

Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding

Jangwoo Kim* Nikos Hardavellas* Ken Mai* Babak Falsafi**† James C. Hoe*

*Computer Architecture Laboratory (CALCM), Carnegie Mellon University

†School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne

<http://www.ece.cmu.edu/~truss>

Abstract

In deep sub-micron ICs, growing amounts of on-die memory and scaling effects make embedded memories increasingly vulnerable to reliability and yield problems. As scaling progresses, soft and hard errors in the memory system will increase and single error events are more likely to cause large-scale multi-bit errors. However, conventional memory protection techniques can neither detect nor correct large-scale multi-bit errors without incurring large performance, area, and power overheads.

We propose two-dimensional (2D) error coding in embedded memories, a scalable multi-bit error protection technique to improve memory reliability and yield. The key innovation is the use of vertical error coding across words that is used only for error correction in combination with conventional per-word horizontal error coding. We evaluate this scheme in the cache hierarchies of two representative chip multiprocessor designs and show that 2D error coding can correct clustered errors up to 32x32 bits with significantly smaller performance, area, and power overheads than conventional techniques.

1. Introduction

With the scaling of process technologies into the nanometer regime, the reliability of embedded memory systems becomes an increasingly important concern for digital system designers. Nanoscale components themselves are increasingly likely to fail, and the growing amount of on-chip memory creates more possible points of failure. As designers integrate more of the memory hierarchy onto the processing die, the number and size of memory arrays on these system-on-a-chip (SoC) and microprocessors will increase [39]. Therefore, errors occurring in the embedded memory systems are a growing threat to the overall SoC and processor reliability and yield.

Errors in memories can be broadly classified as either transient (soft) or permanent (hard) errors. There

are a number of causes of soft errors including energetic particle strikes, signal or power supply noise coupling, and erratic device behavior [3, 15, 53]. In memories, soft errors typically manifest as bit flips in the storage nodes of the memory cell arrays. While the failures-in-time (FIT) rate of an individual SRAM cell remains approximately constant across process technologies, the overall chip FIT rate increases as technologies scale due to the increasing number of SRAM cells on a die [43]. In today's technologies, the majority of soft error events result in only a single cell being disturbed. However, as we scale into the nanometer regime, the single-event multi-bit upset rate exponentially increases because more memory cells fall under the footprint of a single energetic particle strike [29, 34, 41]. The extent of a single-event multi-bit error can range from disrupting a few bits to hundreds of bit flips along entire columns and rows [15, 43]. SRAM designs have already exhibited up to 16 bit corruptions in one dimension [5, 29, 34].

Hard errors can occur either at manufacture-time or in-the-field. Manufacture-time hard errors are due to manufacturing defects such as electrical shorts or opens caused by contaminants during lithography. In memories, these errors are typically repaired at the factory using redundant cell rows, columns, or arrays. The incidence of manufacture-time hard errors is expected to increase dramatically as processes scale to nanometer dimensions overwhelming conventional levels of spare redundancy [9, 33, 40, 44].

In-the-field hard errors are the result of a number of phenomena including electromigration, device wear-out, and highly energetic particle strikes [26, 30, 40, 45]. The frequency of these errors is also expected to rise dramatically, resulting in run-time large-scale information loss (e.g., clusters of cells, rows and columns, or even entire arrays) [13, 40]. This poses a significant threat to the run-time reliability of chips with large embedded memories. Furthermore, increasing inter- and intra-die device variability causes not only an increasing percentage of memory circuits to fail all-together, but also an increasing percentage of

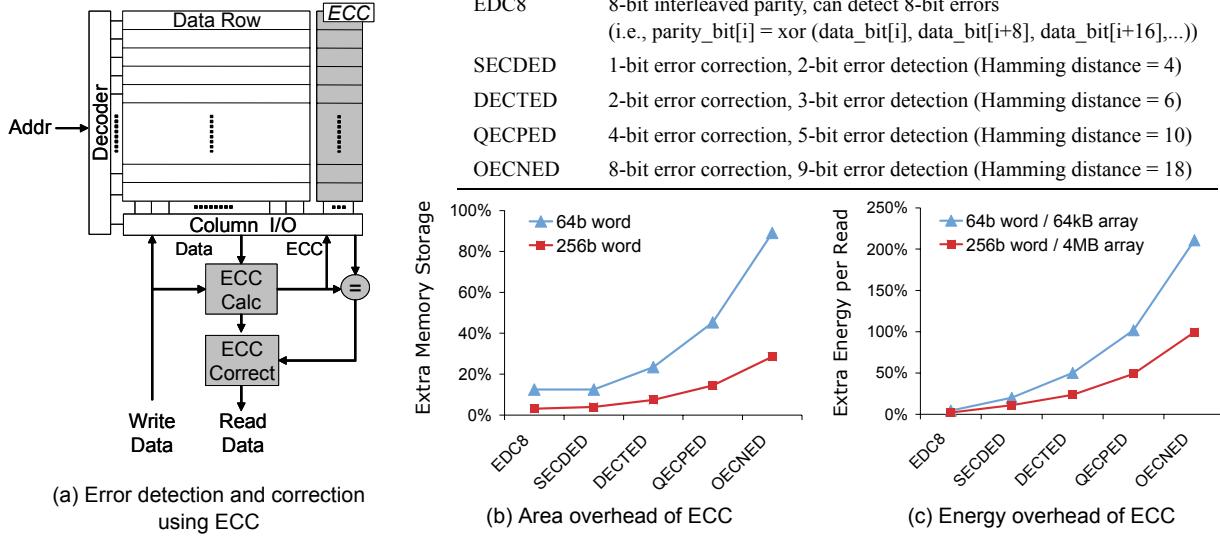


Figure 1. ECC mechanism and overhead. Figure (a) shows the basic ECC mechanism in SRAM. Figures (b) and (c) show the extra area and energy overheads to compute, store, and access ECC bits. We modified Cacti 4.0 [48] to model a standard EDC/ECC logic based on [12].

circuits that are marginally functional and thus more susceptible to hard and soft error phenomena [9, 22].

Modern microprocessors and SoC ASICs already use various protection techniques such as error correcting codes (ECC) [1, 5, 25, 31, 36], bit interleaving [5, 36], and hardware redundancy [8, 24, 33, 35] to safeguard their embedded memories. However, scaling up conventional techniques to cover multi-bit errors will incur large performance, area, and power overheads in part due to the tight coupling between the error detection and correction mechanisms.

Thus, we propose *2D error coding*, an error detection and correction scheme using error coding in two dimensions (horizontal and vertical¹) which decouples the error detection from the error correction mechanisms. The key innovation in this scheme is the combination of conventional per-word horizontal error coding (used only for error detection and possibly small-scale error correction) with vertical error coding across words (used only for error correction). 2D error coding provides **scalable multi-bit error protection** that can reconstruct large clusters of flipped bits, including multiple column/row failures, while incurring significantly **smaller performance, area, and power overheads** when compared to conventional schemes with similar error coverage. This strong multi-bit error protection can also be used to **improve memory manufacturability and yield** by correcting small-scale (e.g., single-bit) manufacture-time hard

errors on-the-fly, while still maintaining soft error protection. We can leverage this resilience gained at the micro-architecture level to aggressively scale SRAM cell sizes while still maintaining robust operation and high yield. We evaluate this scheme in the cache hierarchies of two representative chip multiprocessor (CMP) designs and show that 2D error coding can correct clustered errors up to 32x32 bits with significantly smaller performance, area, and power overheads than conventional techniques.

In the next section, we discuss conventional techniques for protecting memories against errors, deficiencies in those techniques, and how the deficiencies worsen as we scale process technology. In Section 3, we introduce a 2D error coding concept, and Section 4 discusses how to implement the scheme in cache memories. Section 5 explores the overheads of implementing 2D error coding. Section 6 summarizes the related work and Section 7 draws conclusions.

2. Limitations of conventional techniques

This section discusses limitations of conventional memory protection techniques to motivate 2D error coding.

