Resilient Die-stacked DRAM Caches

Jaewoong Sim* Gabriel H. Loh† Vilas Sridharan‡ Mike O'Connor†

*School of ECE
Georgia Institute of Technology
jaewoong.sim@gatech.edu

†AMD Research ‡RAS Architecture
Advanced Micro Devices, Inc.
{gabe.loh,vilas.sridharan,mike.oconnor}@amd.com

ABSTRACT

Die-stacked DRAM can provide large amounts of in-package, high-bandwidth cache storage. For server and high-performance computing markets, however, such DRAM caches must also provide sufficient support for reliability and fault tolerance. While conventional off-chip memory provides ECC support by adding one or more extra chips, this may not be practical in a 3D stack. In this paper, we present a DRAM cache organization that uses error-correcting codes (ECCs), strong checksums (CRCs), and dirty data duplication to detect *and* correct a wide range of stacked DRAM failures, from traditional bit errors to large-scale row, column, bank, and channel failures. With only a modest performance degradation compared to a DRAM cache with no ECC support, our proposal can correct all single-bit failures, and 99.9993% of all row, column, and bank failures, providing more than a 54,000× improvement in the FIT rate of silent-data corruptions compared to basic SECDED ECC protection.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*Cache memories*; B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance

General Terms

Design, Performance, Reliability

Keywords

Reliability, error protection, die stacking, cache

1. INTRODUCTION

Recent proposals for integrating die-stacked DRAM to provide large in-package caches have the potential to improve performance and reduce energy consumption by avoiding costly off-chip accesses to conventional main memory [2, 5, 12, 13, 22, 28, 33]. Die-stacking technology is just beginning to be commercialized [17, 26, 30], but it is limited to certain niche and other market segments that can afford the higher costs of incorporating this new technology.

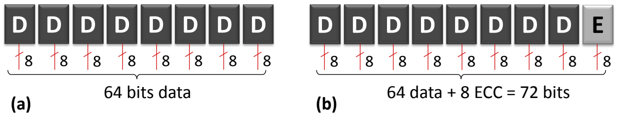

Figure 1: Organization of DRAM chips on a DIMM (one side, x8 chips) for (a) non-ECC DRAM, and (b) ECC DRAM.

Many high-end markets require superior reliability, availability and serviceability (RAS). Die-stacked memories may need to be significantly even more reliable because of their lack of serviceability. Compared to conventional DIMMs that can be easily replaced, a die-stacked memory may require discarding the entire package including the perfectly functioning (and expensive) processor.

Traditionally, RAS for memory has been provided by using error-correcting code (ECC)-enabled DIMMs, where each rank's memory chips are augmented with one or more additional chips to store the ECC/parity information needed to protect the data. Such ECC DIMMs can provide basic single-error correction/double-error detection (SECDED) capabilities, or more complex ECCs (e.g., Reed-Solomon codes [29]) can provide chipkill protection that allows an entire memory chip to fail without compromising any data [3].

For die-stacked DRAM, it may not be practical to add extra chips to provide the additional capacity for storing ECC information. For example, if a DRAM stack has only four chips to begin with, it may not be physically practical to add another "half" chip to provide the extra storage (assuming +12.5% for SECDED). There are other complications with trying to extend a conventional ECC organization to stacked DRAM such as storage/power efficiency and economic feasibility, which we will cover in Section 2. In this work, we propose a series of modifications to a stacked-DRAM cache to provide both fine-grain protection (e.g., for single-bit faults) and coarse-grain protection (e.g., for row-, bank-, and channel-faults) while only utilizing commodity non-ECC stacked DRAM chips. Furthermore, these RAS capabilities can be selectively enabled to tailor the level of reliability for different market needs.

2. BACKGROUND

2.1 ECC for Conventional DRAM

Conventional off-chip memory is organized on dual-inline memory modules (DIMMs), with each side consisting of multiple chips. For the sake of simplicity, this discussion focuses on an organization in which each chip provides eight data bits at a time ("x8"), so the eight chips ganged together implement a 64-bit interface as shown in Figure 1(a). Typically a Hamming code [8] with seven check bits (or a similar code) is used to provide single-bit error correction (SEC) for 64 bits of data. In addition to the SEC code, an additional

ISCA'13 Tel-Aviv, Israel

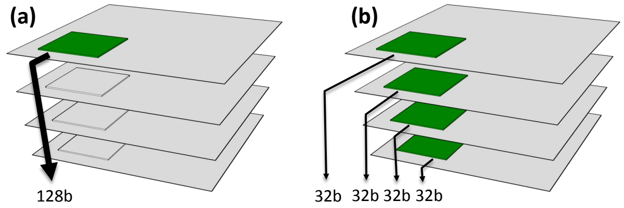

Figure 2: Reading data from a die-stacked DRAM with (a) all data delivered from a single bank from a single layer (similar to JEDEC Wide I/O [11]), and (b) data distributed across banks from four layers.

parity bit is provided to enable double-bit error detection (DED). This introduces eight bits of overhead per 64 bits of data, which is implemented by adding a ninth chip to the DIMM, as shown in Figure 1(b). All 72 bits are read in parallel, and the 8 bits of SECDED coding are used to check and possibly correct one of the 64 bits. Chipkill protection can be achieved in the same area overhead (although typically using x4 chips) by using a Reed-Solomon symbol-correction code and laying out the memory system so each DRAM chip contributes bits to exactly one ECC "symbol" [3].

A key advantage of the conventional ECC DIMM approach is that the silicon for each of the individual chips is identical, which allows the memory manufacturers to incur only the engineering expenses of designing a single memory chip. The difference comes from the design of the DIMM modules: the non-ECC version supports (for example) eight memory chips and the ECC version supports nine, but the engineering cost of designing and manufacturing multiple printed circuit boards is far cheaper than doing the same for multiple chip designs. Maintaining a single chip design also helps optimize a memory vendor's silicon inventory management.

2.2 Die-stacked DRAM

Die-stacked DRAM consists of one or more layers of DRAM with a very-wide data interface connecting the DRAM stack to whatever it is stacked with (e.g., a processor die). Whereas a conventional memory chip may provide only a four- or eight-bit data interface (the reason multiple chips are ganged together on a DIMM), a single layer of die-stacked memory can provide a much larger interface, such as 128 bits [6, 17]. Given this wider interface, all bits for a read request can be transferred across a single interface, and therefore all bits are sourced from a single chip in the stack, as shown in Figure 2(a).

In theory, the stacked DRAM could be organized to be more like a DIMM, in that each of the N chips in a stack provides $\frac{1}{N}^{\text{th}}$ of the bits, as shown in Figure 2(b). This approach is undesirable for a variety of reasons. Requiring the parallel access of N chips means activating banks on all chips. This reduces peak bank-level parallelism by a factor of N, which reduces performance [19]. In addition to the N bank activations, accessing all chips in parallel requires switching N row and column decoders and associated muxes on each access, increasing both the power as well as the number of points of possible failure. Timing skew between different bits coming from different layers for the same request may also make the die-stacked I/O design more challenging. Distributing data across layers also limits flexibility in the design of the stacks. If, for example, data are spread across four layers, then DRAM stack designs will likely be constrained to have a multiple of four DRAMs per stack. In summary, a "DIMM-like" distribution of data across the layers of the DRAM stack is problematic for many reasons.

2.3 DRAM Failure Modes

Conventional DRAM exhibits a variety of failure modes including single-bit faults, column faults, row faults, bank faults, and full-chip faults [10, 35]. These faults can affect one or more DRAM sub-arrays and either be permanent or transient. Recent field studies on DDR-2 DRAM indicate that over 50% of DRAM faults can be large, multi-bit (row, column, bank, etc.) faults, and that DRAM device failure rates can be between 10-100 Failures in Time (FIT) per device, where 1 FIT is one failure per billion hours of operation [35]. Neutron beam testing also shows significant inter-device and inter-vendor variation in failure modes and rates [27].

The internal organization of a die-stacked DRAM bank is similar to an external DRAM, thus failure modes that occur in external DRAM devices are likely to occur in die-stacked DRAM. Die-stacked DRAM may also experience other failure modes, such as broken through-silicon vias (TSVs), and accelerated failure rates from causes such as negative-bias temperature instability (NBTI) and electromigration due to elevated temperatures from being in a stack. Some of these new failure modes (e.g., broken TSVs) will manifest as a single failing bit per row, while others (e.g., electromigration) may cause multiple bits to fail simultaneously.

The cost of an uncorrectable/unrepairable DRAM failure in a die-stacked context may be significantly more expensive than for conventional DIMMs. In a conventional system, a failed DIMM may result in costly down time, but the hardware replacement cost is limited to a single DIMM. For die-stacked memory, the failed memory cannot be easily removed from the package as the package would be destroyed by opening it, and the process of detaching the stacked memory would likely irreparably damage the processor as well. Therefore, the entire package (including the expensive processor silicon) would have to be replaced.

