

EFFICIENT SELECTIVE MULTI-CHANNEL MEMORY PROTECTION

Renzo Bustamante, Rouhuang Zheng, Michael C. Huang
{rbustama,rzheng3}@ur.rochester.edu and mihuang@ece.rochester.edu
University of Rochester

Abstract—Dynamic Random Access Memory (DRAM) is used as the main form of storage in main memory. DRAM is susceptible to cosmic radiation, alpha particles, voltage variations and aging. All of these factors might cause bit faults, word faults, and even entire chip faults. As more main memory is needed on desktops, servers and supercomputers, the chances of finding faults on DRAM will also increase.

Redundant Array of Independent Memories (RAIM) is an approach to protect DRAM. By adding additional parity channels to the main memory, RAIM is able to protect main memory against the failure of an entire channel, which makes the main memory a lot more robust compared to memory systems with only Error-Correcting Code (ECC) protection.

In this paper, we examine the impact on performance and energy that RAIM schemes have on a system, while also proposing a new type of RAIM implementation that reduces on- and off-chip traffic and energy consumption. Compared to contemporary RAIM implementations, our proposed scheme can reduce traffic and energy consumption by more than 2x.

1. Introduction

Computer System's main memory is made up in its majority by DRAM. DRAM's basic cell, as shown in Figure 1, contains a transistor and a capacitor; the transistor is used as a switch to access the capacitor that stores binary information in the form of an electric charge. Figure 2 shows transistor cells connected in the form of an array. This two dimensional array of several rows and columns creates a bank, several banks create DRAM Chips, and finally DRAM Chips are ganged up together to create a Dual In line Memory Modules (DIMM), which are used in computers, servers, smartphones and supercomputers.

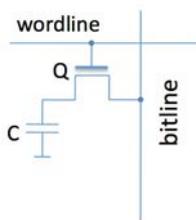


Figure 1. Basic 1T DRAM cell

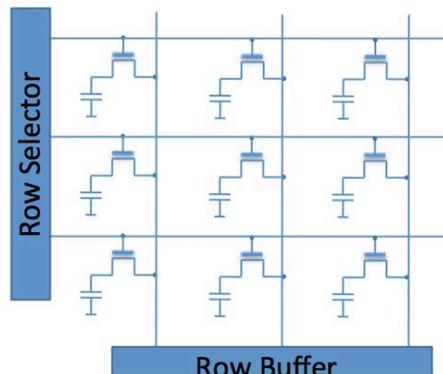


Figure 2. DRAM Chip Bank

Alpha particles, cosmic radiation, voltage variations, process variations and aging have been identified as causes for malfunction of the DRAM chip [1]-[4]. At the same time, studies have shown several other types of faults that can occur within DRAM chips [5][6]. Xin et al. [5] show the types of faults that were found during an evaluation of memory hardware errors of an Internet server farm. The study showed that the type of faults that might occur range from single bit errors in a dram chip to total dram chip failure. Figure 3 shows the findings of that study on DRAM chips; Machine 7 reported column failures, Machine 8 reported whole chip failure, Machine 10 reported one single bit error, and Machine 12 reported a word error.

Vilas et al. [7] performed a study on a supercomputer and presented Failure Rate for the types of error they found. To protect systems against these types of faults found in DRAM chips, protection schemes that allow error detection and error protection have devised; such as SEC-DED [8] and Chipkill [9].

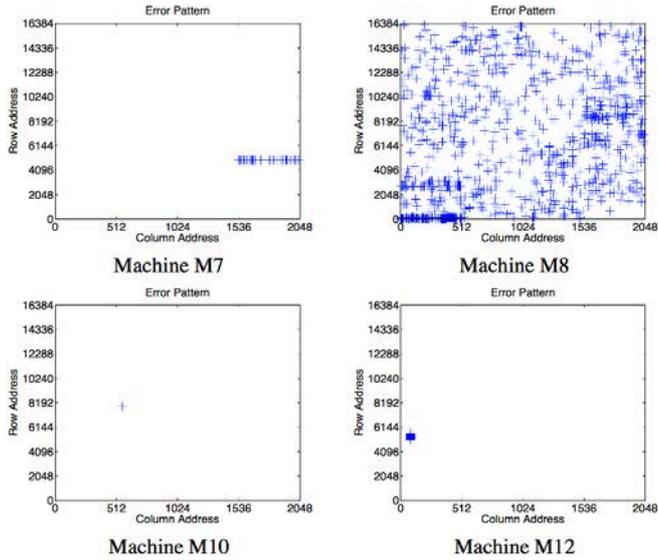


Figure 3. Types of Error found on DRAM Chips. Each cross represents an erroneous cell at its row/column addresses [5].