2.1. Error detection and correcting codes

A common technique to protect memory arrays against errors is to use error detecting codes (EDC) and error correcting codes (ECC). Many conventional cache designs use single-error-correct double-error-detect (SECDED) codes [1, 5, 25, 31, 36]. ECC is typi-

1. Throughout the paper, “horizontal” refers to the wordline direction and “vertical” refers to the bitline direction.

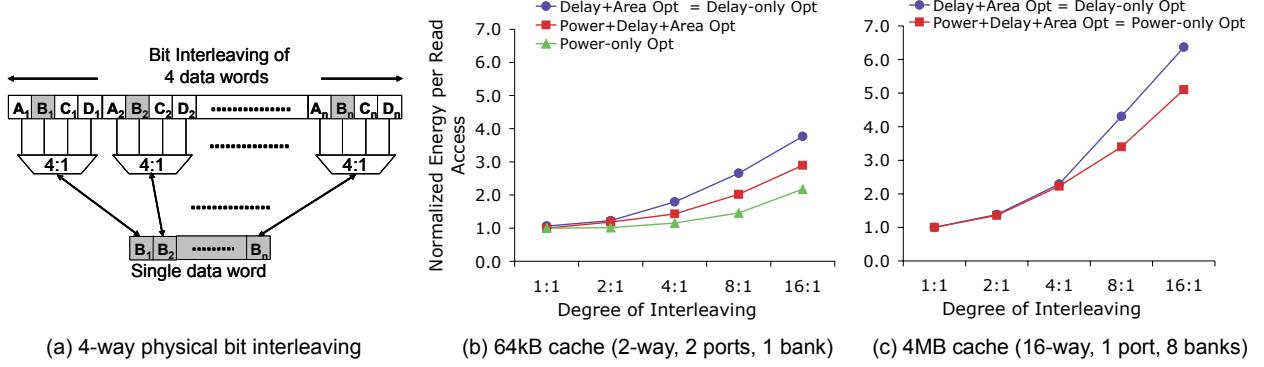


Figure 2. Physical bit interleaving mechanism and overhead. Figure (a) shows the interleaving of four separate ECC codewords. Figures (b) and (c) show the increasing energy consumption as the degree of interleaving increases for (72,64) SECDED ECC words in a 64kB cache and (266,256) SECDED ECC words in a 4MB L2 cache.

cally applied to the data on a per-word basis (e.g., 64 bits) and allows error detection/correction at the cost of extra bits of code storage per word and shared calculation/checking/correction logic (Figure 1(a)). The extra area and energy overhead to implement multi-bit error detection and correction codes grows quickly as the code strength is increased as shown in Figure 1(b) and (c) respectively. In addition, reading and computing the ECC bits for error checking can be a performance bottleneck during read operations. To avoid this latency overhead, some microprocessors use low-overhead EDC (i.e., byte parity) in timing-critical L1 caches, and rely on correction from the next level of the memory hierarchy [25, 31, 36]. However, this technique must duplicate write values in a multi-bit error tolerant L2 cache and incurs significant bandwidth and power overheads, especially in CMP architectures with shared L2 caches. Another method of avoiding the code calculation latency is to periodically sweep through the array, checking the data integrity, but this technique (called scrubbing) has lower error coverage than checking ECC on every read [38].

ECC can also be used for yield enhancement by letting the ECC correct single-bit, manufacture-time hard errors and process-variation errors [2, 44, 46]. While this increases the yield rate of the memory, it sacrifices soft error immunity for the data words that have a pre-existing hard error from manufacturing.

2.2. Physical bit interleaving

Some SRAM arrays use physical bit interleaving (also known as column multiplexing) for higher performance, optimal layout of the cell I/O circuits, and multi-bit error protection. In a bit-interleaved memory, multiple data words are stored in an interleaved fashion along a single physical row of the cell array

(Figure 2(a)). This allows the column decode to be performed in parallel with the row decode. Because a number of columns share cell I/O circuits (i.e., sense amplifier, write driver), their layout is less constrained and does not have to fit in the tiny horizontal pitch of a single cell. In a bit-interleaved memory, SECDED ECC can be used to correct small-scale physically-contiguous multi-bit errors, because these errors affect logically different data words [5, 29, 34, 36].

These advantages come at the cost of additional power due to the unnecessary pseudo-read access of the undesired words in the row (as all cells in a row share a wordline) and additional area and delay due to the long wordline and column mux. These overheads grow significantly as the interleaving factor increases beyond about four, depending on the memory design [4, 20]. To evaluate the overhead of bit interleaving, we modified Cacti 4.0 [48] to model power-efficient bit-interleaved caches in which only sub-arrays containing the selected word are activated. Given the target degree of interleaving, Cacti explores the design space using wordline and bitline segmenting. Cacti optimizes each cache for various objective functions: delay-optimal, power-optimal, delay+area optimal, and delay+area+power balanced. Figure 2(b) and (c) show the dynamic read energy of two cache designs (a 64kB L1 cache and a 4MB L2 cache respectively) as we vary the degree of interleaving. These two cache design points are used later in our architectural studies.

We find that the dynamic power consumption increases significantly with additional degrees of bit interleaving. The increase is more dramatic for the 4MB cache than the 64kB cache, because the wider data word used in the 4MB cache increases the power cost of interleaving and limits possible optimizations. In optimizing for power, Cacti aggressively segments the bitlines to compensate for the increased number of

bitlines driven on each access, which instead increases the area and the delay. When area and delay are also optimized, the steep power consumption increase is unavoidable. While bit interleaving and SECDED ECC can cover small-scale multi-bit errors, large-scale multi-bit errors (e.g., loss of an entire row) are uncorrectable by conventional techniques short of value duplication. Scaling these techniques to cover even modest levels of multi-bit errors (e.g., stronger ECC, higher degrees of bit interleaving) results in large performance, area, and power overheads.

2.3. Hardware redundancy

Current memory designs contain redundant rows, columns, and sub-arrays to tolerate manufacture-time hard errors and thus improve yield. When faulty bits are detected during manufacture-time testing, the faulty addresses are remapped to redundant spare rows, columns, or sub-arrays. If all redundant units are previously allocated, the chip must either be discarded or operate with a reduced memory capacity [8, 31, 33, 35, 36]. Combined with built-in self-test (BIST) or built-in self-repair (BISR) mechanisms [8, 18, 24], the memories can repair hard errors both during manufacturing test and during in-the-field operation.

Conventional hardware redundancy techniques are inefficient because they often use an entire spare row or column to map out only a few (sometimes only one) erroneous bits. To maintain a small area overhead, this technique assumes a low incidence of hard errors. However, the hard error rate in future technologies is expected to increase dramatically and will overwhelm conventional row, column, or sub-array redundancy, resulting in low chip yield rates and high costs [2, 33, 35, 44]. Dramatically increasing the amount of redundancy is infeasible as this would incur large performance and area overheads [2].

2.4. Process-level techniques

In addition to microarchitectural and circuit-level techniques, there are a number of process-level techniques to reduce soft errors such as silicon-on-insulator (SOI) technologies [11], DRAM-like explicit capacitors [19], and specially hardened processes [51]. While these techniques can further reduce the soft error rate, none are “silver bullets” that eliminate soft errors, and they generally do not address hard errors. These techniques are orthogonal to the microarchitectural techniques discussed in this work and can be applied in combination for additional robustness.

3. Two-dimensional error coding

While the frequency and scope of memory errors are increasing, systems still spend the vast majority of their time operating in the absence of errors. Yet in conventional error detection and correction techniques, every memory access must pay the performance, area, and power penalties for the error coverage. For example, if we wish to protect against multi-bit errors using strong multi-bit ECC, every word must store extra bits for the strong ECC codeword, and every access must pay the power and delay penalty of accessing, calculating, and comparing those codewords. Thus, conventional memory protection techniques are not well suited to detect or recover from multi-bit failures and cannot do so without incurring high overheads. If we could decouple the common case error-free operation from the uncommon-case erroneous operation and only pay the overhead of the strong error protection on detection of an error, we could achieve both low overhead and strong error protection.