To summarize, die-stacked DRAM RAS must provide robust detection and correction for all existing DRAM failure modes, should be robust enough to handle potential new failure modes as well, and will likely need to endure higher failure rates due to the reduced serviceability of 3D-integrated packaging.

2.4 Applying Conventional ECC to Stacked DRAM

Conventional external DRAM handles the ECC overhead by increasing overall storage capacity by 12.5%. There are two straightforward approaches for this in die-stacked memories. The first is to add additional chips to the stack to provide the capacity, just like what is done with DIMMs. For instance, in a chip stack with eight layers of memory, adding a ninth would provide the additional capacity. In stacked DRAM, however, when the additional chip is used to store ECC information (as in the conventional SECDED ECC), extra contention occurs on the ECC chip as the stacked DRAM does not require sourcing data bits from multiple chips unlike conventional DIMMs (Section 2.2); this makes an ECC check a significant bottleneck under heavy memory traffic, thereby increasing the load-to-use latency of memory requests.

Note that when the nine-chip stack is used as DIMM-like organizations where accesses are distributed across all of the layers, this approach suffers from all of the described shortcomings (e.g., performance/power issues). In addition, if the number of chips in the stack is not equal to eight (or some integral multiple thereof), then adding another chip is not cost effective. For example, in a four-layer stack, adding a fifth layer provides +25% capacity, which may be an overkill when, for example, SECDED only requires +12.5%.

The second straightforward approach is to increase the storage capacity of each individual chip by 12.5%. The width of each row could be increased from, for example, 2KB to 2.25KB, and the

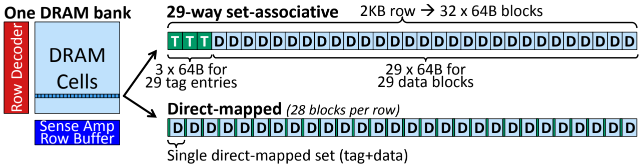

Figure 3: A DRAM bank with 2KB row size. When used as a cache, the row can be organized (top) as a 29-way set-associative set, or (bottom) as 28 individual direct-mapped sets.

data bus width increased correspondingly (e.g., from 128 bits to 144 bits). There are no significant technical problems with this approach, but instead the problem is an economic one. A key to conventional ECC DIMMs is that the same silicon design can be deployed for both ECC and non-ECC DIMMs. Forcing memory vendors to support two distinct silicon designs (one ECC, one non-ECC) greatly increases their engineering efforts and costs, complicates their inventory management, and therefore makes this approach financially undesirable.

2.5 Objective and Requirements

The primary objective of this work is to provide a high level of reliability for die-stacked DRAM caches in a practical and cost-effective manner. From a performance perspective, memory requests usually require data only from a single chip/channel (i.e., bits are not spread across multiple layers of the stack). From a cost perspective, regular-width (non-ECC) DRAM chips must be used. From a reliability perspective, we must account for single-bit faults as well as large multi-bit (row, column, and bank) faults.

In this work, we detail how to provide RAS support for die-stacked DRAM cache architectures while satisfying the above constraints. Furthermore, the proposed approach can provide varying levels of protection, from fine-grain single-bit upsets (SEC coverage), to coarser-grained faults (failure of entire rows or banks), and the protection level can be optionally adjusted at runtime by the system software or hardware.

3. ISOLATED FAULT TYPES IN DRAM CACHES

In this section, we adapt the previously proposed tags-in-DRAM approach [22, 28] to correct single-bit errors and detect multi-bit errors to significantly diminish the probability of silent data corruptions (SDC). Correction of coarser-grained failures (e.g., row, bank or even channel faults) is covered later in Section 4. While many of the constituent components of our overall proposed solutions are similar to or borrow from past works (see Section 7), we provide a novel synthesis and combination of mechanisms to create a new solution for a new problem space.

3.1 Review of Tags-in-DRAM Caches

Several studies have proposed using large, die-stacked DRAMs as software-transparent caches [2, 22]; the primary arguments for using the DRAM as a cache are that it does not require changes to the operating system and performance benefits can be provided for existing software [22]. SRAM-based tag arrays are impractical for conventional cache block sizes (i.e., 64B), and so recent works have proposed embedding tags in the DRAM alongside the data [22].

Figure 3 shows an example DRAM bank, a row from the bank, and the contents of the row. A 2KB row buffer can support 32 64-byte blocks; one previously proposed organization (Figure 3, top) uses 29 of these blocks for data, and the remaining three to implement the tags corresponding to these 29 "ways." It has been shown that a direct-mapped organization, as shown in Figure 3 (bottom), performs better than the set-associative configuration [28], and so for this paper, we use direct-mapped DRAM caches for all experiments although our design easily applies to set-associative caches.

3.2 Supporting Single-bit Error Correction

By placing tags and data together in the same DRAM row, it is relatively easy to reallocate storage between the two types. For example, Figure 3 already provides one example in which the same physical 2KB row buffer can be re-organized to provide either a set-associative or a direct-mapped cache. Only the control logic that accesses the cache needs to be changed; the DRAM itself is oblivious to the specific cache organization.

This same observation that data and tags are "fungible" leads us to a simple way to provide error correction for a DRAM cache. Figure 4(a) shows the components of a basic tag entry and a 64B cache block. This example tag entry consists of a 28-bit tag, a 4-bit coherence state, and an 8-bit sharer vector (used to track inclusion in eight cores)[1]; this example does not include replacement information because we assume a direct-mapped cache. We provide one SECDED ECC code to cover each tag entry, and then a separate SECDED ECC code to cover the corresponding 64B data block. In general, a block of n bits requires a SEC code that is about $\lceil \log_2 n \rceil$ bits wide to support single-bit correction, and then a parity bit to provide double-bit error detection [8]. The tag entry consists of 40 bits in total, thereby requiring a 7-bit SECDED code; the 512 data bits use an 11-bit code.

Placement of tags and data in a single 2KB row requires a little bit of organization to keep blocks aligned. We pack the original tag entry, the ECC code for the tag entry, and the ECC code for the data block into a single combined tag entry. These elements are indicated by the dashed outline in Figure 4(a), which collectively add up to 58 bits. We store eight of these tag entries in a 64B block, shown by the tag blocks ('T') in Figure 4(b). Following each tag block are the eight corresponding data blocks. Overall, a single 2KB row can store 28 64B blocks plus the tags.

Inclusion of ECC codes requires slightly different timing for DRAM cache accesses. Figure 4(c) shows the timing/command sequence to read a 64B block of data from the cache. After the initial row activation, back-to-back read commands are sent to read both the tags and then data [28]. We assume a DRAM interface that can support 32B or 64B reads.[2] The ECC check of the tag entry occurs first to correct any single-bit errors; the corrected tag is then checked for a tag hit. The 64B data block is read in parallel with the tag operations (speculatively assuming a hit), and the data's ECC check is pipelined out over the two 32B chunks. At the end of the ECC check (assuming a cache hit and no errors), the data can be returned to the processor. If the tag must be updated (e.g., transitioning to a new coherence state), the tag entry and the corresponding ECC needs to be updated and then written back to the cache (marked with asterisks in the figure). The timing for a write is similar, and is omitted due to space constraints.

3.3 Supporting Multi-bit Error Detection

In certain mission-critical environments, correction of errors is critical to maintain system uptime. However, many such environments

---

[1] The actual sizes of each field will of course depend on the exact cache size, associativity, coherence protocol, etc.

[2] Current DDR3 supports both sizes, but it does not support switching between them on a per-request basis. We assume that with die-stacked DRAMs, the TSVs provide enough command bandwidth that adding a one-bit signal to choose between single-cycle (32B) or two-cycle (64B) bursts is not an issue.

