The Dissertation Committee for Doe Hyun Yoon
certifies that this is the approved version of the following dissertation:

Flexible and Efficient Reliability in Memory Systems

Committee:

Mattan Erez, Supervisor

Yale N. Patt

Nur A. Touba

Derek Chiou

Jian Li

Flexible and Efficient Reliability in Memory Systems

by

Doe Hyun Yoon, B.S.; M.S.; M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2011

To my family.

Acknowledgments

First of all, I would like to thank my advisor, Professor Mattan Erez. Mattan has been a great advisor, providing me an opportunity to work on computer architecture research, keeping me focusing on relevant research topics, sharing his research experience with me, and helping me interact with various other researchers (inside and outside the university).

I also would like to thank the other members of my dissertation committee: Professor Yale N. Patt, Professor Nur A. Touba, Professor Derek Chiou, and Dr. Jian Li. Their keen insights and comments helped me develop and elaborate the concepts in my dissertation.

I had an opportunity to collaborate with the researchers at HP Labs, and I would like to thank my mentors at HP Labs, including Naveen Muralimanohar, Jichuan Chang, Parthasarathy Ranganathan, and Norman P. Jouppi. They helped me understand a broader aspect of computer systems and why reliability is (really) important in large-scale systems, especially when emerging memory technologies are used.

I also should acknowledge my friends in the LPH research group: Mehmet Basoglu, Jinsuk Chung, Min Kyu Jeong, Evgeni Krimer, Ikhwan Lee, Karthik Shankar, Jongwook Sohn, and Michael Sullivan. Always, they were willing to

give me invaluable comments and feedbacks to my research and papers, and I would not have completed my Ph. D. research without their help.

Finally, I want to thank my family for their consistent support and constant love. Very special thanks to to my wife, Yoen Ju; she encouraged me to start Ph. D. study and has been supportive throughtout the years.

Flexible and Efficient Reliability in Memory Systems

Publication No. _____

Doe Hyun Yoon, Ph.D.
The University of Texas at Austin, 2011

Supervisor: Mattan Erez

Future computing platforms will increasingly demand more stringent memory resiliency mechanisms due to shrinking memory cell size, reduced error margins, higher capacity, and higher reliability expectations. Traditional mechanisms, which apply error checking and correcting (ECC) codes uniformly across all memory locations, are inefficient – Uniform protection dedicates resources to redundant information and demand higher cost for stronger protection, a fixed (worst-case based) error tolerance level, and a fixed access granularity.

The design of modern computing platforms is a multi-objective optimization, balancing performance, reliability, and many other parameters within a constrained power budget. If resiliency mechanisms consume too many resources, we lose an opportunity to improve performance. Hence, it is important and necessary to enable more efficient and flexible memory resiliency mechanisms.

This dissertation develops techniques that enable efficient, adaptive, and dynamically tunable memory resiliency mechanisms.

First, we develop two-tiered protection, apply it to the last-level cache, and present Memory Mapped ECC (MME) and ECC FIFO. Two-tiered protection provides low-cost error detection or light-weight correction in the common case read operations, while the uncommon case error correction overhead is off-loaded to main memory namespace. MME and ECC FIFO use different schemes for managing redundant information in main memory. Both achieve $15-25\%$ reduction in area and $9-18\%$ reduction in power consumption of the last-level cache, while performance is degraded by only 0.7% on average.

Then, we apply two-tiered protection to main memory and augment the virtual memory interface to dynamically adapt error tolerance levels according to user, system, and environmental needs. This mechanism, Virtualized ECC (V-ECC), improves system energy efficiency by 12% and degrades performance only by $1-2\%$ for chipkill-correct level protection. V-ECC also supports ECC in a system with no dedicated storage for redundant information.

Lastly, we propose the adaptive granularity memory system (AGMS) that allows different access granularities, while supporting ECC. By not wasting off-chip bandwidth for transferring unnecessary data, AGMS achieves higher throughput (by 44%) and power efficiency (by 46%) in a 4-core CMP system. Furthermore, AGMS will provide further gains in future systems, where off-chip bandwidth will be comparatively scarce.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Future computing platforms will demand more stringent memory resiliency mechanisms. Among the reasons for this are that soft error propensity is growing with the continuing decrease in memory cell size, that margins are shrinking for improved power efficiency, and that commercial systems are requiring continually higher RAS (reliability, availability, and serviceability) levels. Although reliability concerns are growing, we must meet the increasing needs without significantly impacting power and performance, since energy efficiency will be a key feature for future computing systems.