We propose applying two-dimensional (2D) error coding techniques [47, 52] to the on-die embedded memory system to enable fast common-case error-free operation while still maintaining high error coverage with low VLSI overheads. The key innovation in 2D error coding is the combination of light-weight horizontal per-word error coding with vertical column-wise error coding. The horizontal and vertical coding can either be error detection codes (EDC) or error correcting codes (ECC). The types of codes used in the horizontal and vertical directions allow us to trade-off the error coverage against the VLSI overheads to tailor the design to the types and frequency of errors expected. The vertical codes enable correction of multi-bit errors along rows up to and including entire row failures. Because we use the vertical codes only for error correction, and they are maintained in the background, their impact on error-free operation is minimal.

To illustrate the 2D error coding concept, Figure 3 compares the error coverage and storage overhead of three error protection schemes for a 256x256 bit memory array.¹ Figure 3(a) shows a conventional 4-way bit-interleaved SECDED ECC as might be used in a contemporary memory array. This scheme can cover any error of 4-bits or smaller along a row. Figure 3(b) depicts a scaled version of conventional techniques using an ECC capable of 8-bit correction and 9-bit

1. For the purposes of this illustrative example, we assume that the designer does not have the freedom to alter the memory array aspect ratio. Our more detailed analysis in Section 5 does take this degree of freedom into account.

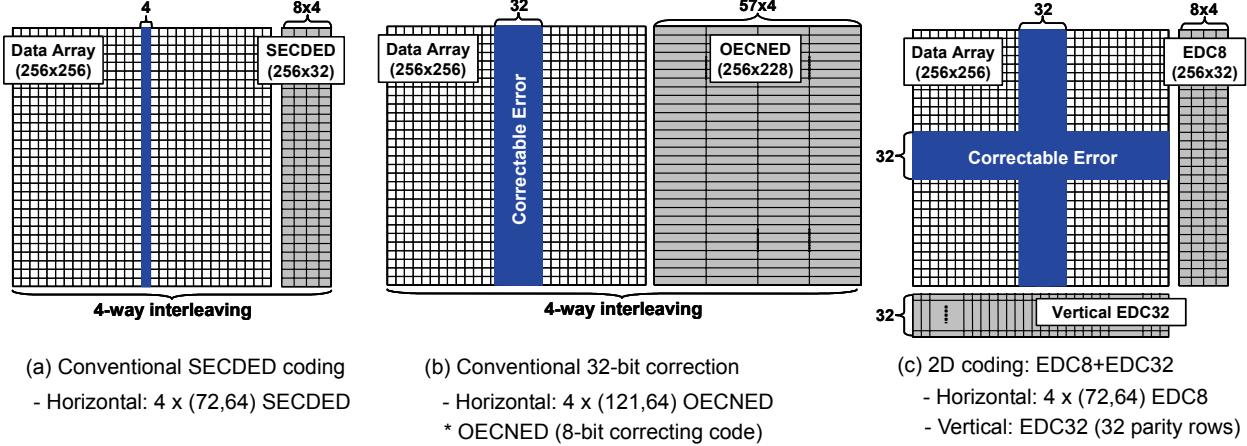


Figure 3. Error coverage and area overhead of conventional coding and 2D coding in a 8kB array. Figure (a) shows the small error coverage of 4-way interleaved SECDED coding (overhead=12.5%). Figure (b) shows the coverage of 4-way interleaved OECNED codes (overhead=89.1%). Figure (c) shows that 2D coding achieves a higher coverage at lower area overhead (overhead=25%).

detection (OECNED), which, in conjunction with 4-way bit interleaving, enables correction of up to 32-bit errors along a row. However, the OECNED ECC code requires significant per-word storage overhead as well as incurring high power and delay costs in calculating and accessing the complex codeword.

Figure 3(c) shows an array using 2D coding with horizontal 4-way interleaved EDC8 and vertical EDC32. This 2D scheme can correct any clustered multi-bit error which does not span more than 32-bits in both the X and Y directions. To detect and correct multi-bit errors, we use a light weight n -bit error detection code, EDC_n , which computes bit-interleaved parity *within* a word. The EDC_n code stores n check bits per word where each check bit stores the parity of every n -th data bit (e.g., for EDC8 parity_bit[i] = data_bit[i] xor data_bit[i+8] xor data_bit[i+16]...) which allows EDC_n to detect all contiguous n -bit errors. EDC8 coding calculation requires the same latency as byte-parity coding (as used in timing-critical L1 caches) and incurs similar power and area overheads as SECDED coding. While this example 2D scheme uses only light-weight error detection codes, the combination of EDCs in the X and Y directions identifies the location of erroneous bits. Logically, their correction in a binary system by simply inverting bits is trivial once the erroneous bits are identified.

Error correction. The combination of the horizontal and vertical codes forms a single strong multi-bit ECC applied to the entire array. The vertical parity rows are interleaved to increase coverage along the columns, with little impact on area, power, or delay. In the example in Figure 3(c), 32-way vertical interleaving, where each vertical parity row stores the parity of every 32nd row, can correct all multi-bit errors that

span up to 32 rows. On the other hand, multi-bit errors spanning over 32 columns can be corrected using the horizontal parity code in the same way that parity rows can correct row failures. This example scheme does not correct multi-bit errors that span over 32 lines in both horizontal and vertical directions. The actual correction algorithm is described in Section 4.

Updating the vertical code. Upon every write to a word in the array, the corresponding bits must first be read out to update the vertical code. The delay, area, and power overheads caused by updating the vertical parity are small because of the modest size of the vertical code array. Furthermore, the vertical update logic can be pipelined in parallel with normal memory operations to hide update latency. The increase in delay and power to perform the “read-before-write” operation is also fairly modest. A number of architectural (e.g., port stealing [27]) and circuit-level (e.g., atomic read-write [16]) techniques can further mitigate this overhead. We evaluate these overheads in Section 5.

Enhancing yield using horizontal SECDED ECC. In-line single-bit error correction can improve memory yield significantly, since it enables single-bit manufacture-time hard errors to go uncorrected via redundancy. For this purpose, 2D error coding can use a horizontal per-word SECDED ECC code to correct common-case single-bit manufacture-time hard errors without utilizing redundancy. This configuration does not significantly affect the overall performance because single-bit corrections bypass the expensive multi-bit error correction sequence. It is important to note that 2D coding maintains multi-bit error protection, even for those words that contain such a single-bit manufacture-time hard error, because the vertical coding still maintains in-the-field hard and soft error

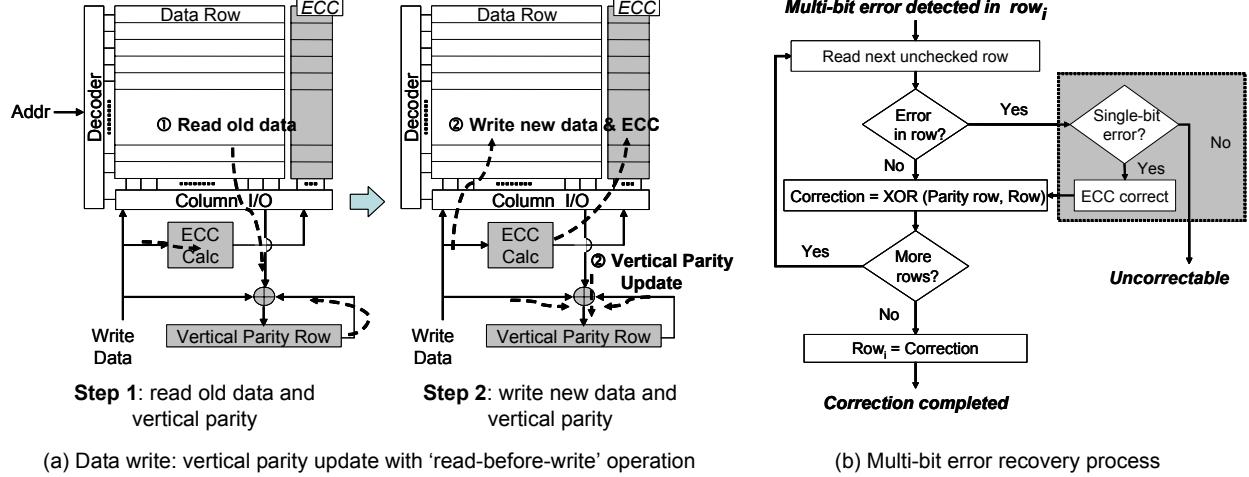


Figure 4. 2D-protected cache architecture. Figure (a) shows the vertical update mechanism illustrated in two steps. Figure (b) shows the vertical multi-bit error correction process embedded as a BIST/BISR algorithm.

protection. SECDED ECC can be extended to increase its multi-bit detection coverage similar to that of interleaved EDC with very low overhead (e.g., SECDED-SBD (single-byte error detection)) [12, 28].