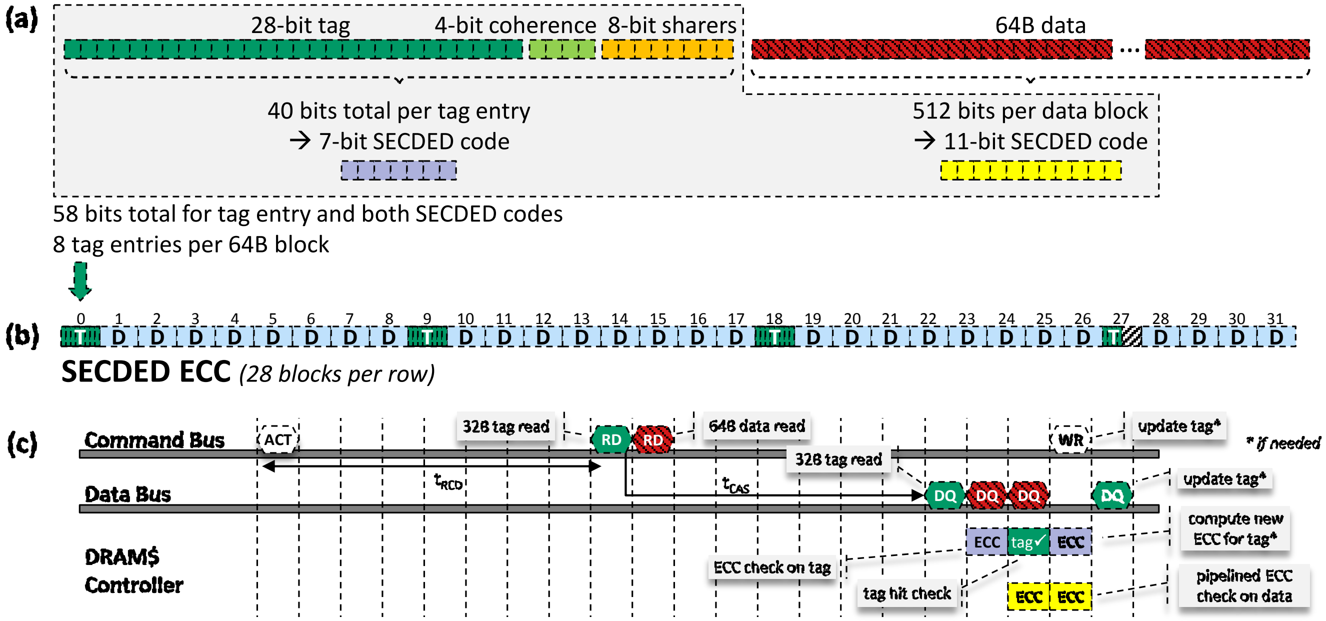

Figure 4: (a) Contents of one tag entry and one 64B data block, along with SECDED ECC codes for each, respectively. (b) Contents of a 2KB DRAM row, with eight tag entries packed into a 64B block and the corresponding eight data blocks following. (c) Timing diagram for reading a 64B cache block; see Section 5 for timing assumptions.

also require robust *detection* of errors (even without hardware correction). It is bad to have a system crash after months of simulation time, but it is even worse for that system to suffer an undetected error and silently produce erroneous results. Therefore, we are likely to want more robust detection than provided by the SECDED ECC scheme in Section 3.2.

For such mission-critical systems, we replace the DED parity bit in each ECC with a very-strong cyclic redundancy check (CRC). While CRCs are traditionally used for bursts of errors in data communications, they can detect much broader classes of errors beyond those that occur in bursts. For example, a 16-bit CRC is capable of detecting all errors of up to 5 bits in 46-bit data (40bit tags+6bit SEC), and all errors of up to 3 bits in 265-bit data (256bit data+9bit SEC), regardless of whether these errors are clustered (i.e., in a burst) or not. Furthermore, these CRCs can detect all burst errors up to 16 bits in length, all errors with an odd number of erroneous bits, and most errors with an even number of erroneous bits [18]. While these CRCs do not increase the DRAM cache's error-correction capabilities, they greatly increase its error-*detection* capability, thereby drastically reducing SDC rates.

Figure 5(a) shows the layout of the tag blocks. Here, we only use SEC codes (not SECDED); the CRCs provide multi-bit error detection and so the parity bit for double-error detection is not needed. We divide the 64B data into two 32B protection regions, each covered by its own SEC and CRC codes. This allows up to two errors to be corrected if each error occurs in a separate 32B region.

The storage requirement for the original tag plus the SEC and CRC codes is 112 bits. Therefore, tag information for four cache blocks can be placed in a 64B block. The overall layout for a 2KB DRAM row is shown in Figure 5(b), with a 64B tag block containing four tags (including SEC/CRC), followed by the four respective data blocks, and then repeated. The increased overhead reduces the total number of data blocks per 2KB row to 25.

The hardware complexity of a CRC is exactly equivalent to that of an equivalent-size ECC. Both CRCs and ECCs are expressed as an H-matrix, and require a logic tree of XOR operations for encode and decode. We assume four DRAM cache cycles to complete the CRC operation on data, which is conservative when considering the logic depth (about 8 XORs deep), additional cycles for wire delay, and so on. For the CRC operation on tags, we use two DRAM cache cycles, which is also conservative.

Figure 5(c) shows the timing for reads, which is very similar to the SECDED-only case from Figure 4, apart from a few minor differences. First, the tag block now contains both ECC and CRC information, so when the tag block must be updated, the final tag writeback is delayed by two extra cycles for the additional latency to compute the new CRC. Second, both the tag and data SEC ECC checks are followed by the corresponding CRC checks. We can return data to the CPU as soon as the ECC check finishes; that is, data can be sent back before the CRC check completes (or even starts). Even if a multi-bit error were detected, there is nothing the hardware can do directly to correct the situation. We assume the hardware simply raises an exception and relies on higher-level software resiliency support (e.g., checkpoint restore) to handle recovery.

3.4 Discussions

Storage Overhead: We use a direct-mapped design in which a single DRAM row contains 28 data blocks. The baseline has enough left-over tag bits to fit the ECC codes; so, SECDED-only can be supported *without* sacrificing further capacity efficiency compared to the non-ECC case. For SEC+CRC, the effective capacity has been reduced from 28 to 25 ways, or an overhead of 3/28 = 10.7%. Compare this to conventional ECC DIMMs in which eight out of nine (or 16 of 18) DRAM chips are used for data, and therefore the corresponding ECC overhead is 1/9 = 11.0%; i.e., the storage overhead of our SEC+CRC approach for DRAM caches is comparable to the effective overhead of conventional off-chip ECC DIMMs.
Controller Support: Our schemes do not require any changes to the stacked DRAM itself; they only require appropriate stacked-DRAM controller support depending on the exact schemes sup-

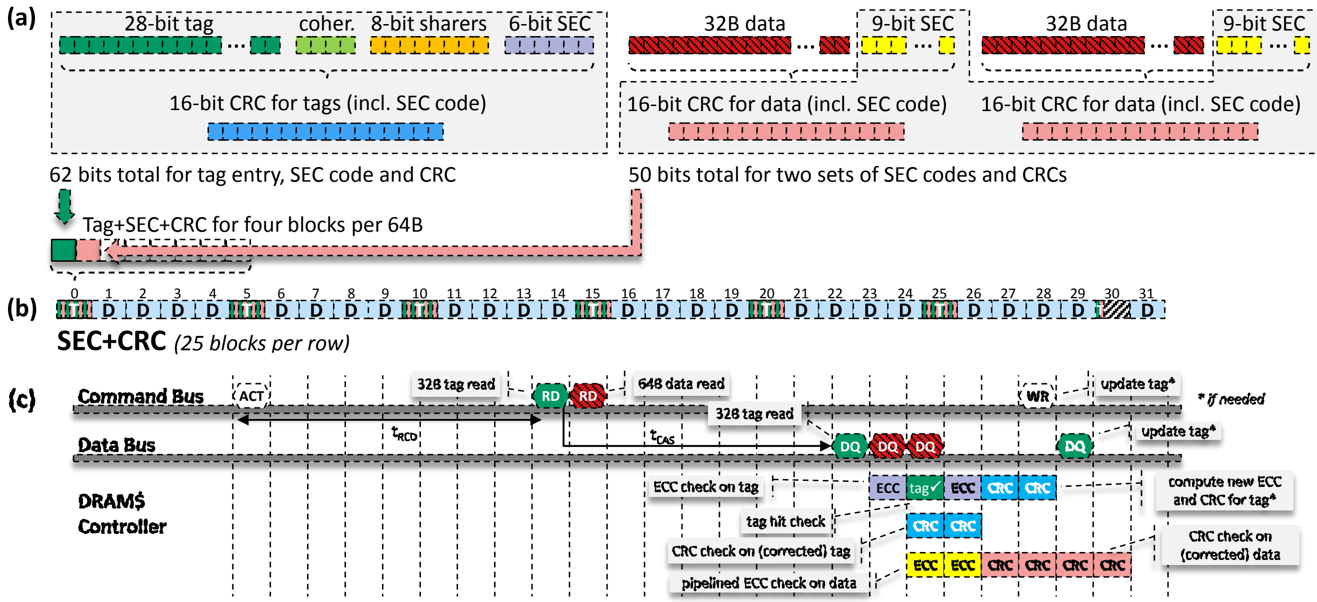

Figure 5: (a) Contents of one tag entry and one 64B data block (treated as two 32B chunks for protection purposes), along with SEC ECC and CRC codes. (b) Contents of a 2KB DRAM row, with four tag entries (tag+SEC+CRC) packed into a 64B block and the corresponding four data blocks following. (c) Timing diagram for reading a 64B cache block; see Section 5 for timing assumptions.

ported. For example, previous work described how non-power-of-two indexing for a direct-mapped DRAM cache can be easily implemented because modulo-by-constant operations are significantly simpler than general-case remainder computations [28]. For a stacked-DRAM controller that supports both SECDED (28 sets per row) and SEC+CRC (25 sets per row), the controller would require two separate modulo-by-constant circuits (i.e., mod 28 and mod 25) and use the appropriate circuit depending on the current RAS mode.