This dissertation explores efficient and flexible resiliency mechanisms in memory systems. The proposed mechanisms minimize the negative impacts of memory resiliency. They are also flexible, so error tolerance levels and access granularities adapt to user, system, and environmental needs.

1.1 Limitations of Current Approaches

Traditional memory protection applies *error checking and correcting* (ECC) codes uniformly across all memory locations (uniform ECC). Figure 1.1 illustrates an example memory hierarchy with uniform ECC. Additional stor-

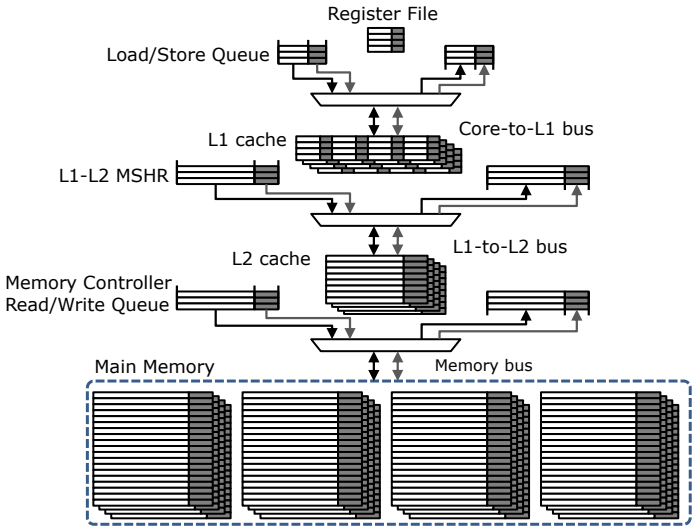


Figure 1.1: The memory hierarchy with uniform ECC (gray segments and arrows denote storage and interconnections dedicated to redundant information).

age and interconnection wires are dedicated to storing and transferring redundant information at every level; even intermediate buffers such as MSHRs (miss status handling registers) and read/write queues at the memory controller have uniform ECC codes. This mechanism is simple and transparent to the programmer; an error is detected and corrected within each memory level and is not exposed to the users unless it is uncorrectable. Uniform ECC, however, has many limitations as discussed below.

Cost of Reliability. With uniform ECC, stronger protection is enabled only with the allocation of more dedicated resources, increasing the cost of reliability. The design of a modern computing platform must trade off among performance, reliability, and many other parameters within a given power

budget. If resiliency mechanisms consume a large fraction of this given power budget, we lose an opportunity to improve performance (through devoting more resources to performance; e.g., accommodating more cores or concurrent threads or larger caches). Hence, it is important and necessary to enable stronger memory protection without significantly impacting the power and performance of a chip.

Fixed Error Tolerance Level. Another limitation of uniform ECC is that it suffers from a fixed error tolerance level; once a memory array is designed with a certain ECC code, that code protects the whole memory array, and users need to pay for the cost no matter what protection their applications demand. More importantly, this error tolerance level, and hence the cost of reliability, is determined at design time based on a "worst-case" scenario of error propensity. For example, even brand new memory chips consume unnecessary power and bandwidth by storing and transferring strong ECC codes which were designed to protect devices towards the end of their life cycles (a worst case). This "over protection" squanders power and bandwidth; hence, it is necessary to adapt error tolerance levels based on wear-out status, environmental parameters, and user demands in order to minimize the negative impacts of memory resiliency.

Recently, researchers have discussed tunable and adaptable resiliency mechanisms through cross-layer optimization: Mixed-mode multicore reliability [126] is a mechanism that enables different reliability modes based on user demands and emphasizes the need to provide flexible reliability modes; and cross-layer reliability [1] is collaborative research in academia and industry that

aims to develop scalable reliability solutions which allow adaptation in error rates. A system with adaptive reliability will allow users to pay for that error tolerance level which they need, and will allow systems to adapt to aging and environmental conditions. As a result, adaptive reliability maximizes efficiency by not wasting unnecessary power, while still guaranteeing the error tolerance level required by the users and systems. Though the discussions so far have focused on adaptive resiliency in computation, it is equally or more important to design flexible reliability mechanisms in memory systems for system-level adaptive reliability.