The key design advantages of 2D error coding are:

- **High multi-bit error coverage.** 2D coding can correct any clustered multi-bit error provided the error footprint does not exceed the error coding interleaving factor in both dimensions simultaneously.
- **Low VLSI overheads.** The performance, area, and power overheads of 2D coding are significantly lower than conventional techniques with similar error coverage, as evaluated in Section 5.
- **Improved manufacturability and yield.** The additional micro-architectural resilience of a 2D coding protected memory enhances the manufacturability and yield of memories. This boost in manufacturability can be leveraged to enable aggressive SRAM cell size scaling. The boost yield is quantified in Section 5.

4. Implementing 2D coding for caches

While L2 caches typically employ write-back to conserve off-chip bandwidth and power, L1 caches can use either write-back or write-through policies. Recently, many designs have opted to implement write-through L1 caches to reduce vulnerability to errors and obviate the need for SECDED ECC in L1 [25, 31, 36]. Because dirty data also resides in other cache levels, write-through L1 caches employ only a light-weight EDC and simply invalidate blocks with detected errors. With trends towards higher levels of

chip integration and chip multiprocessors, however, write-through L1 caches are becoming less attractive due to prohibitive demands on bandwidth and power to the lower cache level.

Figure 4(a) depicts the anatomy of a cache data sub-array equipped with 2D error coding and the process for updating the vertical code on writes. Cache tag sub-arrays are handled identically. Because 2D coding is applicable to both ECC and EDC horizontal codes, in this section, we only show a cache model equipped with SECDED ECC horizontal code and interleaved-parity vertical code.

Next, we describe the operation of the coding scheme during error-free operation and in the presence of errors.

Vertical parity update during error-free mode.

In the absence of errors, the horizontal code and the vertical code are updated on every write. Writes originate from the processor (e.g., storing a word into an L1 cache line) or the cache controllers (e.g., dirty evictions). The updates to the horizontal code are based only on the data to be written, unless they are partial writes. However, as depicted in Figure 4(a), the updates to the vertical code require reading the old data word (Step 1) and XORing with the new data word to compute and update the vertical parity (Step 2). Thus, the cache controller converts every write operation to a “read-before-write” operation reading the old data before performing the write of the new data.

The “read-before-write” operation increases the delay and the power consumed by all write operations. The resulting performance impact on the cache manifests as additional port contention. Fortunately, L1 data caches are typically multiported or multi-banked in so called “fat” out-of-order superscalar processors to

accommodate bursty access patterns and provide target latency and bandwidth [1, 31, 36], while L1 data caches in “lean” highly-threaded in-order processors hide the access latencies using thread scheduling [25], thus, much of the time there is spare cache bandwidth [26]. The L2 caches in CMPs are also typically multi-banked [25, 31] to provide high bandwidth for multiple threads. Moreover, write-backs from L1 often account for a small fraction of overall L2 traffic, especially for commercial server workloads.

In Section 5, we present empirical results for two CMP architectures and a wide spectrum of workloads indicating that the performance degradation due to “read-before-write” operations in 2D-protected L1 and L2 caches is modest. We also evaluate a cache port stealing technique based on [27] to reduce the performance degradation in L1 caches of out-of-order processors. Port stealing is a simple scheduling technique that issues the read parts of “read-before-write” operations ahead of the write portion, scheduling the read for idle L1-port cycles. This scheme eliminates most of the port contention problem in L1 caches.

To simplify the update logic and avoid delays in cache port scheduling, the cache controller conservatively issues “read-before-write” requests for both write hits and write misses, even though the old data are used to update the vertical parity row only upon write hits. Moreover, because tag sub-arrays are always read upon a cache access, there is no need for a separate “read-before-write” operation when updating the vertical codes for the tag sub-arrays.

The vertical parity update logic can be pipelined and taken off the cache access critical path and thus has no direct effect on performance. As long as the vertical parity update rate matches the data access rate of the cache, the vertical parity update does not affect the access time or cycle time of the cache sub-array. Updating a register-like vertical parity row is faster than accessing the main array so the rate can be easily matched, in practice.

2D recovery mode. When the horizontal code detects an error that it cannot correct, the controller initiates a 2D recovery process. The controller reads all data rows that share a vertical parity row with the faulty data row and XORs their values together. The result of the XOR operation is the original value of the faulty row, which is then written back to the proper location. If many rows detect a single-bit error in the same bit position, this indicates a large-scale failure along a column (e.g., column failure). The recovery process then examines the vertical code to locate the faulty column, and re-initiates the correction process in the horizontal direction.

The recovery process can be implemented as part of the on-chip BIST/BISR hardware [8, 18, 24]. Figure 4(b) shows an example vertical multi-bit error correction algorithm. The grey area is enabled when the horizontal SECDED ECC code is used. The latency of the 2D correction process is similar to that of a simple BIST march test applied to the data array (i.e., a few hundred or thousand cycles, depending on the number of rows). Because errors are very rare on the order of one every few days [43], the overhead of recovery process does not affect the overall performance.

5. Experimental Results

We evaluate 2D coding with FLEXUS [21], a cycle-accurate, full-system simulator, using two different CMP designs: (1) a “fat” CMP system consisting of four 4-wide superscalar out-of-order cores using dual-port L1 data caches and 16MB of shared L2 cache, and (2) a “lean” CMP system consisting of eight 2-wide superscalar 4-way fine-grain multi-threaded in-order cores using single-port L1 data caches and 4MB of shared L2 cache. The fat CMP balances throughput and single-thread performance, whereas the lean CMP targets only high throughput. Both systems use a cache coherence protocol derived from the Piranha CMP architecture [7]. We list simulated system and workload parameters in Table 1.

We use a mix of commercial and scientific workloads running on Solaris 8. The commercial workloads consist of online transactional processing (OLTP), decision support system (DSS), and web server workloads. For OLTP, we run a TPC-C-like workload, in which multiple clients issue short update transactions against a database. For the DSS workload, we run a TPC-H-like throughput test on a representative subset of scan- and join-bound queries [42], in which multiple concurrent clients issue complex read-only queries against large datasets. Both database workloads run on IBM DB2 v8 Enterprise Server Edition. The web server workload consists of SPECweb99 running on Apache HTTP Server v2.0. We drive the web server using a separate client system and a high-bandwidth link tuned to saturate the server. Finally, we include three parallel shared-memory scientific workloads that exhibit a diverse range of memory access patterns.