Comparison to Stronger ECC: An alternative approach to our SEC+CRC method is to provide a stronger ECC, such as a Double-Error-Correct Triple-Error-Detect (DECTED) ECC, instead of a CRC. However, note that ECC codes trade detection capability for correction, so they will always detect fewer errors than an equivalent-length CRC. For example, we evaluated using a DECTED ECC and found that it has substantially higher SDC rates than our SEC+CRC (7.9% versus 0.0007%).

4. COARSE-GRAIN FAILURES IN DRAM CACHES

DRAM failure modes are not limited to single-bit/few-bit soft errors from corrupted bitcells. Coarse-grain failures also occur with a non-trivial frequency in real systems [35], affecting entire columns, rows, banks, etc. This section details an approach to deal with coarse-grain DRAM cache failures.

4.1 Identifying Coarse-grain Failures

Before handling failures, the failure must be detected. Here we cover how failures are detected for different scenarios.

Row Decoder Failures: The failure of an entire row can occur due to a fault in the row decoder logic [1]. If the row decoder has a fault in which the wrong wordline is asserted, then the data from the wrong row will be sensed, as shown in Figure 6(a). The DRAM should have returned row 110001's contents: the data Y and the ECC codes for Y, namely E(Y). In this case, however, the adjacent row's contents, X and E(X), are returned instead, but the data and ECC codes are self-consistent, and so the system would not be able to detect that the wrong row had been accessed.

To detect this scenario, we fold in the row index into the data. Figure 6(b) shows example data and ECC fields. We first compute the ECC on the data. Then, instead of storing the raw data, we store the exclusive-OR of the data and several copies of the row index. When reading the data, we first XOR out the row index, which returns the original data; from here, we perform the ECC check.

If the row decoder has a fault, then the wrong row will be read. For example, Figure 6(c) shows a case when reading row 110001_2 results in the previous row 110000_2 instead. We XOR out the row index *for the row that we requested* (e.g., 110001_2), but this leaves the data in a state with multiple "wrong" bits with respect to the stored ECC code. The multi-bit error is detected, and the system can then attempt to do something about it (e.g., raise a fault). A similar approach was proposed in the Argus microarchitecture [23], although our usage covers more fault scenarios due to our CRCs.

Column Failures: An entire column may fail due to problems in the peripheral circuitry. For example, the bitline may not be precharged precisely to the necessary reference voltage, or variations in transistor sizing may make the sense amplifier for a particular column unable to operate reliably. These faults may be permanent failures (e.g., manufacturing defects, wearout) or intermittent (e.g., temperature-dependent). For a bank with a single column failure, reading any row from this bank will result in a corresponding single-bit error. This can be caught and corrected by the baseline ECC. If multiple columns have failed, the baseline CRC checksum can detect the failure in most cases.

Column failures may also occur if the column decoder selects the wrong column (e.g., if the column index was incorrectly latched due to noise). Similar to hashing in the row index already described, we can easily extend the scheme to also XOR in the column index. Prior to reading bits from the DRAM caches, both the row and column indexes are XOR'ed out. An error in either case will cause a CRC mismatch with high probability. The system may

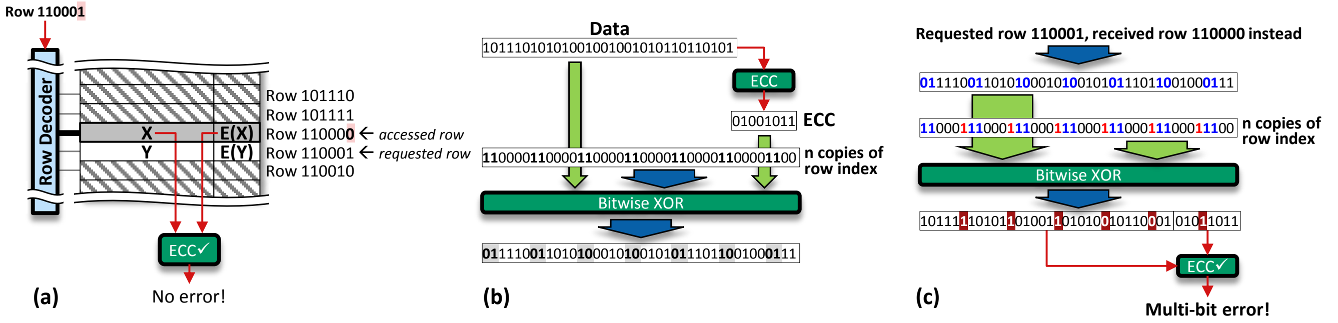

Figure 6: (a) Row decoder error that selects the incorrect row, which is undetectable using within-row ECC. (b) Process for folding in the row index (row 110000_2), and (c) usage of the folded row index to detect a row-decoder error.

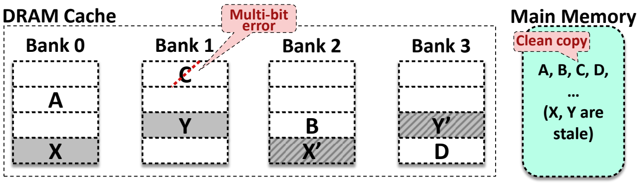

Figure 7: Example DRAM cache contents in which clean data are backed-up by main memory, but dirty data are duplicated into other banks.

record some amount of fault history, and from this it would be very easy to detect that errors consistently occur in a particular column or set of columns. When this has been detected, the system could map out the column using spare resources [16], but such schemes are beyond the scope of this paper.

Bank/Channel Failures: In the case of an entire bank failing, reading *any* row from that bank likely will result in garbage/random bit values, all zeros, or all ones. For random bit values, the probability of the CRC fields being consistent with the data portion will be very low, and so this would manifest itself as an uncorrectable multi-bit error. For all zeros or all ones, instead of just XORing in multiple copies of the row index, some copies (e.g., every other one) are bitwise inverted. Similar to the row-decoder failure, it is possible that the bank decoder fails and sends a request to the wrong bank. The row-index XOR scheme can be extended to include the bank index. The failure of an entire channel and/or channel decoder faults can be treated in a similar manner.

4.2 DOW: Duplicate-on-Write

To tolerate multi-bit errors, row failures and even bank failures, we use a *Duplicate-on-Write* (DOW) approach that has some similarities to RAID-1 used for disk systems, but does not incur the same amount of storage overhead. RAID-1 duplicates *every* disk block (so the filesystem can tolerate the failure of a complete disk), and therefore a storage system must sacrifice 50% of its capacity to provide this level of protection (and 50% of its bandwidth for writes).

The key observation for the DRAM cache is that for *unmodified* data, it is sufficient to detect that an error occurred; the correct data can always be refetched from main memory. For dirty data, however, the modified copy in the cache is the *only* valid copy, and so there is no where else to turn to if this sole copy gets corrupted beyond repair. This observation has been leveraged to optimize the protection levels of physically distinct caches (e.g., parity in the IL1 and SECDED ECC in the DL1), we extend this concept to vary protection within the shared, unified same cache structure.

DOW stores a duplicate copy of data *only when the data are modified*. This way, the duplication overhead (capacity reduction) is limited to only dirty cache blocks. Figure 7 shows a few example cache blocks; blocks A, B, C, and D are all clean, and so the cache stores only a single copy of each. If any are corrupted beyond repair (e.g., C), then the clean copy in memory can provide the correct value.[3] Blocks X and Y are dirty, and so we create duplicate copies X' and Y' in other banks. If the rows (or entire banks) for X or Y fail, we can still recover their values from X' or Y'.

In this example, we use a simple mapping function for placing the duplicate copy. For N banks, a cache line mapped to bank i has its duplicate placed in bank $i + \frac{N}{2} \mod N$. To support channel-kill, the duplicate from channel j is instead mapped to channel $j + \frac{M}{2} \mod M$, assuming M total DRAM cache channels. More sophisticated mapping could reduce pathological conflict cases, but we restrict our evaluations to this simple approach.

Operational Details Here, we briefly explain the mechanics of the different possible scenarios for reading, writing, evictions, etc., and how DOW operates for each of these. The guiding invariants for DOW operation are: (1) if a cache line is clean, then there exists one and only one copy in the DRAM cache, and (2) if a line is dirty, then there exists exactly two copies of the modified data in the DRAM cache.

Read, Clean Data: The cache line is clean; so, only one copy exists in the cache in its original location. This line is used.

Read, Dirty Data: While two copies exist, we simply use the copy located in the original location.

Write, Currently Clean Data: The cache line is now being modified, and so a duplicate must be created to satisfy the invariant that two copies of a dirty line must exist in the cache. A line from the duplicate's target bank must first be evicted, and then the (modified) duplicate is installed. The original copy is modified as well. These operations may be overlapped.

Write, Already Dirty Data: The original copy is modified. When checking the location of the duplicate, there will be a hit and so no additional lines need to be evicted. The duplicate is modified to match the newly updated original copy.

Read/Write, Cache Miss: The bank where the original copy should reside is checked and a miss is discovered. The request is then forwarded directly to main memory. The location to where a duplicate would map is not checked because the duplicate exists only if the line was dirty (by invariant #2). Given that the original location resulted in a miss, the cache line necessarily cannot be dirty and therefore the duplicate cannot exist.