2. Higher Level Protection Schemes

On top of ECC, higher protection schemes have been devised in order to supply more reliability and availability to computer systems such as memory scrubbing, DIMM sparing and memory mirroring. Figure 4 shows different higher-level protection schemes. Figure 4A shows a regular memory system with four channels, with no data protection where different data blocks A, B, C & D reside in different channels. Figure 4B shows memory mirroring; all the information that is located in channels 0 & 1 is duplicated in channels 2 & 3. Should one channel fail, data can be recovered from the mirrored channel.

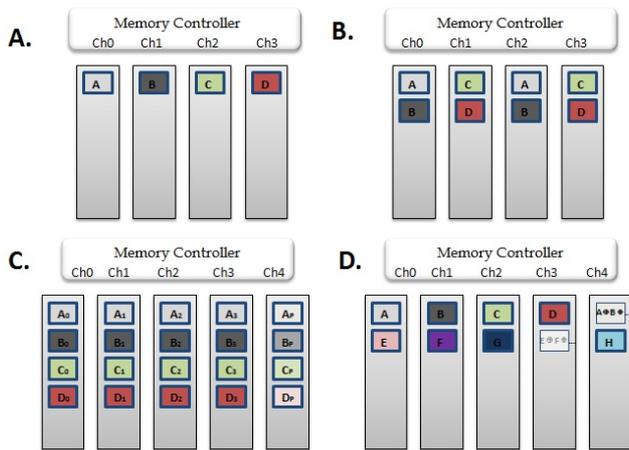


Figure 4. Four types of implementations for higher level protection schemes.

Figure 4C shows how a data block is striped among channels 0-3 and channel 4 is used to store the parity data obtained from

the partitioned data block. Should one channel fail, it can recover by reading the other 3 data channels and the parity channel. Figure 4D shows how data from different data blocks are used to create the parity block, which is not fixed to a specific channel, but changes in order to provide better bandwidth. Should one channel fail it would need to read the data from the other three channels and the parity channel in order to recover.

3. Proposed Implementations

Our implementation has three variants and is based on a RAID 5 like scheme where we have five independent memory channels as show in figure 4D.

The basic idea is that on every memory write request, we also need to issue a write request to the parity channel. To calculate the parity, we have three approaches:

The first one approach, which we will call **raim5a**, is to first check if the data in the parity group are in cache, this is a small optimization since the data in the same parity group is likely to be residing in cache. If a member of the parity group is in cache and is clean we use the data in cache to calculate parity, however if one or more lines in the parity group are not in cache, we fetch the data from memory. This implementation requires up to three read and two write operations if none of the parity group members are in cache, and only two write operations if all the parity group members are present in cache.

The second approach, which we will call **raim5b**, will fetch the data with the address that is going to be written. This data is referred as “old data”, and the data that is going to overwrite the data are referred as “new data”. We also fetch the parity data corresponding to the parity group from the parity channel; this data is referred as “old parity”. Then we XOR the new data with the old data fetched from memory to yield a Delta. This Delta is later XOR-ed to the data read from the parity channel, and written back. The whole write operation always requires 2 reads and 2 writes.

The final third approach is a combination of raim5a and raim5b and it is called **raim5c**. Whenever we have two or more of the cache lines from the data parity group we execute raim5a, if we only have one cache line from the data parity group in cache we execute raim5b. This approach utilizes the best case of raim5a and raim5b. This approach always guarantees the minimum number of extra memory accesses

4. Experimental Setup

We used GEM5 [10], a cycle accurate system simulator coupled with the ruby memory model and DRAMSim2 [11]. We evaluated the PARSEC benchmarks suite [13] using full system simulation. Our simulated machine has 16 cores with 2D mesh interconnect and MESI coherence protocol. The last level cache is 16 banked distributed shared. The size of each bank is 256KB and therefore the whole L2 size is 4MB. We ran 2 billion instructions (excluding kernel instructions) for each scheme.