In our simulations, we use the SimFlex statistical sampling methodology [50]. Our samples are drawn over an interval of 10 to 30 seconds of simulated execution time for OLTP, DSS, and web workloads and a single iteration for scientific workloads. We target 95% confidence intervals with $\pm 5\%$ error using

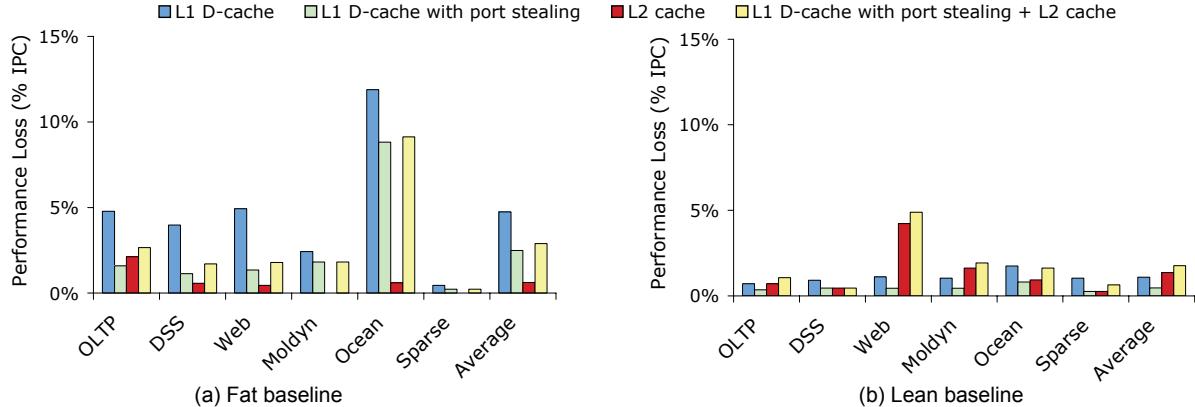


Figure 5. Performance (IPC) loss in 2D-protected caches. The fat baseline experiences a larger performance loss due to L1 cache port contention. Port stealing effectively reduces the contention.

matched-pair comparison for relative performance measurements. We launch measurements from checkpoints with warmed caches and branch predictors, then run for 100,000 cycles to warm the pipeline and queue states prior to 50,000 cycles of cycle-accurate measurement. We assess performance by measuring the aggregate user instructions committed per cycle (IPC), which is proportional to the system throughput [50].

We use Cacti 4.0 [48] to model the power, delay, and area overhead of caches used in this study. We model the shared L2 cache as multiple cache banks residing on each corner of the die with a core-to-L2 crossbar in the center. The latencies of L2 caches are measured taking into consideration the average core-to-L2 crossbar latency, wire delay, and cache bank access latency. The area, delay, and power models assume a 70nm process technology.

5.1. 2D coding overheads

We implement 2D error coding for L1 data caches and L2 shared caches in the fat and lean baseline CMP

systems. We apply EDC8 and EDC16 horizontal codes to 64-bit words in L1 caches and a 256-bit word in L2 caches, respectively. We implement 4-way bit-interleaved L1 caches and 2-way bit-interleaved L2 caches to allow the horizontal codes to detect 32-bit errors along a row. We also apply an EDC32 vertical code by adding 32 parity rows per cache bank. This configuration can detect and correct 32x32-bit clustered errors.

The right-most bars in Figure 5 show the performance overheads of the two baseline systems where both the L1 and L2 caches are protected by the 2D error coding described. The fat CMP experiences a 2.9% average performance loss whereas the lean CMP experiences a 1.8% average performance loss, showing that both baseline systems can tolerate the cache latency and port occupancy introduced by 2D coding. The operations directly affected by 2D coding are write requests, dirty evictions, and L1-to-L1 transfers of dirty data, most of which update the cache off the critical path and thus have a low impact on performance. 2D coding indirectly affects cache performance by increasing occupancy and potentially delaying other

Table 1. Simulated systems and workload parameters

	Fat CMP	Lean CMP	Commercial Workloads	
Cores	Four OoO cores 4GHz, UltraSparc ISA 4-w superscalar, 64-entry store queue	Eight in-order cores 4GHz, UltraSparc ISA 2-w superscalar, 4-threads, 64-entry store queue	DB2 OLTP	100 warehouses (10GB), 64 clients, 2GB buffer pool
L1 cache	64kB split I/D, 2-way assoc, 64B lines, 2-cycle hit, write-back, 2-port D-cache	64kB split I/D, 2-way assoc, 64B lines, 2-cycle hit, write-back, 1-port D-cache	DB2 DSS	Queries 1,6,13,16, 1GB dataset, 16 clients, 480MB buffer pool
L2 cache	16MB unified, 8-way assoc, 64B lines, 1 port, 16-cycle hit, 1-cycle crossbar latency, write-back, 64 MSHRs	4MB unified, 16-way assoc, 64B lines, 1 port, 12-cycle hit, 1-cycle crossbar latency, write-back, 64 MSHRs	Apache Web	16K connections, fastCGI, worker thread model
Memory	3GB, 60ns access latency, 64 banks		Scientific Workloads	
			Moldyn	19,652 molecules, boxsize 17, 2.56M max iterations
			Ocean	258x258 grid, 9600s relaxation, 29K res., error tolerance 1e-7
			Sparse	4096x4096 matrix

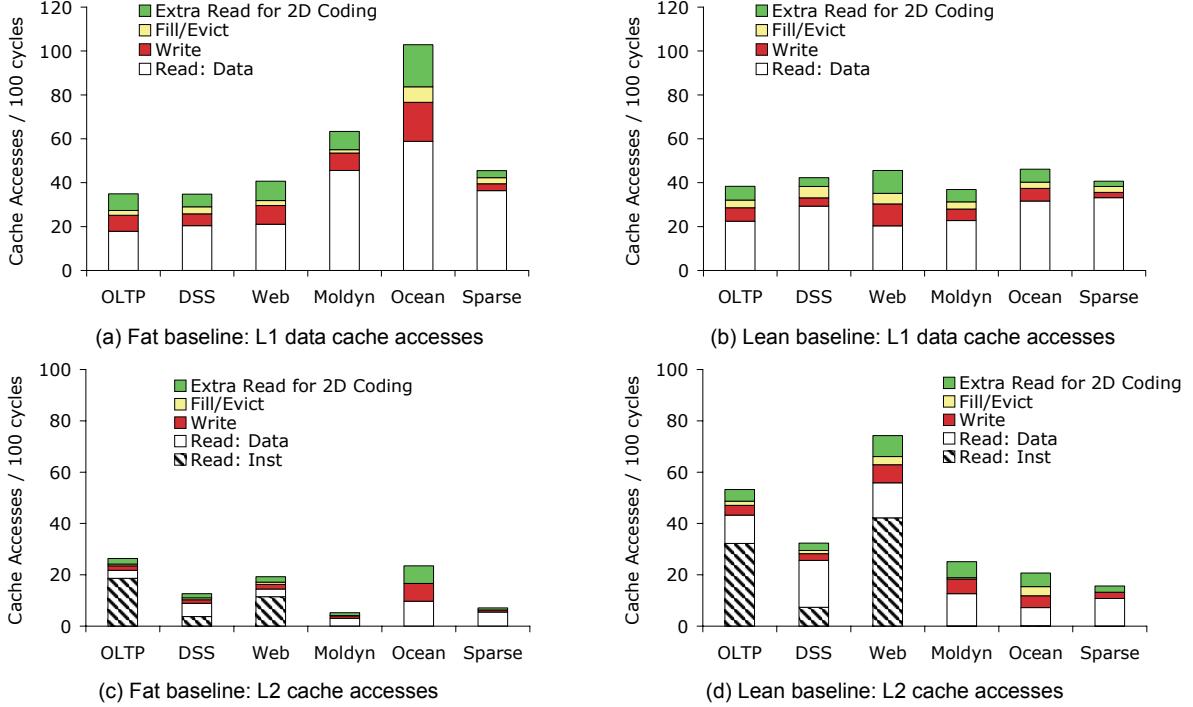
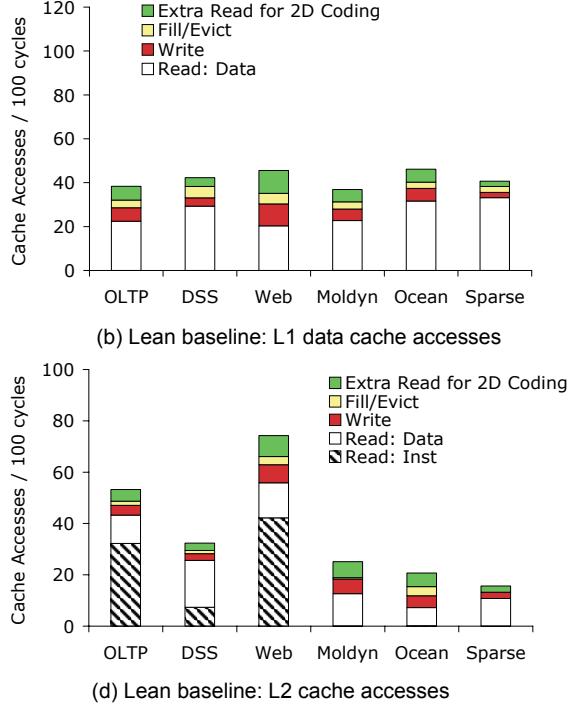


Figure 6. Cache access breakdown for each baseline system during 100 CPU cycles. The distribution shows that writes, that issue ‘read-before-write’ operations, take only a small fraction of overall cache accesses.

cache requests. However, the out-of-order execution of the fat CMP and the multiple hardware threads of the lean CMP effectively hide the additional delay.