---

[3] We assume some adequate level of protection of main memory; protection of main memory has been well researched and is outside the scope of this paper.

Eviction of Clean Data: This is the only copy in the cache, and it is consistent with main memory, so the cache line may simply be dropped. Updates to the home node may still be required if using a directory-based cache-coherence protocol.

Eviction of Duplicate: The line is dirty and so it must be written back to main memory on an eviction. The original copy either may be invalidated or downgraded to a clean state (our invariants do not permit a single copy of dirty data existing by itself, although one clean copy or zero copies are allowed).

Eviction of Dirty Original: Like the previous case, the line is dirty and so it is written back to main memory. In this case, the duplicate is invalidated; it is not useful to keep the duplicate around even if we downgrade it to a clean state, because on a cache miss to the original cache line's bank, we would proceed directly to main memory and *not* check the duplicate's bank.

Read, Corrupted Data: For a single-bit error, the baseline ECC will correct it and then the read proceeds as normal according to the relevant scenario described previously. In the case of an uncorrectable multi-bit error, if the data is clean, then a request is sent to main memory. The value from memory is returned to the requesting core. If the data is dirty, then the copy from the duplicate is retrieved and returned to the user (assuming that the duplicate has no uncorrectable errors). If both copies have uncorrectable errors, then a fault is raised (i.e., there is nothing more the hardware can do about it). Whether the correct data is provided by main memory or the duplicate copy, the correct data are rewritten into the original location, which effectively corrects the multi-bit errors. Optionally, the data can immediately be read out again and compared to the known-good value. If the data come out corrupted again, then this strongly suggests that the problems are not the result of random soft errors (e.g., high-energy particle strikes), but are in fact due to an intermittent or hard fault of some sort.

Read, Corrupted Tag: If a tag entry has an uncorrectable multi-bit error, then we cannot know for certain whether we would have had a cache hit or miss, nor whether the data would have been dirty or clean. In this case, we must conservatively assume that there was a dirty hit, so we access the duplicate location. If we find the requested block in the duplicate location, then that is the sole surviving copy of the dirty data, which we return to the requesting core and reinstall into the original cache line location. If we do not find the line in the duplicate location, then either the line was not present in the cache to begin with, or it (the original copy) was present but in a clean state when it was corrupted. For either case, it is safe (in terms of correctness) to read the value from main memory.

4.3 Summary of Coverage

The ECC+CRC scheme provides single-error correction and strong multi-error detection for one or more individual bitcell failures as well as column failures. Further layering DOW provides row-failure coverage and complete bank-kill and channel-kill protection. The idea of providing two different levels of protection is similar in spirit with the memory-mapped ECC proposal [39], although the details for our DRAM cache implementation are completely different. In the best case for our approach, each of the N banks is paired with one other bank (in another channel), and so one bank may fail from each of the $\frac{N}{2}$ pairs. If a system implements multiple DRAM stacks with channels interleaved across stacks, then DOW automatically provides *stack-kill* protection as well. Note that while this paper focuses on a specific configuration and corresponding set of parameters, the proposed *approach* is general and can be tailored to specific stacked-DRAM performance and capacity attributes as well as the overall RAS requirements for different systems.

4.4 Optimizations for DOW

While DOW limits the duplication overhead to only those cache lines that have been modified, in the worst case (i.e., when *every* cache line is dirty) the overhead still degenerates back to what a "RAID-1" solution would cost. We now discuss several variants on DOW that can reduce duplication overheads.

Selective DOW: Not all applications and/or memory regions are critical, and therefore not all need to have the high-level of reliability that DOW provides. IBM's Power7 provides a feature called *selective memory mirroring* where only specific portions of memory regions are replicated into split channel pairs [15]. Similarly, dirty-data duplication (DOW) can be selectively applied to specific applications and/or memory regions. For example, critical operating-system data structures may require strong protection to prevent system-wide crashes, but low-priority user applications need not be protected beyond basic ECC. Even within an application, not all pages need to be protected, although to support this level of fine-grain coverage, the application must provide some hints about what should be protected.

Background Scrubbing: On-demand scrubbing can occur based on the amount of dirty data in the cache. Each time a dirty line is added to the cache, a counter is incremented (and likewise decremented for each dirty eviction). If the counter exceeds a threshold, then scrubbing is initiated. The scrubbing would proceed until the number of dirty lines drops below a "low-water mark" (some value less than the threshold) to prevent the scrubbing from constantly kicking in. The writeback traffic can be scheduled during bus idle times and/or opportunistically when a DRAM row is already open. This can also be used in concert with eager-writeback techniques [20].

Duplication De-prioritization: When initially installing duplicates, or when updating existing duplicates, these writes are not typically on the program's critical path (these are the result of dirty line evictions from the upper-level caches, not the result of a demand request). The DRAM cache's controller can buffer the duplicate write requests until a later time when the DRAM cache is less busy, thereby reducing the bank-contention impact of the duplicate-update traffic.

5. EXPERIMENTAL RESULTS

5.1 Methodology

Simulation Infrastructure: We use a cycle-level x86 simulator for our evaluations [9]. We model a quad-core processor with two-level SRAM caches and an L3 DRAM cache. The stacked DRAM is expected to support more channels, banks, and wider buses per channel [6, 17, 21]. In this study, the DRAM cache has eight channels each with 128-bit buses, and each channel has eight banks [14], while the off-chip DRAM has two channels, each with eight banks and a 64-bit bus. Also, key DRAM timing parameters with bank conflicts and data bus contention are modeled for both the DRAM cache and main memory. Virtual-to-physical mapping is also modeled to ensure that the same benchmarks in different cores do not map into the same physical addresses. Table 1 shows the system configurations used in the study.

Workloads: We use the SPEC CPU2006 benchmarks and sample one-half billion instructions using SimPoint [32]. We then categorize the applications into two different groups based on the misses per thousand instructions (MPKI) in the L2 cache. We restrict the study to workloads with high memory traffic; applications with low memory demands have very little performance sensitivity to memory-system optimizations and therefore expose very little additional insight (we did verify that our techniques do not negatively

Processors	
Core	4 cores, 3.2 GHz out-of-order, 4 issue width
L1 cache	4-way, 32KB I-Cache + 32KB D-Cache (2-cycle)
L2 cache	16-way, shared 4MB (24-cycle)
Stacked DRAM caches	
Cache size	128 MB
Bus frequency	1.6 GHz (DDR 3.2GHz), 128 bits per channel
Channels/Ranks/Banks	8/1/8, 2048 bytes row buffer
tCAS-tRCD-tRP	8-8-8
tRAS-tRC	26-34
Off-chip DRAM	
Bus frequency	800 MHz (DDR 1.6GHz), 64 bits per channel
Channels/Ranks/Banks	2/1/8, 16KB row buffer
tCAS-tRCD-tRP	11-11-11

Table 1: System parameters used in the study.

Group M	MPKI	Group H	MPKI
milc	18.59	leslie3d	27.69
wrf	19.04	libquantum	28.39
soplex	23.73	astar	29.26
bwaves	24.29	lbm	36.62
GemsFDTD	26.44	mcf	52.65

Table 2: L2 misses per thousand instructions (L2 MPKI).

Mix	Workloads	Group
WL-1	4 × mcf	4×H (rate)
WL-2	4 × leslie3d	4×H (rate)
WL-3	mcf-lbm-milc-libquantum	4×H
WL-4	mcf-lbm-libquantum-leslie3d	4×H
WL-5	libquantum-mcf-milc-leslie3d	4×H
WL-6	mcf-lbm-libquantum-soplex	3×H + 1×M
WL-7	mcf-milc-wrf-soplex	2×H + 2×M
WL-8	lbm-leslie3d-wrf-soplex	2×H + 2×M
WL-9	milc-leslie3d-GemsFDTD-astar	2×H + 2×M
WL-10	libquantum-bwaves-wrf-astar	1×H + 3×M
WL-11	bwaves-wrf-soplex-astar	1×H + 3×M
WL-12	bwaves-wrf-soplex-GemsFDTD	4×M

Table 3: Multi-programmed workloads.

Failure Mode	Failure Rate (FIT)
Single Bit	33
Complete Column	7
Complete Row	8.4
Complete Bank	10

Table 4: Failure rates measured on external DRAM [35].

impact these benchmarks). From the memory-intensive benchmarks, those with average MPKI rates greater than 27 are in Group H (for **H**igh intensity), and of the remaining, those with 15 MPKI or more are in Group M (for **M**edium).

We select benchmarks to form rate-mode (all cores running separate instances of the same application) and multi-programmed workloads. Table 3 shows the primary workloads evaluated for this study. Section 6 also includes additional results covering a much larger number of workloads. We simulate 500 million cycles of execution for each workload. We verified that this simulation length is sufficiently long to cause the contents of the 128MB DRAM cache to turn over many times (i.e., the DRAM cache is more than sufficiently warmed up).

