory organization [16] multiple accesses can be served in parallel as long as no arrays and/or TSV conflicts are incurred.

The scrubbing procedure steps are depicted in Figure 3. At memory system start-up a scrubbing initialization process is performed (*step 0*). This may include, e.g., (i) memory initialization, (ii) LDPC codec allocation and instantiation as the codec dedicated tier may contain more than one LDPC codec, (iii) scrubbing start address(es) allocation. In *step 1* the LDPC codeword comprising parts located at the to be currently scrubbed address are brought on the codec tier. This step may

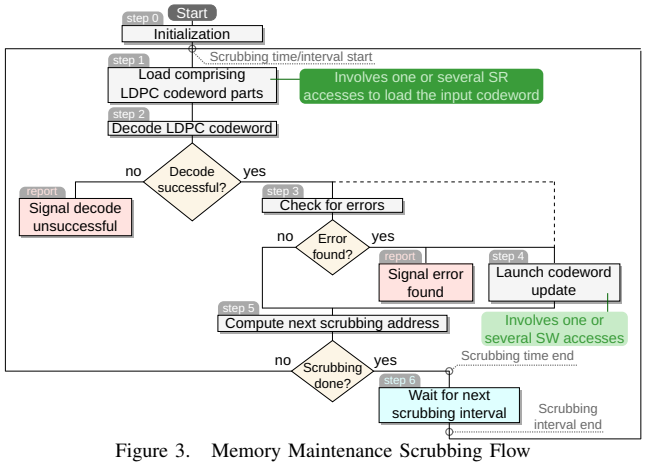

Figure 3. Memory Maintenance Scrubbing Flow

require several SR accesses in various arrays which can be performed serially or in parallel. When all the comprising codeword parts are loaded, they can be provided as input to the LDPC decoder such that *step 2*, i.e., the actual decoding, can proceed. In case the LDPC decoder fails to identify a valid codeword at *step 2* an exception is raised which treatment is up to the application (OS) policy in place. A successful decoding triggers *step 3*, in which possible errors are identified by comparing the decoder input codeword with the decoder output codeword. Eventual discovered errors are reported and a codeword update process is initiated (*step 4*). The codeword update process is similar to *step 1*, with the difference that now SW accesses accesses are performed. In *step 5* the next to be scrubbed address is computed and if this is the initial to be scrubbed address, the scrubbing iteration ends and *step 6* in which the scrubbing process is idle is entered, otherwise, the flow restarts from *step 1*. The scrubbing time depends on: (i) employed LDPC codec performance, (ii) memory latency (*steps 1,4*), (iii) LDPC codecs number (*step 2*), (iv) memory array and TSV conflicts (*steps 1,4*), (v) overall memory capacity, and, (vi) bit-error rate, which depends on aging and environmental aggression profile. Consequently, in order to diminish the scrubbing time, the scrubbing controller can take advantage of: (i) the 3D memory rich access mode set, and (ii) the low maintenance related accesses occurrence, to dynamically adapt its access schedule such that memory conflicts are minimized, if not completely avoided.

Even though from the user point of view the proposed memory system data write seems coding free, a launch of an encoding process (and sometimes also of a decoding process) is required when a write at address A invalidates all the codeword C check bits to which the data item at address A belongs. Consequently, the codeword C data bits need to be transferred via TSVs on the dedicated EC die in order to be re-encoded and written back into the memory. Depending on the actual bit error rate we may decide to place the invalidated message in the front of the scrubbing queue or to take no further action as there is a large chance that another write may occur within the same message in the close future. Nevertheless, a read from an invalidated message should immediately trigger the reconstruction of the check bits.