The first two bars in Figure 5 represent the performance degradation when protecting only L1 data caches with 2D coding whether port stealing is applied or not. For the fat CMP, port stealing reduces approximately 72% and 34% of port contention for commercial workloads and memory-intensive scientific workloads, respectively. For the lean CMP, we find port stealing has a smaller impact because port contention in L1 caches is already minimal. The third and fourth bars in Figure 5 show the performance degradation when protecting only the L2 cache or both the L1 (with port stealing) and L2 caches, respectively.

Figure 6 shows the average distribution of cache accesses over 100 processor cycles. Both the L1 data caches and L2 shared caches in the two systems execute approximately 20% more cache requests due to the extra reads imposed by 2D coding. The fat CMP, consisting of four out-of-order cores, consumes higher L1 cache bandwidth per core, whereas the lean CMP, consisting of eight in-order cores, consumes higher L2 cache bandwidth. The different bandwidth usage explains the fat CMP’s larger performance loss for L1 cache protection and the lean system’s larger performance loss for L2 cache protection. The web server workload in the lean CMP exhibits a 4% performance



loss due to high bank contention in the L2 cache caused by the increased workload throughput.

Figure 7 compares the overhead of code storage, coding latency, and dynamic power consumption of 2D coding with that of conventional methods in protecting a 64kB L1 data cache and a 4MB L2 cache. We evaluate the overhead of two baseline 2D coding methods; 4-way interleaved horizontal EDC8 with vertical EDC32 (*EDC8+Intv4, EDC32*) and 2-way bit-interleaved horizontal EDC16 with vertical EDC32 (*EDC16+Intv2, EDC32*). To achieve the same 32-bit error coverage, conventional methods combine ECC coding and physical bit interleaving. Therefore, DECTED, QECPED, and OECNED codes are combined with 16-way (*DECTED+Intv16*), 8-way (*QECPED+Intv8*), and 4-way physical bit interleaving (*OECNED+Intv4*), respectively. We measure the dynamic power consumption assuming that each cache receives 20% more read requests due to 2D coding (Figure 6) and that read and write operations take the same amount of dynamic energy. We estimate the code size of multi-bit ECC codes based on Hamming distance [28]. Coding latency is estimated as the depth of syndrome generation and comparison circuit that consists of an XOR tree and an OR tree. We assume that there is an XOR tree dedicated to each check bit to compute all check bits within a single word in parallel. We apply EDC/ECC coding to 48-bit, 64-bit, and 256-

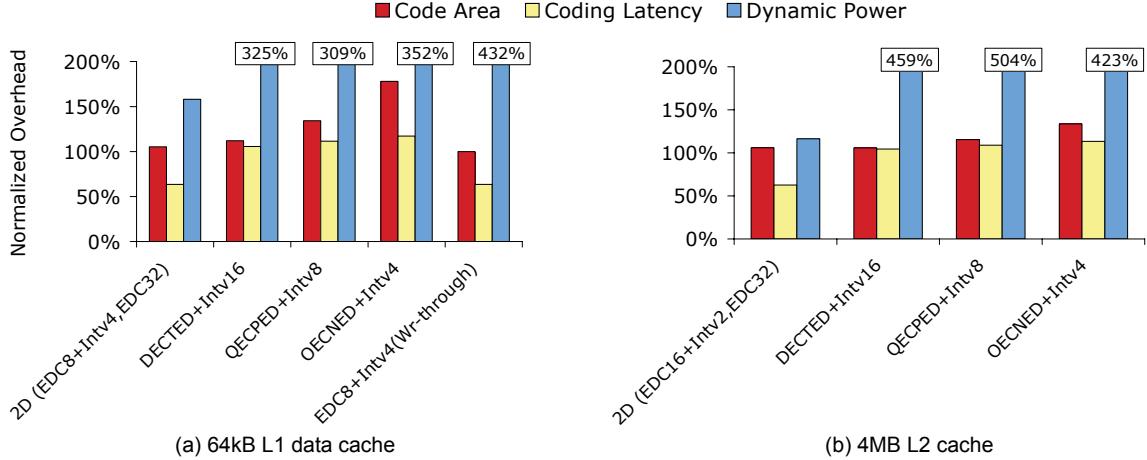


Figure 7. Area, delay and power overhead of various coding schemes for 32x32-bit coverage, normalized to SECDED protection with 2-way physical bit interleaving. 2D coding incurs significantly lower overheads.

bit words for the tag arrays, the 64kB L1 data array, and the 4MB L2 data array, respectively. All these overheads are normalized to the overhead of conventional SECDED protection with 2-way physical bit interleaving.

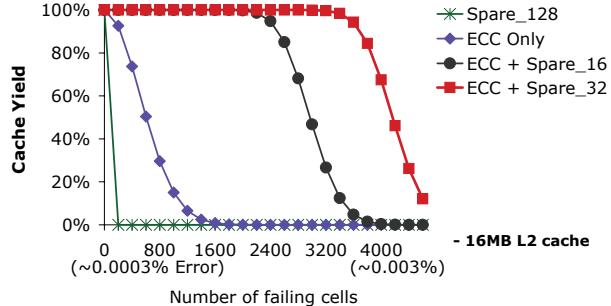
Conventional methods consume a significant amount of power due to complex coding logic, extra check bit access, and bit interleaving. Using relatively simpler multi-bit ECC codes (e.g., DECTED) necessitates increasing the degree of power-hungry bit interleaving to meet the coverage target, whereas using stronger codes (e.g., OECNED) can alleviate the power overhead of bit interleaving, but increases the power consumed for coding and accessing the check bits. On the other hand, 2D coding only incurs a minimal amount of power due to the simple horizontal and vertical codes. 2D coding’s extra power consumption in Figure 7(a) is mainly due to the 4-way bit interleaving required to detect 32-bit errors using EDC8 codes.

2D coding also need less area to store check bits than conventional techniques, because the horizontal code requires fewer check bits per word and the vertical code requires only 32 parity rows per bank. The extra area overhead of 2D coding compared to the baseline SECDED protection is only 5% and 6% for the L1 and L2 caches, respectively. The right-most bar in Figure 7(a) shows that L1 caches using a write-through policy can duplicate values in the L2 cache to avoid the 2x area overhead at the cost of significant power and bandwidth increase in the L2 cache. Because 2D coding’s horizontal coding latency is close to the conventional byte-parity checking latency, the L1 cache can employ a write-back policy without increasing the cache access delay. This scheme has a significant power and performance advantage over protecting L1 caches using a write-through policy.