Performance Metric: We report performance of our quad-core system using weighted speed-up [7, 34], which is computed as:

$$\text{Weighted Speed-up} = \sum_{i} \frac{\text{IPC}_i^{\text{shared}}}{\text{IPC}_i^{\text{single}}}$$

We use the geometric mean to report average values.

Failure Modes and Rates: We assume DRAM failure modes and rates similar to those observed from real field measurements (FIT in Table 4 [35]). We report results using both the observed failure rates (Table 4) as well as failure rates of 10× the observed rates to account for potential increases in failures due to die stacking and for inter-device variations in failure rates [27]. At this point, we are not aware of any published studies reporting failure types and rates for die-stacked DRAM. The assumption that failure rates will be 10× may be somewhat pessimistic, but by providing our results across the range of 1-10× the currently known FIT rates, the actual stacked DRAM results should fall somewhere in between. Furthermore, the *relative* mean time to failure (MTTF) rates for our proposed schemes compared to the baseline are independent of the underlying device FIT rates.

ECC and CRC simulation: We evaluate the performance of our ECC and ECC+CRC schemes using Monte Carlo simulation assuming a bit error rate (BER) of 0.5 in a faulty DRAM location (e.g., within a faulty row). This bit error rate corresponds to a 50% probability that a given bit is in error, and is chosen because the erroneous value returned by the DRAM will sometimes match the expected data value. For example, a DRAM row fault will not produce errors on every bit within that row (which would correspond to a BER of 1), but rather only on those bits where the returned value does not match the expected value.

We separately simulate row, column, and bank faults, and run 100 million simulations for each type of fault. Each simulation randomly flips bits within a single row, column, or bank (with a probability of 50%), and then applies the ECC and CRC logic to determine whether the error will be corrected or detected. We do not model row decoder or column decoder faults; we assume these are caught 100% of the time by the row and column address XOR.

To calculate failure rates, we apply the detection and correction coverage to previously reported FIT rates [35]. For the purposes of this work, we assume all undetected errors result in silent data corruption (SDC). This is likely to be somewhat conservative due to fault masking effects [24], but note that our relative improvements will remain accurate.

In evaluating DOW, we assume a single-fault model, i.e., only one fault at a time will exist in the DRAM cache. Similar to traditional chipkill, DOW does not guarantee correction when there are multiple independent faults in the DRAM cache (i.e., DOW provides "bank-kill" or "channel-kill" capability, depending on placement of the duplicate line). Unlike traditional chipkill, there is a very small likelihood that DOW will not detect all two-fault cases. This can occur if both the original and duplicate lines have a fault and the duplicate's CRC mismatches. Given the failure rates and detection coverage, the failure rate of this case is less than 1e-13 FIT, or once per 80 quadrillion years for a four-chip DRAM cache.

5.2 Fine-grain Protection

We first evaluate the performance impact of the proposed fine-grain protection schemes on DRAM caches. Figure 8 shows the speed-up over no DRAM cache between no RAS, ECC and ECC+CRC configurations. The results show that the performance impact of our proposed schemes is small. On average, the ECC and ECC+CRC

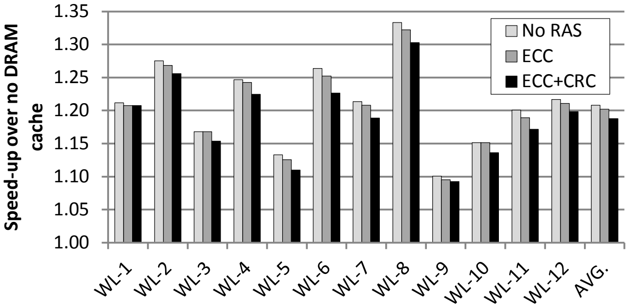

Figure 8: Performance comparison among no RAS, ECC, and ECC+CRC (normalized to no DRAM cache).

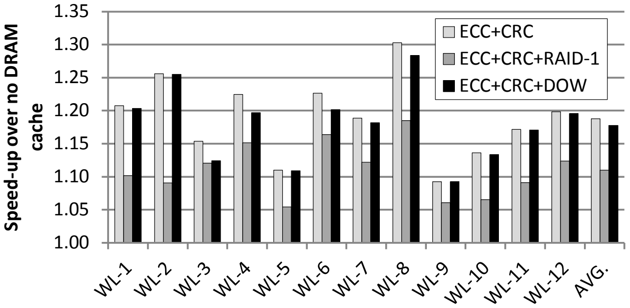

Figure 9: Performance comparison between fine-grain (ECC+CRC) and coarse-grain (ECC+CRC+RAID-1 and ECC+CRC+DOW) schemes (normalized to the performance without the DRAM cache).

schemes only degrade an average of 0.50% and 1.68% compared to a DRAM cache with no RAS support, respectively. ECC+CRC reduces the cache capacity compared to no RAS, and it also slightly increases bandwidth consumption; we further analyze the capacity and bandwidth impact of ECC+CRC in Section 6.2.

5.3 Coarse-grain Protection

Figure 9 shows the performance of our proposed coarse-grain protection scheme (DOW) when applied on top of ECC+CRC. We compare the results with ECC+CRC and ECC+CRC+RAID-1 to show the effectiveness of DOW. In the case of the RAID-1-style approach (i.e., full duplication), not only is cache capacity reduced by half, but effective write bandwidth is reduced by half as well, thereby leading to a non-negligible performance degradation of as much as 13.1% compared to ECC+CRC (6.5% on average). However, the DOW scheme retains much of the overall performance benefit of having a DRAM cache (on average, only -2.5% and -0.8% performance loss compared to no RAS and ECC+CRC, respectively) while providing substantial RAS improvements.

5.4 Fault Coverage and Failure Rates

Table 5 shows the percentage of faults detected in each failure mode by each of our schemes, assuming a single four-layer stack of DRAM. ECC-only detects all single-bit faults and most column faults, because most of these faults affect only one bit per row [35]. ECC-only also detects 50% of all row and bank faults, which look to the ECC like double-bit errors. ECC+CRC substantially improves the detection coverage of column, row, and bank faults, detecting 99.9993% of all faults. The detection percentage of the CRC depends on the fault model. We assume that every bit in the row or bank has a 50% chance of being incorrect; lower error rates (as might be observed with different failure modes) substantially increase the detection percentage of the CRC. We also conservatively assume that row and bank failures are due to all bits in the row or bank being bad, rather than to row or bank decoder failures which would be caught by XORing the row and bank address.

	Detection			
Failure Mode	**No RAS**	**ECC Only**	**ECC+CRC**	**DOW**
Single Bit	0%	100%	100%	100%
Column	0%	85%	99.9993%	99.9993%
Row	0%	50%	99.9993%	99.9993%
Bank	0%	50%	99.9993%	99.9993%

Table 5: Detection coverage for each technique.

	Correction			
Failure Mode	**No RAS**	**ECC Only**	**ECC+CRC**	**DOW**
Single Bit	0%	100%	100%	100%
Column	0%	85%	▷ 85% ◁	99.9993%
Row	0%	▷ 0% ◁	▷ 0% ◁	99.9993%
Bank	0%	▷ 0% ◁	▷ 0% ◁	99.9993%

Table 6: Correction coverage for each technique. Cases where correction coverage differs from detection coverage (Table 5) are marked with ▷◁.

Failure Rates	No RAS	ECC Only	ECC+CRC	DOW
SDC FIT	234-2335	41-410	0.0008-0.0075	0.0008-0.0075
DUE FIT	0	37-368	52-518	0

Table 7: Results using observed and 10× DRAM failure rates.

Table 6 shows the fraction of faults, by failure mode, that our schemes can correct. ECC-only and ECC+CRC correct all single-bit and 85% of column faults, but cannot correct any row or bank faults.[4] DOW, on the other hand, corrects all detected faults.

Table 7 shows the overall silent data corruption (SDC) and detectable unrecoverable error (DUE) FIT rates for our techniques, using both observed and 10× DRAM failure rates. We assume that all undetected failures will cause a silent data corruption. No RAS leads to a SDC FIT of 234-2335, or an SDC MTTF of 49-488 years. ECC-only reduces the SDC FIT by 5.7×, but increases the DUE FIT to 37-368 FIT. ECC+CRC reduces the SDC FIT to just 0.0008-0.0075 FIT, but this comes at the expense of an increase in DUE FIT to 52-518 (220-2200 years MTTF). Finally, DOW adds the ability to correct all detected errors while maintaining the same SDC FIT as ECC+CRC. Overall, DOW provides a more than 54,000× improvement in SDC MTTF compared to the ECC-only configuration. While the reported MTTF rates may appear to be adequately long, these can still result in very frequent failures in large datacenter or HPC installations. The impact on such systems will be further quantified in Section 6.

6. ANALYSIS AND DISCUSSIONS

6.1 Reliability Impact on Large System Sizes

The bandwidth requirements of future HPC systems will likely compel the use of significant amounts of die-stacked DRAM. The impact of DRAM failures in these systems is significantly worse than for single-socket systems because FIT rates are *additive* across nodes. For example, using the baseline (1×) FIT rates from Table 4, a 100,000-node HPC system with four DRAM stacks per

---

[4] Row and bank faults with only a single bit in error can be corrected by these schemes, but we assume that this does not occur.