Fixed Access Granularity. The memory access granularity is the minimum data chunk that is read and written as a unit. Using an ECC code for the whole memory array implies that every read/write must be carried out at a fixed access granularity. Traditional wisdom is to use a coarse access granularity of 64B or larger, expecting spatial locality, such that ECC overhead is amortized over a large data block. This, however, will not lead to an optimal design any more since off-chip bandwidth is becoming the bottleneck for system efficiency. Many applications, if not all, touch only a fraction of a data block [32, 93]; hence, a fixed coarse access granularity wastes power and bandwidth for transferring unnecessary data. Intuitively, designing a memory system with a (fixed) fine access granularity can be a solution to this problem. This fine-grained-only system, however, becomes inefficient when spatial locality is high. Also, supporting ECC in such a system is a challenge since every fine-grained block requires an ECC code.

1.2 Contributions

This dissertation proposes efficient and flexible memory resiliency mechanisms and makes the following major contributions.

1. We present *two-tiered protection*, a generalization of previously proposed decoupled error detection/correction schemes [64, 71, 99]. We present detailed two-tiered mechanisms for last-level caches and main memory. The two-tiered protection mechanism minimizes the common case error detection/correction penalty and provides flexibility in choosing ECC codes.

2. We propose a mechanism to decouple data and its associated ECC by *virtualizing* redundant information. In two-tiered cache protection, we off-load the T2EC storage overhead to the main memory namespace. In main memory protection, this relaxes module design constraints in chipkill-correct level protection so that more energy-efficient configurations can be used, while providing chipkill-correct.

3. We develop main memory protection mechanisms with virtualized ECC, even for systems without dedicated ECC storage. This allows the reliable execution of mission-critical applications on low-cost or performance-oriented platforms such as GPGPUs (general purpose graphics processing units).

4. We present memory resiliency mechanisms that can adapt or tune error tolerance levels. With virtualized ECC, the same hardware can provide

different ECC codes for different memory pages according to user, system, and environmental needs. As a result, we maximize performance for non-critical applications and execute mission-critical applications reliably, all on a single platform with virtualized ECC.

5. We present a system that adaptively chooses memory access granularity with ECC support. We augment the virtual memory interface and virtualize ECC codes for fine-grained accesses so that the system allows both coarse-grained and fine-grained accesses with ECC. This utilizes finite off-chip bandwidth in a more efficient way, improving throughput and power efficiency.

1.3 Dissertation Organization

The remainder of this dissertation is organized as follows: Chapter 2 reviews background for memory resiliency mechanisms; Chapter 3 develops a set of mechanisms used for the proposed efficient and flexible memory resiliency mechanisms; Chapter 4 proposes last-level cache protection mechanisms, including Memory Mapped ECC (MME) and ECC FIFO; Chapter 5 presents flexible main memory protection mechanisms, including Virtualized ECC; Chapter 6 presents the adaptive granularity memory system (AGMS); and Chapter 7 concludes the dissertation and presents future research directions.

Chapter 2

Background

This chapter provides background for memory resiliency mechanisms. We start by briefly reviewing memory error propensity in Section 2.1. We then describe device- and circuit-level techniques for addressing soft errors in Section 2.2, information redundancy for error checking and correcting codes in Section 2.3, and current memory resiliency mechanisms with uniform ECC in Section 2.4.

2.1 Memory Error Propensity

Though many mechanisms can cause failures in memory systems, this dissertation specifically targets soft errors in SRAM and DRAM and DRAM chip failures, as they are major issues in current and future systems [100, 101]. We do not discuss hard failures that are screened by post-fabrication testing and/or fixed by redundant rows and columns. We first review soft errors due to particle strikes in Section 2.1.1, then discuss DRAM chip failures in Section 2.1.2.

2.1.1 Particle-Induced Soft Errors

A soft error is any change in the output or state of a circuit that is not permanent and can be corrected by a simple re-write, re-compute, or circuit reset operation [108]. The main cause of soft errors is charge generated by an energetic-particle strike including *alpha* particles and high-energy *neutrons*.