5.2. Manufacturability and yield improvement

2D error coding’s ability to correct multi-bit errors can also increase the manufacturability and yield of memories. As the areas of SoCs and microprocessors are increasingly dominated by the embedded memory system, overall chip yield will closely track the on-die memory yield [39]. As discussed in Section 2.3, scaling conventional horizontal per-word ECC to cover multi-bit errors incurs considerable overheads, and thus memories can only practically employ an error detecting code or SECDED ECC. In a 2D-protected memory, the horizontal code can be used to correct small-scale (e.g., single-bit) manufacture-time hard errors without using redundant elements, while maintaining run-time error immunity. A small degree of interleaving increases the hard error correction coverage further. Because most manufacture-time hard errors are single-bit errors, this greatly decreases the spare redundancy requirements [2, 44, 46].

Figure 8 shows the expected yield and reliability when ECC corrects single-bit hard errors. Figure 8(a) estimates the expected yield for a 16MB L2 cache as we vary the faulty-bit hard error rate (HER). We use a yield model similar to Stapper *et al.* [46] which assumes a random distribution of hard faults throughout the memory array. We then calculate the probability of a single data word containing a multi-bit error, and if more data words have multi-bit errors than redundant rows, the memory is considered faulty. The yield of a cache equipped with only spare rows falls quickly with the error rate due to the high number of words with one or more faults. The yield with ECC alone has also poor yield because it cannot correct any word with a multi-bit error. When we combine ECC and a small number of spare rows, the yield improves



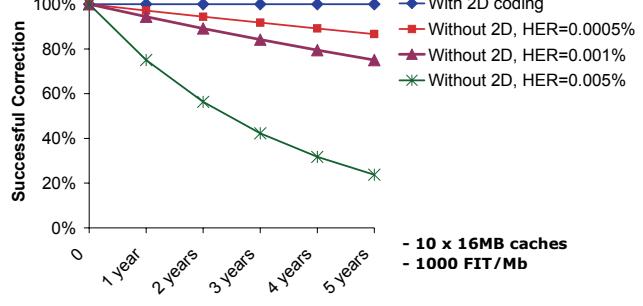
(a) Yield of 16MB L2 cache using ECC-based hard error correction

Figure 8. L2 cache yield and reliability when ECC corrects hard errors. (a) 2D protection using the horizontal SECDED ECC greatly reduces the amount of spare lines, thus improving the yield. (b) 2D coding’s multi-bit error correction capability maintains in-the-field hard and soft error immunity.

dramatically because the redundancy is best suited for correcting words with multi-bit errors and SECDED ECC is best suited for correcting words with only a single-bit error.

Figure 8(b) estimates the expected reliability for a system consisting of ten 16MB caches when ECC corrects single-bit hard errors with or without 2D coding. Assuming a soft error rate of 1000FIT/Mb [43], we measure the probability of all soft errors in a given period that occur only in non-faulty data words. We vary the faulty-bit hard error rate from 0.0005% to 0.005%. When a single-bit soft error occurs in a faulty cache block, it is combined with a faulty bit to create a multi-bit error within a single data word which SECDED ECC cannot correct. The graph shows that as the system continues to operate, there is a significant probability of soft errors occurring in faulty cache blocks, even at low hard error rates. Therefore, ECC should not be used to correct hard errors unless the memory has a multi-bit error correction capability. On the other hand, 2D coding always maintains run-time error immunity, even at the presence of hard errors.

We can leverage this robustness gained at the micro-architecture level to aggressively scale SRAM cell sizes while still maintaining robust operation and high yield. Since 2D coding can correct any single-bit error in a word in-line with the memory operation, this obviates the need for using expensive redundant rows/columns/arrays to correct these single-bit errors. Thus, in a memory with 2D coding, we can scale the cell down further than in designs without robust micro-architectural error correction, since even after redundancy assignment, the cell array does not need to be perfect. In-line error correction afforded by 2D error coding transparently corrects any single-bit error in a word, and yet the memory retains runtime error tolerance against additional soft and hard errors.



(b) Reliability of ECC-based hard error correction

6. Related work

Some well-known SECDED ECC codes, such as HV parity [52] and 2D-parity product codes [10, 17], employ horizontal and vertical parity bits to detect and correct single-bit errors within a word. To detect an error, these schemes must read both horizontal and vertical parity bits. Tanner [47] applies product codes to entire RAM arrays to reduce the code area. This scheme detects and corrects single-bit errors within a memory array. Mohr [32] also applies product codes to memory arrays and uses horizontal byte-parity codes to enable low-latency error detection. The proposed 2D coding in this paper similarly achieves the area-efficient protection by applying only a single strong error correcting code to the entire array. However, our scheme differs from these codes by (1) enabling large-scale multi-bit error detection and correction, (2) separating the high-overhead multi-bit error correcting logic from low-overhead error detection logic, and (3) maintaining the vertical codes off the critical path of normal operations.

Many microprocessors use low-overhead EDC codes (i.e., byte parity) in L1 caches and duplicate the data in lower level caches using a write-through policy [25, 31, 36]. However, this technique assumes a multi-bit error tolerant L2 cache and incurs significant bandwidth and power overheads in the L2 cache (especially in CMPs with shared L2 caches). To alleviate the costly writes to L2 caches, Zhang et al. [54] propose using a small, fully-associative structure that holds duplicate copies of recently-written blocks. When evicting from the small cache, the data is written back to the multi-bit error tolerant L2 cache. This scheme provides full protection to the L1 cache. However, duplications in a small cache can still increase the performance loss and power overhead by several

factors if contention in the small cache causes frequent writes to the L2 cache [37].

Some L1 cache designs employ early writebacks of dirty blocks to reduce the vulnerability of cache blocks to soft errors using scrubbing, dead-block intervals, and fixed-interval delays [6, 14, 49]. These schemes can reduce the vulnerability of L1 cache blocks against multi-bit soft errors. However, these L1 cache protection techniques protect only portions of the L1 cache and can incur significant performance degradation due to increased miss rates in the L1 cache.

Sadler and Sorin [37] separate error detection from correction in L1 data caches by integrating fast error-detection codes with the data array and maintaining a separate array for holding error correcting codes. The chosen codes detect and correct multi-bit faults. However, this scheme allocates a large area to store a multi-bit ECC for every cache word. To reduce the area overhead, Kim and Somani [23] protect only the most error-prone cache blocks based on the access frequency. However, this scheme neither provides full protection nor considers large-scale multi-bit errors.

The proposed 2D error coding in this paper provides full protection against large-scale multi-bit errors. This mechanism can be equally applicable to all levels of the cache hierarchy, while maintaining only small performance, area, and power overheads. 2D coding can be configured for the low-latency operation required for timing-critical arrays or yield-enhancement without sacrificing run-time error immunity.

7. Conclusion

Just as power has emerged in recent years as a primary constraint in digital system design, as we scale down to the nanometer regime, reliability and manufacturability will emerge as first-class design constraints. Given the large amount of memory on modern microprocessors and SoCs, memory reliability and manufacturability will be of paramount importance. Conventional error protection techniques will be unable to cope with the frequency and scope of error events in future technology generations. Our 2D error coding scheme provides scalable multi-bit error protection to enhance memory reliability and manufacturability, making it a promising reliability technique for future memory designs.

Acknowledgements

We thank the SimFlex and TRUSS research groups at Carnegie Mellon. We also thank Jaume Abella, Xavier Vera, and Antonio González at Intel

Barcelona Research Center for their helpful feedback. This work is supported by NSF award ACI-0325802, NSF CAREER award CCF-0347568, Sloan fellowship, the Center for Circuit and System Solutions, FCRP, Cylab, and by grants and equipment from Intel.