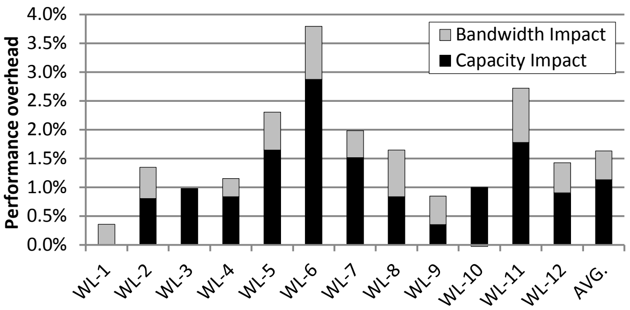

Figure 10: Capacity and bandwidth impact of the ECC+CRC scheme.

node would have an SDC MTTF of only 10 hours from the die-stacked DRAM alone with no RAS support. Our ECC-only technique would increase the SDC MTTF to only 60 hours. By contrast, ECC+CRC and DOW have an SDC MTTF of 350 years for the entire 100,000-node system. Inclusion of DOW is likely necessary, because ECC+CRC (without DOW) has a 48-hour DUE MTTF on such a system. While DUEs might be handled by software techniques, the performance overheads of restoring the system to a checkpoint every two days may not be acceptable. This analysis optimistically assumes the baseline DRAM FIT rates, which are likely lower than what will be observed with die-stacked DRAM (e.g., if we assume a 10× FIT rate, then without DOW, the system would have to rollback to a checkpoint about every five hours).

6.2 Capacity and Bandwidth Impact

With RAS support, the performance of DRAM caches is lower due to reduced cache capacity as well as higher latency/bandwidth. To quantify the capacity impact, we evaluate a DRAM cache configuration with the same capacity as ECC+CRC, but without the additional traffic. The bottom portion of the bars in Figure 10 shows the performance *reduction* due to the cache capacity reduction alone. The remaining performance loss comes from the additional bandwidth and latency effects of handling the ECC and CRC codes.

6.3 Impact of Early Data Return

As described in Section 3.3, our ECC+CRC scheme (no DOW) returns data before the completion of data CRC checks, which is based on the observation that hardware cannot correct the multi-bit errors anyway; thus, we do not need to wait for the check to be finished for every cache hit. We also simulated a version of ECC+CRC where we wait for the full CRC check to complete before returning data, which may be of use to support data poisoning. On average, this degrades performance by only 0.5%; this is because the CRC check adds only four more DRAM cache cycles to the load-to-use latency, which is a small relative increase compared to the latency of activating and reading the DRAM cache row in the first place.

6.4 Duplication Overheads of DOW

While DOW provides much better performance than a naive RAID-1 approach, DOW still causes extra data duplication and writeback traffic. Figure 11(a) shows the percentage of cachelines in the DRAM cache that are dirty for ECC+CRC and also when DOW is added. One would expect that the amount of dirty data should *increase* due to duplication, but in fact the amount of dirty data decreases because of the maintenance of the invariant that all dirty cache lines must have two copies. Thus, if *either* copy is evicted, then the amount of dirty data in the cache goes down. The impact of this is that there is an increase in the DRAM cache's writeback traffic. Figure 11(b) shows the number of writebacks from the DRAM cache per thousand cycles. For most workloads, the increase in writeback traffic is not significant. The standouts are workloads WL-3, WL-4, WL-6 and WL-8 that start with relatively higher amounts of writeback traffic, and then DOW causes that traffic to increase by another approximately 20%. Not surprisingly, these are also the workloads that exhibited the largest performance losses when applying DOW (see Figure 9). While some applications show moderate increases in writeback traffic, the absolute traffic is still low, which explains why DOW does not have a significant negative performance impact in most cases.

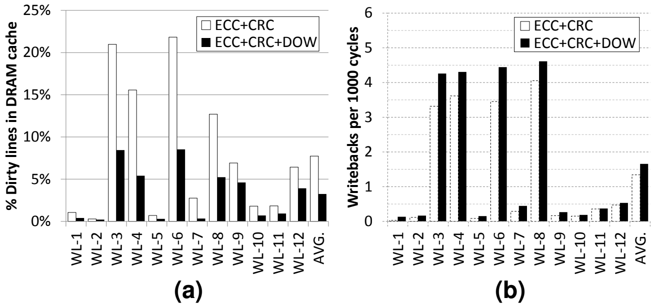

Figure 11: Impact of DOW on (a) the amount of dirty lines in the DRAM cache and (b) the writeback traffic from the DRAM cache.

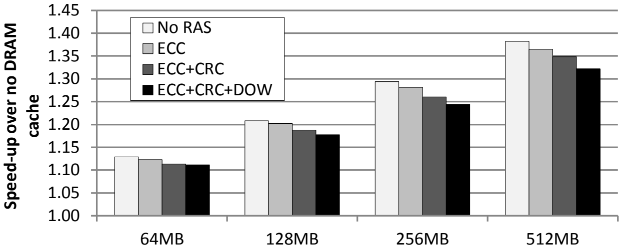

Figure 12: Sensitivity to different cache sizes (workloads in Table 3).

6.5 Sensitivity to Cache Size

Figure 12 shows the average speed-up of no RAS, ECC, ECC+CRC, and ECC+CRC+DOW over no DRAM cache with different cache sizes. For the fine-grain protection schemes, the performance degradation is relatively small across the cache sizes. For DOW, the relative performance loss increases with larger caches, which is due to an increase in the amount of duplicated data. However, DRAM caches with DOW still deliver the majority of the performance benefit of the no RAS approach while providing far superior RAS capabilities.

6.6 Sensitivity to Different Workloads

We apply our protection schemes to all $\binom{10}{4} = 210$ combinations of the applications in Table 2 to ensure that the performance impact is also minimal for a broader set of workloads beyond the primary workloads used in the paper. Figure 13 presents the average speed-up (with ± one standard deviation) over the 210 workloads. The proposed RAS capabilities have a relatively small impact on performance. ECC, ECC+CRC and ECC+CRC+DOW degrade performance by only 0.50%, 1.65% and 2.63% on average over no RAS, respectively.

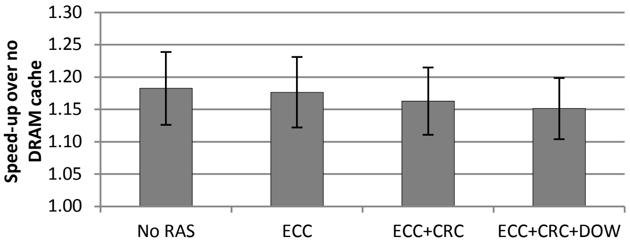

Figure 13: Average speed-up of no RAS, ECC+CRC and ECC+CRC+DOW over the 210 workloads.

6.7 Value of Configurable RAS Levels

The proposed protection schemes are just a few possible implementations of RAS support for DRAM caches. Other approaches are possible, from providing only light-weight parity-based error-detection, to very robust multi-bit error correction (e.g., BCH or Reed-Solomon codes). The more sophisticated schemes may require higher overheads, which reduce the overall data storage capacity or available bandwidth, but the high-level approach described here (e.g., embedding tags, ECC, and data in the same DRAM row) allows the designer to make the appropriate performance-versus-resiliency trade-offs.

At the implementation level, the DRAM cache is just a commodity memory component with no knowledge of how the data storage is being used. This allows system designers (OEMs) to take a single processor (with stacked DRAM) design but configure different levels of protection for different deployments of the design. In a commodity consumer system, one may choose to turn off ECC entirely and make use of the full DRAM cache capacity. In servers and certain embedded systems, basic SECDED ECC may be sufficient; this is comparable to the level of protection used today (e.g., SECDED ECC DIMMs). In mission-critical enterprise servers and HPC supercomputers, the full ECC+CRC and DOW protections could be used. The selection of which level of protection to use could be configured simply by a BIOS setting read during system boot (e.g., NVIDIA's Fermi GPU can enable/disable ECC in the GDDR5 memories with a reboot [25]).

Protection levels conceivably could be configured dynamically. Critical memory resources (e.g., operating system data structures) may receive a high level of protection, while other low-priority user applications may receive no or minimal protection. The level of protection could be reactive to the observed system error rates; e.g., a system may by default use only ECC+CRC fine-grain protection, but as the number of corrected errors increases beyond a certain threshold, DOW is enabled to prevent data loss. Likewise, this selective protection could be applied to only specific banks or even rows of the DRAM cache if elevated error rates are localized. The enabled protection level can be increased gradually as a product slowly suffers hardware failures from long-term wear-out [4, 31].