Since soft errors due to alpha particles and high-energy neutrons were first observed [46, 81, 142], researchers have measured *soft error rate* (SER) in memory devices. SER measure uses a unit of *failures in time* (FIT); 1 FIT is one failure per billion (10^9) hours. Based on the SER measurement over design generations reported in [108] and other literature, we summarize soft error propensity in SRAM and DRAM as below and exploit it in the proposed memory resiliency mechanisms (Chapter 3–6).

- SER in SRAM cells is only slightly decreasing, and almost unchanged over generations. It is roughly a constant, 10^{-3} FIT/bit.
- SER in SRAM devices is increasing rapidly mainly due to growing cache capacity. Systems with a 24MB cache, for example, has SER ranging $20,000$ to $200,000$ FIT (around 0.2 to 2 errors / year).
- SER in DRAM cells is decreasing and is as low as 10^{-9} FIT/bit at 40 nm technology. This is partly because DRAM cell capacitance is not scaling as design rules shrink.

- SER in large capacity main memory (up to 250GB in today's servers) is around $2,000$ to $20,000$ FIT (around 0.02 to 0.2 errors per year, $10\times$ lower than SRAM device SER).

- Multi-bit errors in the array are becoming significant; memory cells can fall under the footprint of a single energetic-particle strike as semiconductor design rules shrink [79, 90, 103], and particle strikes in the periphery circuit can cause errors in thousands of cells (a row or column).

2.1.2 DRAM Chip Failures

In addition to soft errors, main memory (commodity DRAM chips in today's computing platforms) has another failure mechanism – a memory chip failure – that is becoming increasingly important. It has been believed that soft errors are more frequent than hard errors, such as chip failures. A recent study, however, shows that DRAM behavior in the field differs from this commonly held assumption [101]. Through the measurements of memory errors in large-scale servers over 2.5 years, the authors of the paper report DRAM error rates of more than $25,000$–$75,000$ FIT per Mbit. This is significantly higher than the previously reported error rates of 200–$5,000$ FIT per Mbit. This high error rates is dominated by hard chip failures possibly due to packaging and global circuit issues.

DRAM chip failures cannot be easily tolerated by commonly used bit-error correcting codes described in Section 2.3.2 and require more stringent

protection, called *chipkill-correct* [37]. [1] Chipkill-correct is a memory resiliency mechanism that can tolerate a complete chip failure and can detect up to two-chip failures. Section 2.3.3 and Section 2.4.2 explore ECC codes and memory system organization for chipkill-correct in more detail.

In addition, soft errors in periphery logic may also require chipkill-correct level protection [106]. We list a couple of such examples as follows:

- Soft errors on an address register in DRAM chips could result in reading from or writing to a wrong address.
- If latches for reconfiguring the DRAM cell array using redundant rows and columns are affected by particle strikes, a defective row or column could be activated, appearing as hard failures. This defect remains until the chip is power cycled.
- Though rare, particle strikes on the cell array can cause errors on thousands of memory cells.

2.2 Device- and Circuit-Level Techniques for Soft Errors

Though this dissertation focuses on architecture- and microarchitecture-level mechanisms, there has also been extensive work on memory reliability using process-, layout-, and circuit-level techniques.

[1] Though not specifically discussing chipkill-correct, the discussion of this strigent memory protection mechanisms including coding techniques can be also found in [9, 24, 33, 34].

Process-Level Techniques. *Silicon On Insulator* (SOI) technology can mitigate soft errors. Researchers report that SRAM built in SOI technology shows reduced SER partly because SOI has much smaller sensitive volume for charge collection than bulk-silicon devices [18, 29, 102].

The ^{10}B isotope in *Borophosphosilicate Glass* (BPSG), which is used for inter-metal layers in device fabrication, has a large capture cross section for thermal neutrons [20]. The ^{10}B nucleus is unstable when exposed to thermal neutrons; upon absorbing a thermal neutron, the ^{10}B nucleus breaks apart, or "fissions", with an accompanying release of energy in the form of an excited ^{7}Li recoil nucleus, a gamma photon, and an alpha particle. Hence, BPSG in logic and SRAM processes are largely replaced with chemical polishing techniques to reduce SER [106].