References

- [1] Advanced Micro Devices. AMD eighth-generation processor architecture. *AMD white paper*, Oct 2001.
- [2] A. Agarwal, et al. Process variation in embedded memories: Failure analysis and variation aware architecture. *IEEE Journal of Solid-State Circuits*, Sep 2005.
- [3] M. Agostinelli, et al. Erratic fluctuations of SRAM cache Vmin at the 90nm process technology node. In *International Electron Devices Meeting*, Dec 2005.
- [4] B. S. Amrutur and M. A. Horowitz. Speed and power scaling of SRAM's. *IEEE Transactions on Solid-State Circuits*, 35(2):175–185, Feb 2000.
- [5] H. Ando, et al. Accelerated testing of a 90nm SPARC64V microprocessor for neutron SER. In *The Third Workshop on System Effects on Logic Soft Errors*, 2007.
- [6] G. Asadi, et al. Balancing performance and reliability in the memory hierarchy. In *International Symposium on Performance Analysis of Systems and Software*, Mar 2005.
- [7] L. Barroso, et al. Piranha: A scalable architecture base on single-chip multiprocessing. In *International Symposium on Computer Architecture*, Jun 2000.
- [8] D. K. Bhavsar. An algorithm for row-column self-repair of RAMs and its implementation in the Alpha 21264. In *International Test Conference*, pp. 311–318, Sep 1999.
- [9] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–17, Nov-Dec 2005.
- [10] P. Calingaert. Two-dimensional parity checking. *Journal of the ACM*, 8(2):186–200, Jan/Mar 1961.
- [11] E. H. Cannon, et al. SRAM SER in 90, 130 and 180 nm bulk and SOI technologies. In *International Reliability Physics Symposium Proceedings*, pp. 300–304, Apr 2004.
- [12] C. L. Chen and M. Y. Hsiao. Error-correcting codes for semiconductor memory applications: A state-of-the-art review. *IBM Journal of Research and Development*, 28(2):124–134, Mar 1984.
- [13] K. Chkraborty and P. Mazumder. *Fault-Tolerance and Reliability Techniques for High-Density Random-Access Memories*. Prentice Hall PTR, 2002.
- [14] V. Degalahal, et al. Soft errors issues in low-power caches. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(10):1157–1165, Oct 2005.
- [15] P. E. Dodd, et al. Neutron-induced soft errors, latchup, and comparison of SER test methods for SRAM technologies. In *International Electron Devices Meeting*, Dec 2002.
- [16] J. Dorsey, et al. An integrated quad-core Opteron processor. In *International Solid-State Circuits Conference*,

- 2007.
- [17] P. Elias. Error-free coding. *IRE Transactions on Information Theory*, PGIT-4:29–37, Sep 1954.
 - [18] M. Franklin and K. K. Saluja. Built-in self-testing of random-access memories. *IEEE Computer*, Oct 1990.
 - [19] Y. Fujii, et al. Soft error free, low power and low cost superSRAM with 0.98um² cell by utilizing existing 0.15um DRAM process. In *Digest of Technical Papers, Symposium on VLSI Technology*, Jun 2004.
 - [20] K. Ghose and M. B. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation. In *International Symposium on Low Power Electronics and Design*, Aug 1999.
 - [21] N. Hardavellas, et al. Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. *SIGMETRICS Performance Evaluation Review*, 31(4):31–35, Apr 2004.
 - [22] R. Heald and P. Wang. Variability in sub-100nm SRAM designs. In *International Conference on Computer-Aided Design*, Nov 2004.
 - [23] S. Kim and A. Somani. Area efficient architectures for information integrity checking in cache memories. In *International Symposium on Computer Architecture*, May 1999.
 - [24] I. Kim, et al. Built in self repair for embedded high density SRAM. In *International Test Conference*, 1998.
 - [25] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded Sparc processor. *IEEE Micro*, 25(2):21–29, Mar-Apr 2005.
 - [26] S. J. Krumbein. Metallic electromigration phenomena. *IEEE Transactions on Components Hybrids, and Manufacturing Technology*, 11(1):5–15, Mar 1998.
 - [27] K. M. Lepak and M. H. Lipasti. Silent stores for free. In *International Symposium on Microarchitecture*, Dec 2000.
 - [28] S. Lin and D. J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice Hall, 1983.
 - [29] J. Maiz, et al. Characterization of multi-bit soft error events in advanced SRAMs. In *International Electron Devices Meeting*, Dec 2003.
 - [30] J. G. Massey. NBTI: What we know and what we need to know: A tutorial addressing the current understanding and challenges for the future. In *International Integrated Reliability Workshop Final Report*, pp. 199–201, Oct 2004.
 - [31] J. Mitchell, D. Henderson, and G. Ahrens. IBM POWER5 processor-based servers: A highly available design for business-critical applications. *IBM White paper*, Oct 2005.
 - [32] K. C. Mohr and L. T. Clark. Delay and area efficient first-level cache soft error detection and correction. In *International Conference on Computer Design*, Oct 2006.
 - [33] M. Nicolaidis, N. Achouri, and S. Boutobza. Dynamic data-bit memory built-in self-repair. In *International Conference on Computer Aided Design*, Nov 2003.
 - [34] K. Osada, K. Yamaguchi, and Y. Saitoh. SRAM immunity to cosmic-ray-induced multierrors based on analysis of an induced parasitic bipolar effect. *IEEE Journal of Solid-State Circuits*, 39(5):827–833, May 2004.
 - [35] S. Ozdemir, et al. Yield-aware cache architectures. In *International Symposium on Microarchitecture*, Dec 2006.
 - [36] N. Quach. High availability and reliability in the Itanium processor. *IEEE Micro*, 20(5):61–69, Sep-Oct 2000.
 - [37] N. N. Sadler and D. J. Sorin. Choosing an error protection scheme for a microprocessor’s L1 data cache. In *International Conference on Computer Design*, Oct 2006.
 - [38] A. M. Saleh, J. J. Serrano, and J. H. Patel. Reliability of scrubbing recovery-techniques for memory systems. *IEEE Transactions on Reliability*, Apr 1990.
 - [39] U. Schlichtmann. Tomorrows high-quality SoCs require high-quality embedded memories today. In *International Symposium on Quality Electronic Design*, Mar 2002.
 - [40] J. Segura and C. F. Hawkins. *CMOS Electronics: How It Works, How It Fails*. Wiley-Interscience, 2004.
 - [41] N. Seifert, V. Zia, and B. Gill. Assessing the impact of scaling on the efficacy of spatial redundancy based mitigation schemes for terrestrial applications. In *The Third Workshop on System Effects on Logic Soft Errors*, 2007.
 - [42] M. Shao, A. Ailamaki, and B. Falsafi. DBmbench: fast and accurate database workload representation on modern microarchitecture. In *International Conference on Computer Science and Software Engineering*, 2005.
 - [43] C. W. Slayman. Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations. *IEEE Transactions on Device and Materials Reliability*, 5(3), Sep 2005.
 - [44] M. Spica and T. Mak. Do we need anything more than single bit error correction ECC? In *International Workshop on Memory Technology, Design and Testing*, 2004.
 - [45] J. Srinivasan, et al. The impact of technology scaling on lifetime reliability. In *International Conference on Dependable Systems and Networks*, Jun 2004.
 - [46] C. H. Stapper and H. Lee. Synergistic fault-tolerance for memory chips. *IEEE Transactions on Computers*, 41(9):1078–1087, Sep 1992.
 - [47] R. M. Tanner. Fault-tolerant 256k memory designs. *IEEE Transactions on Computers*, Apr 1984.
 - [48] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. Cacti 4.0. *HP Technical Report*, Jun 2006.
 - [49] X. Vera, et al. Reducing soft error vulnerability of data caches. In *The 3rd Workshop on Silicon Errors in Logic-System Effects*, 2007.
 - [50] T. F. Wenisch, et al. Simflex: Statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, Jul-Aug 2006.
 - [51] Y. Z. Xu, et al. Process impact on SRAM Alpha-particle SEU performance. In *International Reliability Physics Symposium*, Apr 2004.
 - [52] J. Yamada, et al. A submicron 1 Mbit dynamic RAM with a 4-bit-at-a-time built-in ECC circuit. *IEEE Journal of Solid-State Circuits*, 19(5):627–633, Oct 1984.
 - [53] J. F. Zeigler, et al. Accelerated testing for cosmic soft-error rate. *IBM Journal of Research and Development*, 40(1):19–39, Jan 1996.
 - [54] W. Zhang. Enhancing data cache reliability by the addition of a small fully-associative replication cache. In *International Conference on Supercomputing*, June 2004.