Table 1. Experimental Setup Parameters

Cores	OoO, 2GHz, 16-core, 2D-mesh interconnect, MESI
Width	8-wide issue and commit
Issue Queue	64-entry unified
LSQ	32-entry load queue and 32-entry store queue
ROB	192-entry
Physical Reg.	128-int, 128-fp
L1 ICache	Private, 32-KB, 4-way, 2-cycle latency
L1 Dcache	Private, 32-KB, 4-way, 2-cycle latency
L2 Cache	Distributed shared, 4-MB, 16-way, 20-cycle latency
Main Memory	5 channels (4-data + 1-parity), 1-GB per channel
DRAM Chip	DDR3-1600
Branch Predictor	Tournament, 2K-entry local, 8K-entry global,

5. Results

Our baseline is a four-channel memory system with no protection scheme whatsoever, **raim5_none** is a five-channel memory system with no protection scheme either, **raim3** is an industry implementation of raim scheme, similar to the one on Figure 4C.

5.1 Performance Impact

Figure 5 shows the impact on IPC for our implementations. On average, none of the implementations have a significant impact on IPC.

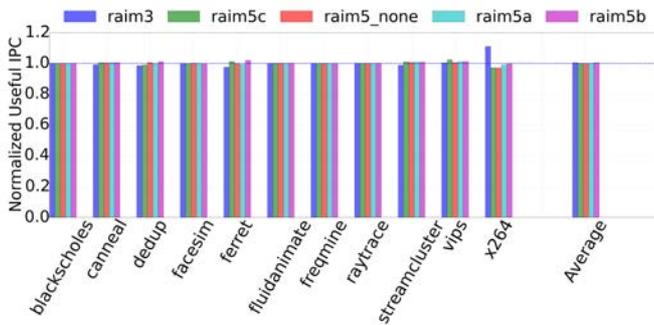


Figure 5. Useful IPC normalized to baseline.

5.2 Traffic impact

Figure 6 shows the impact for on-chip traffic for our implementations. We see that raim5a has a 16% increase in on-chip traffic; this is due to the fact that it is getting data from cache instead of going all the way down to memory for the data parity group. Raim5b shows no increase in on-chip traffic because it goes to memory to obtain the parity group, while raim5c shows a 1% increase in traffic, striking a balance

between going all the way to memory and fetching parity data group from cache.

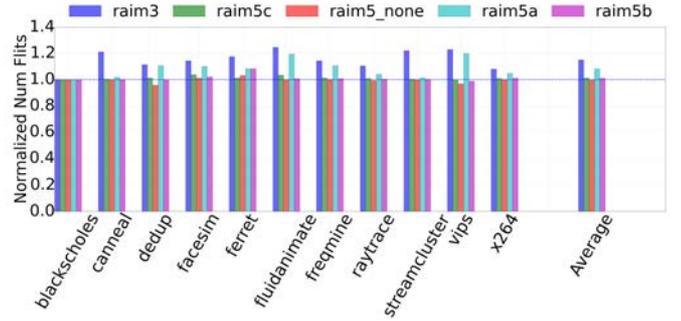


Figure 6. On-chip traffic normalized to baseline. We count the total number of flits

Figure 7 shows the impact of our implementation in off-chip traffic. Raim5a has average increase in traffic of 2.1x, this is due to the fact that whenever a write operation is issued it will perform three reads and one write operation. Raim5b shows an average increase in traffic of 1.6x while Raim5c shows an average increase in traffic of 1.55x. Raim5c doesn't have to go to memory all the time to obtain its data parity group, thus generates less traffic than raim5a.