The least intrusive process level method is deep N-well technology, which uses a triple-well process in which NMOS devices are constructed inside P-wells, which are themselves inside deep N-wells [39]. The deep N-wells provide isolation from alpha particles and neutron cosmic rays and reduce the failure rate of SRAMs by a factor of approximately $1.5 - 2.5$.

Layout-Level Techniques. Modifying the SRAM cell layout can increase Q_{crit}(critical charge that is required to flip a stored bit), reducing soft error rates. This is done by adding poly-diffusion overlaps to the critical nodes [39], improving SER by a factor of 20, but at a 40% area overhead and a penalty of $6-8\%$ in latency. Another technique is to add a metal-insulator-metal (MIM) node capacitor [58, 96], which can impact write cycles by around 20 ps/fF [58].

Circuit-Level Techniques. Hardened SRAM cells that use a larger number of transistors have been suggested to achieve a reduction in soft-error propensity and a more robust operation even at low V_{CC} values. For example, 8T cells [31], 10T cells [27], and Schmidt Trigger (ST) based 10T cells [66] can significantly improve reliability, but at the cost of increased access latency and SRAM cell area (8T - 30%, 10T - 60%, and ST 10T - 100% [127]).

To tolerate multi-bit burst errors, physical bit interleaving [79, 92] is commonly used in both SRAM and DRAM. By interleaving bits from adjacent data blocks in the physical layout, a multi-bit error caused by a single upset appears as single-bit errors in multiple data blocks rather than a single block with a multi-bit error. Physical interleaving, however, is suitable only for small-scale multi-bit errors. Scaling physical interleaving to higher potential error counts (beyond 4-way interleaving) results in large performance, area, and power overheads [63].

2.3 Information Redundancy

A common solution to tolerate memory errors is to apply error checking and correcting (ECC) codes. ECC codes exploit *information redundancy*, which uses extra bits of information to detect and potentially correct errors. For instance, an 8-bit data block can represent an unsigned number that ranges between 0 and 255. With information redundancy, we encode this number using a 9-bit *codeword*. The 9-bit codeword has 512 different cases: Only 256 cases are *valid*, and the other cases are *invalid*. As an occurrence of some er-

rors turns a valid codeword to an invalid codeword, we can detect (and correct in more powerful coding schemes) the error. Although there are a variety of ECC codes, we describe only those widely used for memory protection.

Nomenclature. A bit-error correcting code, which is described in Section 2.3.2, is composed of *data bits* and *check bits*, where check bits are redundant bits added to data bits for the purpose of error detection or correction. We use the notation (n, k) code, where n is codeword length in bits, and k is data length in bits. For a symbol-error correcting code, which is described in Section 2.3.3, we use *data symbols* and *check symbols* similar to the data bits and check bits used for bit-error correcting codes. The notation for symbol-error correcting codes is (N, K, b), where N is the number of total symbols, K is the number of data symbols, and b is the symbol width in bits.

2.3.1 Parity

Parity is a simple error detecting code that adds 1 parity bit to a data block. There are two types of parity codes: even parity and odd parity. The parity bit indicates whether the number of ones in a codeword is even (in even parity) or odd (in odd parity). Parity can detect any error, including an error in the parity bit itself, so long as the error corrupts an odd number of bits in the codeword. For its simplicity, parity is widely used for error detection in low-latency memory structures such as L1 caches and register files.

2.3.2 Bit-Error Correcting Codes

SEC-DED. Parity can detect, but cannot correct errors. The most common error correcting codes include Hamming [50] [2] and Hsiao [53] codes, which provide *single bit-error correction and double bit-error detection* (SEC-DED). Hsiao codes are also known as *odd-weight-column* codes and provide improvements over Hamming codes in terms of speed, cost, and reliability of decoding logic [34]. An SEC-DED code requires $1 + \lceil \log_2(k+1) \rceil$ check bits for a k-bit data block.

DEC-TED. When a higher reliability level is necessary, *double bit-error correcting and triple bit-error detecting* (DEC-TED) codes are used. DEC-TED codes, however, require a larger number of check bits than SEC-DED codes and use more complex hardware for error detection and correction. The storage overhead of a DEC-TED code is $1 + 2 \times (\lceil \log_2(k+1) \rceil)$ bits for a k-bit data block, and it is almost twice the overhead of SEC-DED codes.