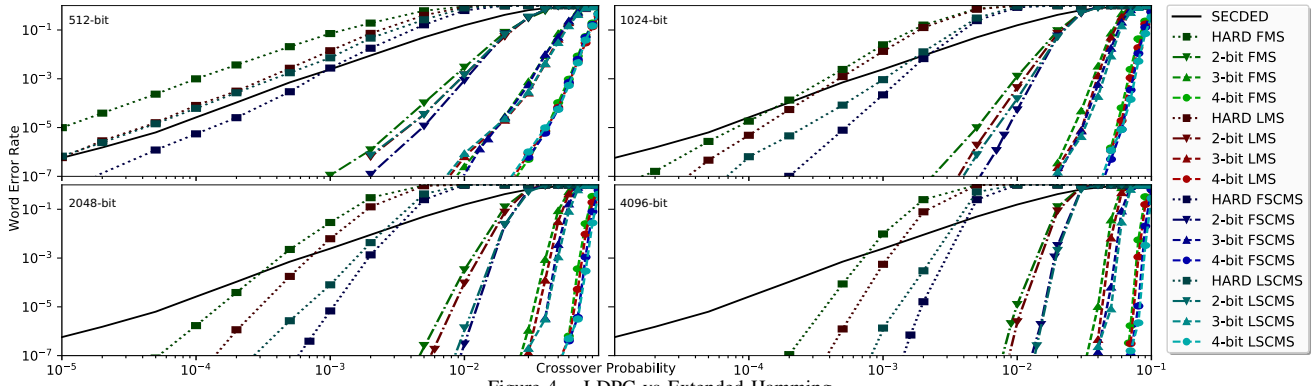

Figure 4. LDPC vs Extended Hamming

We refer to this write policy as *write invalidation* and we note that it may have an impact on the EC efficiency. If aging and environmental conditions ask for a better protection, a *write with on the fly encoding* operation should be employed. Thus, when a word write is requested at address A a series of actions have to take place. First, the codeword C (to which address A belongs) has to be loaded on the EC die. Next, C is decoded, the possible errors if any are corrected and the value at address A is replaced with the new to be written value and codeword C is re-encoded. Finally, C is stored back into the memory. Regardless of the write policy, errors could still affect memory integrity as bit-flips occurring between two scrubbing scans could propagate when read operations on those locations are performed. To avoid such situations the *read with on the fly* operation can be employed, such that a read request at address A (which is part of codeword C) comprises the following actions: (i) load the codeword C on the codec die, decode it, and correct possible errors, if any (ii) send the data value stored at A to the computation core that issued the request, and (iii) re-encode the codeword C, if errors were identified, and store it back to the memory.

If memory operates in highly aggressive environmental conditions or aging effects become predominant resulting in high error rate it is possible that the scrubbing process cannot keep the pace in cleaning the induced bit-flips. Such situations can be detected by the Operating System (OS), which can decide to change the ECC modus operandi and to enable *write with on the fly encoding* and *read with on the fly* operations. We note however that the execution of codec activities during read/write operations required when on the fly modes are activated might have a detrimental impact on SoC performance in terms of throughput, latency, and energy consumption. Since 3D memories are constructed by stacking identical memory dies which dimensions determine the IC footprint, the EC die might be able to accommodate several LDPC codecs. Hence, the memory maintenance process can be parallelized resulting in a higher error resilience. In addition, by placing on the ECC die LDPC codecs of different characteristics and sizes more options become available for the system adaptation to the environmental aggression level. In this way the 3D memory structure can be split into multiple ECC memory blocks on which different ECC mechanisms are applied when sensing that part of memory requires a more/less powerful protection.

Table I
LDPC ENCODERS AND DECODERS IN ASIC IMPLEMENTATIONS

	Energy/Bit (pJ/Bit)		Latency/Bit (ps/Bit)		Area/Bit (um^2/Bit)	
Instances	Dec	Enc	Dec	Enc	Dec	Enc
K2048_Z16	3041.99	0.19	19702.15	2.19	87.89	57.96
K2048_Z32	3417.96	0.19	10849.61	2.19	146.48	57.72
K2048_Z64	4877.93	0.20	6875.00	2.19	288.08	60.32
K4096_Z16	4077.14	0.41	20546.88	1.09	58.59	176.34
K4096_Z32	3549.80	0.41	10834.96	1.09	85.44	175.75
K4096_Z64	4667.96	0.40	6611.32	1.09	153.80	172.03
K4096_Z128	7929.68	0.41	4626.46	1.09	302.73	177.27

III. EVALUATION

To evaluate the upper bound performance of our proposal, we simulate memories protected by: (i) state of the art (64 data bits and 8 check bits) extended Hamming (**SECDED**), and (ii) the proposed LDPC-based mechanism with (512, 64), (1024, 128), (2048, 256), and (4096, 512) codeword sizes. For LDPC we consider Quasi-Cyclic (QC) codes with variable-nodes degree $d_v = 4$ and check-nodes degree $d_c = 36$, which make use of Layered Min-Sum (**LMS**), Flooded Min-Sum (**FMS**), Layered Self-Corrected Min-Sum (**LSCMS**), and Flooded Self-Corrected Min-Sum (**FSCMS**) decoders operating on 3-bit and 4-bit soft information inputs and internal exchanged messages. To simulate memory fault occurrence, we use a binary symmetric channel model with various crossover probabilities (α). The decoders performances in terms of WER are plotted in Figure 4. It can be noticed that: (i) when a $WER = 10^{-6}$ is targeted, the SECDED protected memory tolerates memory faults up to $\alpha = 1 \times 10^{-5}$ while the LDPC protected ones bear up to 8×10^{-2}, (ii) utilizing soft information as input significantly improves the decoding performance, and, (iii) the longer the employed LDPC code the higher the obtained benefit.