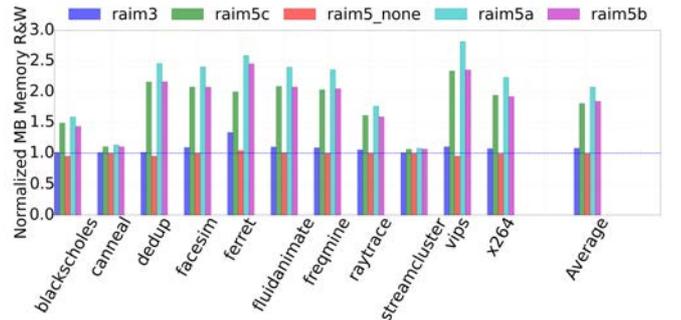


Figure 7. Off-chip traffic normalized to baseline. We count the total amount of data transferred between main memory and cores

5.3 Energy Impact

Figure 8 shows the impact of our implementations on energy consumption on the Memory System. Raim3 consumes an average of 2.8x baseline, this is due to the fact that on a read request 4 channels are activated and on a write request 5 channels are activated. Raim5a shows an increase of 1.7 baseline while our optimized scheme raim5c shows 1.4x increase in energy consumption. Raim5c shows an energy improvement of 2x over raim3. Raim5c also consumes 40% less energy than raim5a.

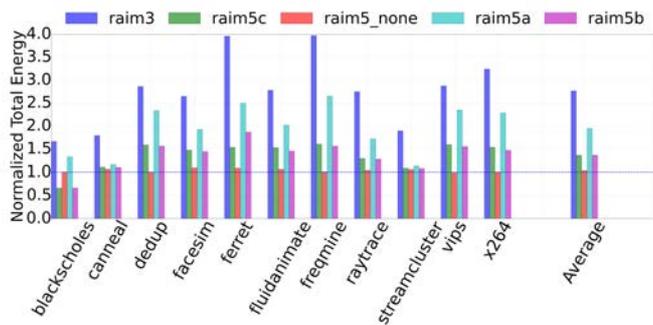


Figure 8. Total memory energy consumption normalized to baseline.

6. Conclusions

- Our optimized raim5c scheme reduced the overhead produced by off-chip traffic in comparison to raim5a.
- Energy reductions of raim5c are significant when compared to industry implementations and also to scheme raim5a.
- The impact in performance of all our schemes is negligible, this is due to the ratio of reads/writes and also that writes are not on the critical path.
- We the implementation of raim5c we can map data that needs the most protection to address ranges that raim5c is implemented on, giving us selective protective memory.

References

[1] J.F Ziegler. Terrestrial cosmic rays. IBM Journal of Research and Development, 40(1):19-39

[2] J.F Ziegler et al. Accelerated testing for cosmic soft-error rate. IBM Journal of Research and Development, 40(1): 51-72, 1996.

[3] J.F Ziegler and W.A. Lanford, "The Effect of Sea Level Cosmic Rays on Electronic Devices", in IEEE International Solid.

[4] T.C May and M.H Woods. Alpha-particle-induced soft errors in dynamic memories. IEEE Trans on Electron Devices, 26(1): 2-9, 1979

[5] X.Lin et at. "A Realistic Evaluation of Memory Hardware Errors and Software Susceptibility" in USENIX, Boston, MA, 2010.

[6] V. Sridharan and D. Liberty "A Study of DRAM Failures in the Field" in High Performance Computing, Networking, Storage and Analysis (SC) 2012 International Conference for, pp. 1-11, 2012.

[7] V. Sridharan et al. "Feng shui of supercomputer memory: Positional effects in dram and sram faults", in Porceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, SC '13, pp. 22:1-22:11,2013.

[8] R.W Hamming. "Error detecting and error correcting codes" BELL SYSTEMS TECHNICAL JOURNAL, vol 29, no. 2, pp. 147-160, 1950.

[9] T.J Dell "A white paper on the benefits of chipkill-correct ecc for pc server main memory" Tech . Rep. 11/19/97, IBM, 1997.

[10] N. Binkert et at. "The gem5 simulator," SIGARCH Comput. Archit. News, vol. 39, no. 2, pp. 1-7, Aug. 2011.

[11] P. Rosenfeld et al. "DRAMSim2: A Cycle accurate Memory Simulator" Computer Architecture Letters, PP(99):1,2011.

[12] C. Bienia et al. "The PARSEC Benchmark Suite: Characterization and Architectural Implications. In PACT, 2008.