BCH. Both SEC-DED and DEC-TED codes are a special case of BCH (Bose-Chaudhuri-Hocquenghem) codes [23, 52]. A t-bit correcting BCH code requires $t \times \lceil \log_2(k+1) \rceil$ check bits for a k-bit data block and can detect $t+1$ bit errors with 1 additional bit. Due to the area and latency overheads of BCH codes, which are approximately proportional to t [113], BCH codes with more than 2-bit correction are, in general, not used for memory protection.

[2] The Hamming code originally proposed in 1950 was a *single bit-error correcting* (SEC) code, but Hamming code is often used with an additional parity bit that extends it to SEC-DED.

Table 2.1: Redundancy and latency overheads of bit-error correcting codes. FO4 stands for Fan-Out of 4 delay.

	SEC-DED			**DEC-TED**			**6EC-7ED**		
Data	check	latency (FO4)		check	latency (FO4)		check	latency (FO4)	
bits	bits	encoder	decoder	bits	encoder	decoder	bits	encoder	decoder
16	6	9	45.1	11	9	81.1	31	10.8	259.8
32	7	10.8	48.3	13	10.8	85.5	37	12.6	265.6
64	8	12.6	50.7	15	12.6	88.7	43	12.6	265.6
128	9	14.4	53.2	17	14.4	91.8	49	14.4	271.4
256	10	16.2	65.3	19	16.2	111.8	55	16.2	322.9
512	11	18	68.5	21	18	116.3	61	18	332.7

ECC Storage and Complexity Overheads. Table 2.1 compares the redundancy and latency overheads of bit-error correcting codes. We compare commonly used SEC-DED and DEC-TED codes with a many-bit correcting code, 6EC-7ED. 6EC-7ED is 6 bit-error correction and 7 bit-error detection (BCH with t=6). As shown in Table 2.1, the relative check bit overhead decreases as data size increases.

To estimate BCH coding latencies, we use the BCH decoder model developed by Strukov [113]. [3] We separately report encoder latency and decoder (including error detection and error correction) latency to show the asymmetry in the complexity of encoding and decoding. An encoder implements a set of XOR trees, and a decoder uses the same XOR trees for syndrome calculation that detects errors. So, the complexity of detection-only logic is comparable to that of an encoder. Note that the latency of ECC encoding (and error de-

[3] We estimate the latency of SEC-DED decoding using a generic BCH decoder model with $t = 1$. An SEC-DED decoder, however, can be implemented in a more compact way, since the complex steps after syndrome calculation can be simplified.

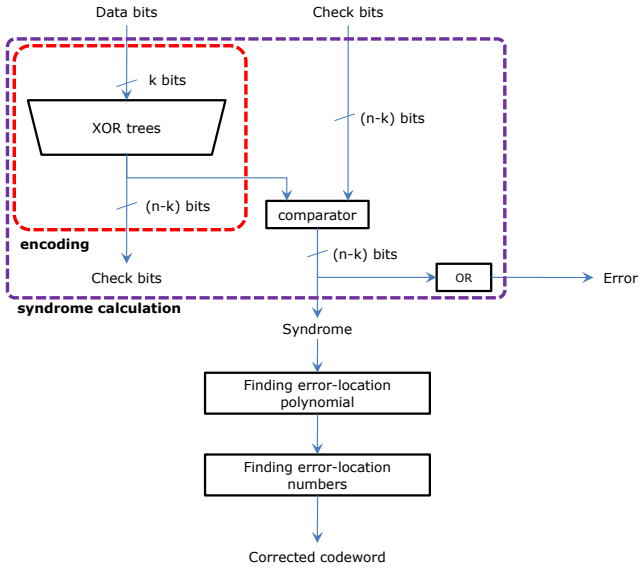


Figure 2.1: A block diagram of a simplified BCH encoder/decoder.

tection) is relatively small (below 20 FO4) even with strong ECC codes such as 6EC-7ED.

The whole decoder operation, however, is much more complex than encoding or error detection; decoding is composed of three major steps [22] including syndrome calculation, finding an error-location polynomial, and finding error-location numbers, where the latter two operations involve relatively complex operations.