7. RELATED WORK

As far as we are aware, this is the first work to target increased RAS capabilities in die-stacked DRAM designs. However, several recent works target more power-efficient external DRAM implementations, and bear some similarities to the techniques described in this paper. Single-subarray access (SSA) proposes fetching an entire cache line from a single DRAM device, using a RAID scheme to support chipkill [37]. LOT-ECC and virtualized ECC propose supporting chipkill with x8 DRAM devices by separating the error-detection and error-correction steps and storing some ECC information in data memory [36, 40]. The tiered coverage of memory-mapped ECC [39] and LOT-ECC share some similarities with the discussed ECC+CRC+DOW approaches, but the structures and implementations are quite different. Mini-rank proposes storing ECC bits along with the associated data bits to efficiently support RAS for their mini-rank design [42]. The embedded ECC complicates memory address translation (this technique is proposed for main memory), but the approach taken in our work does not affect the address translation as we store ECC information in the tag storage and thus do not change the data block size. Abella et al. propose adding hard-wired ROM entries to each row of an SRAM to provide a hard-wired row index when reading out data to detect row-decoder errors [1], but such an approach does not extend easily to commodity DRAMs. All of these techniques would require significant modifications to support die-stacked DRAM architectures.

The idea of replicating cache blocks has also been studied, although in a different context. In-cache replication (ICR) improves the resiliency of level-one data (DL1) caches [41]. ICR differs from the proposed scheme in that the structures, constraints, and performance implications of the DL1 force a very different design from DOW; ICR makes use of dead-line predictors, more complex indexing of duplicate copies (possibly with multiple locations), and in some variants ICR duplicates both clean and dirty data.

There has also been a significant amount of recent research in architecting die-stacked DRAM as large caches [2, 5, 13, 22, 28, 33], but to our knowledge, none of these have addressed the problems of providing robust error correction and detection capabilities for such large, in-package storage arrays. Micron's Hybrid Memory Cube provides ECC support for its 3D DRAM stacks [26], but this requires custom DRAM chips, whereas the proposed approaches are compatible with commodity non-ECC DRAM.

8. CONCLUSIONS

Stacked DRAM caches may play a significant role in attacking the Memory Wall [38] going forward, but the technology will be economically viable only if it can be deployed in sufficient volume. Only a minority segment of the market demands RAS support, so developing ECC-specific stacked DRAM chips is likely undesirable for memory manufacturers. This work presented a general approach for enabling high levels (well beyond current ECC schemes) as well as configurable levels of RAS that works within the constraints of commodity, non-ECC DRAM stacks. Beyond the benefit to memory vendors from having to support only a single (non-ECC) DRAM chip design, this approach also benefits system vendors (i.e., OEMs, ODMs) who can stock a single processor type and then deploy it in consumer systems with the stacked-DRAM ECC support disabled while using the same part in workstations and servers with RAS support turned on.

Acknowledgments

We would like to thank the Georgia Tech HPArch members and the anonymous reviewers for their suggestions and feedback. Part of this work was conducted while Jaewoong Sim was on an internship at AMD Research. Jaewoong Sim is also supported by NSF award number 1139083.

9. REFERENCES

[1] J. Abella, P. Chaparro, X. Vera, J. Carretero, and A. González. On-Line Failure Detection and Confinement in Caches. In *IOLTS*, 2008.

[2] B. Black, M. M. Annavaram, E. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCauley, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. P. Shen, and

C. Webb. Die-Stacking (3D) Microarchitecture. In *MICRO-39*, 2006.

[3] T. J. Dell. A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory. 1997.

[4] F. M. d'Heurle. Electromigration and Failure in Electronics: An Introduction. *Proceedings of the IEEE*, 59(10), 1971.

[5] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi. Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support. In *SC*, 2010.

[6] Elpida Corporation. Elpida Completes Development of TSV (Through Silicon Via) Multi-Layer 8-Gigabit DRAM. Press Release, August 27, 2009. http://http://www.elpida.com/en/news/2009/index.html.

[7] S. Eyerman and L. Eeckhout. System-Level Performance Metrics for Multiprogram Workloads. *IEEE Micro Magazine*, 28(3), May–June 2008.

[8] R. W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29(2), 1950.

[9] HPArch. MacSim Simulator. http://code.google.com/p/macsim/.

[10] A. A. Hwang, I. A. Stefanovici, and B. Schroeder. Cosmic Rays Don't Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design. In *ASPLOS-XVII*, 2012.

[11] JEDEC. Wide I/O Single Data Rate (Wide I/O SDR). http://www.jedec.org/standards-documents/docs/jesd229.

[12] D. Jevdjic, S. Volos, and B. Falsafi. Footprint Cache: Effective Page-based DRAM Caching for Servers. In *ISCA-40*, 2013.

[13] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian. CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms. In *HPCA-16*, 2010.

[14] Joint Electron Devices Engineering Council. JEDEC: 3D-ICs. http://www.jedec.org/category/technology-focus-area/3d-ics-0.

[15] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd. Power7: IBM's Next-Generation Server Processor. *IEEE Micro Magazine*, 30(2), March–April 2010.

[16] S. Kikuda, H. Miyamoto, S. Mori, M. Niiro, and M. Yamada. Optimized Redundancy Selection Based on Failure-Related Yield Model for 64-Mb DRAM and Beyond. *IEEE Journal of Solid-State Circuits*, 26(11), 1991.

[17] J.-S. Kim et al. A 1.2V 12.8GB/s 2Gb Mobile Wide-I/O DRAM with 4x128 I/Os Using TSV-Based Stacking. In *ISSCC*, 2011.

[18] P. Koopman and T. Chakravarty. Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks. In *DSN*, 2004.

[19] C. J. Lee, V. Narasiman, O. Mutlu, and Y. N. Patt. Improving Memory Bank-Level Parallelism in the Presence of Prefetching. In *MICRO-42*, 2009.

[20] H.-H. S. Lee, G. S. Tyson, and M. K. Farrens. Eager Writeback - a Technique for Improving Bandwidth Utilization. In *MICRO-33*, 2000.

[21] G. H. Loh. 3D-Stacked Memory Architectures for Multi-Core Processors. In *ISCA-35*, 2008.

[22] G. H. Loh and M. D. Hill. Efficiently Enabling Conventional Block Sizes for Very Large Die-Stacked DRAM Caches. In *MICRO-44*, 2011.

[23] A. Meixner, M. E. Bauer, and D. Sorin. Argus: Low-Cost, Comprehensive Error Detection in Simple Cores. In *MICRO-40*, 2007.

[24] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor. In *MICRO-36*, 2003.

[25] NVidia Corp. Tuning CUDA Applications for Fermi. DA 05612-001_v1.5, 2011.

[26] J. T. Pawlowski. Hybrid Memory Cube: Breakthrough DRAM Performance with a Fundamentally Re-Architected DRAM Subsystem. In *Hot Chips 23*, 2011.

[27] H. Quinn, P. Graham, and T. Fairbanks. SEEs Induced by High-Energy Protons and Neutrons in SDRAM. In *REDW*, 2011.

[28] M. K. Qureshi and G. H. Loh. Fundamental Latency Trade-offs in Architecting DRAM Caches. In *MICRO-45*, 2012.

[29] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2), 1960.

[30] K. Saban. Xilinx Stacked Silicon Interconnect Technology Delivers Breakthrough FPGA Capacity, Bandwidth, and Power Efficiency. White paper, Xilinx, 2011. WP380 (v1.1).

[31] D. K. Schroder and J. A. Babcock. Negative Bias Temperature Instability: Road to Cross in Deep Submicron Silicon Semiconductor Manufacturing. *Journal of Applied Physics*, 94(1):1–18, 2003.

[32] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *ASPLOS-X*, 2002.

[33] J. Sim, G. H. Loh, H. Kim, M. O'Connor, and M. Thottethodi. A Mostly-Clean DRAM Cache for Effective Hit Speculation and Self-Balancing Dispatch. In *MICRO-45*, 2012.

[34] A. Snavely and D. Tullsen. Symbiotic Job Scheduling for a Simultaneous Multithreading Processor. In *ASPLOS-IX*, 2000.

[35] V. Sridharan and D. Liberty. A Study of DRAM Failures in the Field. In *SC*, 2012.

[36] A. Udipi, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. P. Jouppi. LOT-ECC: LOcalized and Tiered Reliability Mechanisms for Commodity Memory Systems. In *ISCA-39*, 2012.

[37] A. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi. Rethinking DRAM Design and and Organization for Energy-Constrained Multi-Cores. In *ISCA-37*, 2010.

[38] W. A. Wulf and S. A. McKee. Hitting the Memory Wall: Implications of the Obvious. *Computer Architecture News*, 23(1):20–24, March 1995.

[39] D. H. Yoon and M. Erez. Memory Mapped ECC:Low-Cost Error Protection for Last Level Caches. In *ISCA-36*, 2009.

[40] D. H. Yoon and M. Erez. Virtualized and Flexible ECC for Main Memory. In *ASPLOS-XV*, 2010.

[41] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam. ICR: In-Cache Replication for Enhancing Data Cache Reliability. In *DSN*, 2003.

[42] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. Mini-Rank: Adaptive DRAM Architecture for Improving Memory Power Efficiency. In *MICRO-41*, 2008.