Physical synthesis on a commercial 40nm low-power CMOS was performed using Cadence Encounter RTL Compiler [21] for (2048, 256) and (4096, 512) codecs with their designs being automatically generated as in [22]. Various QC LDPC codes with different sub-matrix sizes (Z) were considered. The synthesis results are presented in Figure 5 and Table I from which one can observe that: (i) the sub-matrix size Z is the main design parameter from the hardware cost point of view, (ii) the decoder **K4096_Z16** is one of the most balanced options, (iii) the energy and the latency needed for encoding are 3 to 4 orders of magnitude smaller when compared to decoding, (iv) the encoding energy, latency, and area figures are relatively Z independent, and, (v) encoding

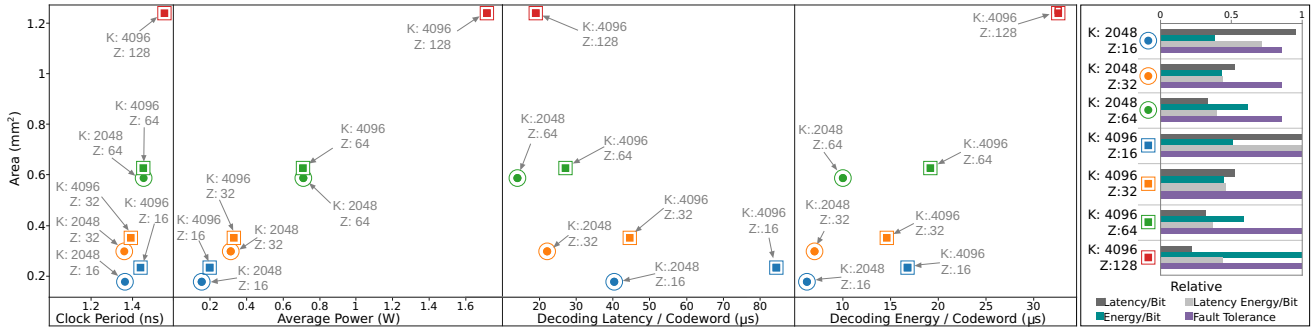

Figure 5. Area vs. Clock Period, Area vs. Power, Area vs. Decoding Latency, and Area vs. Decoding Energy

Memory Size (MB)	Memory Footprint (mm^2)	LDPC Decoder & Encoder Utilization (%)		Free Area Available on the EC die (mm^2)		Additional Decoders that can be Accomodated (#)	
		2048	4096	2048	4096	2048	4096
1	2.44	24	40	1.86	1.48	10	6
2	3.13	19	30	2.55	2.17	14	9
4	4.43	13	21	3.85	3.47	21	14
8	7.06	8	13	6.48	6.09	36	25

Figure 6. LDPC Error Correction Die Real Estate Utilization

smaller codes is more energy and area efficient at the cost of a longer latency per bit. In Figure 6 we provide an LDPC (**K2048/4096 Z16**) tier real estate utilization estimation for 4-dies 3D memories. Memory footprints are obtained with a modified Cacti 6 simulator [23] for 40nm CMOS technology as in [16]. We mention that the 2048-bit LDPC protection is assured by means of two parallel scrubbing blocks, which each having allocated half of the memory size.

Next we performed experiments on memory access traces considering a system based on an ARMv7-A processor with a two level cache hierarchy: (i) 32-kB instruction and data L1 caches, and (ii) a unified 4-MB L2 cache implemented as a 4 die 3D memory. We simulated the system with a modified gem5 simulator [24] in order to obtain access traces for the L2 cache when running a memory intensive SPEC CPU2000 [25] suite benchmarks subset. We run each benchmark for 1 million committed instructions and we injected transient faults into the memory at random data locations during each cycle. In Figure 7 we present: (i) the total number of erroneous 64-bit words which become visible to the executed application on average per run, and, (ii) the total number of runs when no error was propagated. It can be noticed from Figure 7 that the **SECDED** protection manages to significantly reduce the number of propagated 64-bit erroneous words only for $\alpha < 10^{-3}$ while **LDPC** protection significantly reduces the number of 64-bit erroneous propagated words already at $\alpha = 5 \times 10^{-2}$.

IV. Conclusion

In this paper we proposed and investigated an adaptive LDPC-based memory error correction mechanism best suited for 3D memories and potentially applicable to a large memory technologies range. Our results indicate that our proposed LDPC based memory protection offers substantially improved error correction capabilities when compared to state of the art extended Hamming ECC, being able to assure correct application execution for $\alpha < 3 \times 10^{-2}$ memory error rates.