Figure 2.1 illustrates a simplified block diagram of a BCH encoder/decoder; this shows how encoding and error detecting share XOR trees also. At low error rates (as discussed in Section 2.1), we can implement a decoder with only error detection; the complex steps for error correction can be invoked only when an error occurs.

2.3.3 Symbol-Error Correcting Codes

Modern commodity DRAM chips are designed with a b-bit data path (4-bit, 8-bit, or 16-bit devices). When memory chips are configured with b bits per chip, the failures described in Section 2.1.2 generate b-bit error patterns (b-bit symbol errors) [33].

Interleaved Bit-Error Correcting Codes. We can detect and correct symbol errors by logically interleaving multiple instances of bit-error correcting codes described in Section 2.3.2. For instance, 4-way interleaved $(72, 64)$ SEC-DED codes can provide one 4-bit symbol-error correction and two 4-bit symbol-error detection. This, however, requires a 288-bit ($4 \times$ 72 bits) wide channel, which increases the minimum memory access granularity (to 256B in DDR3). Note that we can also use interleaved bit-error correcting codes for bursty errors in caches; interleaving SEC-DED codes has much lower complexity than many-bit correcting BCH codes.

Symbol-Based ECC Codes. Compared to bit-error correcting codes, symbol-based error correcting codes derived from Reed-Solomon (RS) codes [94] are inherently more effective for off-chip memory protection. Although there are many different symbol-error correcting codes, we specifically discuss error codes for chipkill-correct [37].

ECC Codes for Chipkill-Correct. ECC codes for chipkill-correct require *single symbol-error correction and double symbol-error detection* (SSC-DSD) capability so that data is correctly read from and written to a memory module

even if a DRAM chip in it completely fails, and that up to two failing chips are detected. We explain two types of SSC-DSD codes: a 3-check-symbol code and a 4-check-symbol code.

A 3-check-symbol code provides SSC-DSD with maximum coding efficiency [33]. The codeword length of the 3-check-symbol code is, however, limited to $2^b + 2$ symbols so it is a poor match for $\times 4$ DRAM chips – three 4-bit check symbols (12 bits) protect 15 4-bit data symbols (60 bits), which is a non power of two.

Instead, a 4-check-symbol code [34] is more practical for 4-bit symbols; the 4^{th} check symbol allows a longer codeword. We refer to this 4-check-symbol code as a *single nibble-error correcting and double nibble-error detecting* (SNC-DND) code [33]; it is composed of four 4-bit check symbols (16 bits) and 32 4-bit data symbols (128 bits), yielding 12.5% redundancy overhead. The SNC-DND code is widely used in many commercial designs [15, 116] that implement chipkill-correct.

Symbol-Based ECC Complexity. Error correction in symbol-based ECC is composed of syndrome calculation and more steps to identify error-symbol locations as well as error-symbol values. SSC-DSD codes, however, can simplify the correction procedure after syndrome calculation, since we only need to find a single error symbol [34]. Similar to bit-error correcting codes, we can implement error-detecting logic on the critical path and only invoke the complex error correction steps in the rare event that errors are actually detected. Note that syndrome calculation, as well as an ECC encoding procedure,

requires only XOR trees, not *Galois Field* (GF) multiplications [74]. Furthermore, symbol-error correcting codes are encoded and decoded at the memory controller such that the additional latency for ECC encoding/decoding is relatively small compared to the long DRAM access latency.

2.4 Uniform ECC Based Memory Protection

This section describes the uniform ECC based memory resiliency mechanisms used in current architectures. Since L1 and L2 caches and DRAM have different requirements and access properties, different ECC codes are used for L1 and L2 caches and DRAM. Section 2.4.1 describes resiliency mechanisms in caches, and Section 2.4.2 discusses main memory protection mechanisms including chipkill-correct.

2.4.1 Caches

In cache memory, different error codes are used based on cache levels and write-policy (write-through or write-back).

Write-Through L1. If the first-level cache (L1) is write-through (WT) and the LLC (Last Level Cache; for example, L2 cache) is inclusive, it is sufficient to provide only error detection (using 1-bit parity per word) on the L1 data array because the data is replicated in L2. Then, if an error is detected in L1, error correction is done by invalidating the erroneous L1 cache line and re-fetching the cache line from L2. Such an approach is used in the SUN UltraSPARC-T2 [116] and IBM Power 4 [118] processors. The L2 cache is protected by

ECC, and because L1 is write-through, the granularity of updating the ECC in L2 must be as small as a single word. For instance, the UltraSPARC-T2 uses a 7-bit SEC-DED code for every 32 bits of data in L2, an ECC overhead of 22%.