		64b Erroneous Propagated Words (Average)				Clean Runs			
				LDPC				LDPC	
Benchmark	Alpha	NO EC	SECDED	2048	4096	NO EC	SECDED	2048	4096
gzip	6×10^{-2}	7816.01	7439.35	4186.59	6059.53	0	0	0	0
	5×10^{-2}	8906.20	8129.51	786.24	326.88	0	0	0	0
	4×10^{-2}	9728.93	8244.60	8.55	0.09	0	0	627	1977
	3×10^{-2}	9813.73	7285.47	0.01	0	0	0	1996	2000
	10^{-2}	4051.61	1374.14	0	0	0	0	2000	2000
	10^{-3}	77.81	2.90	0	0	0	1177	2000	2000
	10^{-4}	0.81	0.01	0	0	1732	1998	2000	2000
gcc	6×10^{-2}	465.90	443.45	248.68	361.05	0	0	0	0
	5×10^{-2}	422.86	386.18	37.97	15.88	0	0	5	243
	4×10^{-2}	362.39	307.34	0.33	0	0	0	1913	2000
	3×10^{-2}	280.47	208.05	0	0	0	0	2000	2000
	10^{-2}	64.65	21.91	0	0	0	0	2000	2000
	10^{-3}	0.95	0.04	0	0	979	1919	2000	2000
	10^{-4}	0.01	0	0	0	1985	2000	2000	2000
mcf	6×10^{-2}	446.36	424.95	238.13	345.24	0	0	0	0
	5×10^{-2}	365.20	333.23	32.28	13.48	0	0	35	344
	4×10^{-2}	285.22	241.93	0.23	0	0	0	1940	2000
	3×10^{-2}	199.80	148.44	0	0	0	0	2000	2000
	10^{-2}	37.63	12.80	0	0	0	0	2000	2000
	10^{-3}	0.50	0.02	0	0	1356	1969	2000	2000
	10^{-4}	0.01	0	0	0	1986	2000	2000	2000

Figure 7. L2 Correction Capability

References

[1] S. Nassif, "The light at the end of the cmos tunnel," in *ASAP*, 2010.
[2] S. Rusu *et al.*, "T3: Trends and challenges in vlsi technology scaling towards 100nm," in *ASP-DAC*, 2002.
[3] G. Georgakos *et al.*, "Investigation of increased multi-bit failure rate due to neutron induced seu in advanced embedded srams," in *VLSI*, 2007.
[4] A. Dixit and A. Wood, "The impact of new technology on soft error rates," in *IRPS*, 2011.
[5] E. Ibe *et al.*, "Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule," *ED*, 2010.
[6] T. Suzuki *et al.*, "0.3 to 1.5v embedded sram with device-fluctuation-tolerant access-control and cosmic-ray-immune hidden-ecc scheme," in *ISSCC*, 2005.
[7] R. Naseer and J. Draper, "Parallel double error correcting code design to mitigate multi-bit upsets in srams," in *ESSCIRC*, 2008.
[8] M. Horiguchi and K. Itoh, *Nanoscale Memory Repair*, 2011.
[9] P. Garrou *et al.*, *3D Integration: Technology and Applications*, 2008.
[10] B. Black *et al.*, "Die stacking (3d) microarchitecture," in *MICRO*, 2006.
[11] J. Zhao *et al.*, "Overview of 3d architecture design opportunities and techniques," *DT*, 2017.
[12] B. Prince, *Vertical 3D Memory Technologies* , 2014.
[13] S. Safiruddin *et al.*, "Is 3d integration the way to future dependable computing platforms?" in *OPTIM*, 2012.
[14] M. Lefter *et al.*, "Is tsv-based 3d integration suitable for inter-die memory repair?" in *DATE*, 2013.
[15] H. M. C. Consortium, "Hybrid memory cube specification 2.1," 2014.
[16] M. Lefter *et al.*, "A shared polyhedral cache for 3d wide-i/o multi-core computing platforms," in *ISCAS*, 2015.
[17] JEDEC, "High bandwidth memory," 2016.
[18] D. U. Lee *et al.*, "25.2 a 1.2v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv," in *ISSCC*, 2014.
[19] M. Lefter *et al.*, "Low cost multi-error correction for 3d polyhedral memories," in *NANOARCH*, 2017.
[20] R. Gallager, "Low-density parity-check codes," *TIT*, 1962.
[21] Cadence. (2016) Cadence Encounter RTL Compiler product description.
[22] O. Boncalo *et al.*, "Template-based qc-ldpc decoder architecture generation," in *ICICSP*, 2015.
[23] N. Muralimanohar *et al.*, "CACTI 6.0: A tool to model large caches," HP Laboratories, Tech. Rep., 2009.
[24] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH*, 2011.
[25] J. L. Henning, "SPEC CPU2000: measuring CPU performance in the new millennium," *Computer*, 2000.