Write-Back L1. If the L1 cache is write-back (WB), both the L1 and L2 caches need error correcting codes. L1 caches typically use an SEC-DED code per word. L2 accesses are, however, at the granularity of a full L1 cache line; hence, the granularity of ECC can be much larger, reducing ECC overhead. The Intel Itanium processor, for example, uses a 10-bit SEC-DED code that protects 256 bits of data [130] with an ECC overhead of only 5%. Other processors, however, use a smaller ECC granularity even with L1 write-back caches to provide higher error correction capabilities. The AMD Athlon [55] and Opteron [61] processors, as well as the DEC Alpha 21264 [41], interleave eight 8-bit SEC-DED codes for every 64B cache line to tolerate more errors per line at a cost of 12.5% additional overhead.

2.4.2 Main Memory

In this subsection, we first briefly review memory system organization, from DRAM chips to memory modules, since they are closely related to memory error protection, and then proceed to main memory error protection including chipkill-correct.

2.4.2.1 Modern Memory Systems

Modern computers hide the complexity involved in designing memory systems from the programmer. Virtual memory abstracts nearly all memory system details, including resource allocation and virtual to physical mapping, and provides the illusion of a flat and uniform address space to software. Physical memory is, however, composed of memory channels, each with multiple memory modules.

An individual DRAM chip has address/command input pins, bi-directional data pins, as well as data storage. A single DRAM chip has a narrow external data path, typically 4, 8, or 16 bits wide (referred to as $\times 4$, $\times 8$, or $\times 16$, respectively), and multiple chips operate together to form a wide data path, called a *rank*. For example, a 64-bit wide rank is composed of 16 $\times 4$ DRAMs, 8 $\times 8$ DRAMs, or 4 $\times 16$ DRAMs. A rank is the minimum logical device that a memory controller can control individually; hence, all DRAM chips in a rank are addressed simultaneously.

A memory module, or a DIMM (dual in-line memory module), is a physical device that has, typically, 1 to 8 ranks; a standard 64-bit wide DIMM is also referred to as a Non-ECC DIMM. DIMMs (also ranks) in a memory channel share the physical address/command and data buses, but only one rank is addressed at any given time to avoid bus conflicts. Depending on the type of DRAM chips used, DIMMs are classified into $\times 4$ DIMMs, $\times 8$ DIMMs, and $\times 16$ DIMMs. If the total capacity is the same, a DIMM with wider DRAMs ($\times 8$ or $\times 16$) has fewer DRAM chips and consumes less power [17]; hence, $\times 8$

or ×16 DIMMs are preferred. Systems that require high reliability/availability, however, favor ×4 DIMMs, especially systems with chipkill-correct. This is, in part, due to the constraints in symbol-error correcting codes as discussed in Section 2.3.3.

2.4.2.2 Memory Error Protection

Main memory error protection uses DRAM modules that can store redundant information and apply ECC to detect and correct errors. This ECC DIMM requires a larger number of DRAM chips and I/O pins than a Non-ECC DIMM.

SEC-DED for Main Memory. Typically, an ECC DIMM is used to provide SEC-DED for each DRAM rank without impacting memory system performance. SEC-DED codes [50, 53] use 8 bits of ECC to protect 64 bits of data. To do so, an ECC DIMM with a 72-bit wide data path is used, where the additional DRAM chips store the redundant information. An ECC DIMM is constructed using 18 ×4 chips (×4 ECC DIMM) or 9 ×8 chips (×8 ECC DIMM), but there is no ×16 ECC DIMM. Note that an ECC DIMM only provides additional storage for redundant information, but that actual error detection/correction takes place at the memory controller, yielding the decision of protection mechanisms to system designers.

Chipkill-Correct. As discussed in Section 2.1.2, it is important to tolerate chip failures, especially in large-scale systems for high availability and reliability. Hence, business critical servers and datacenters demand chipkill-correct

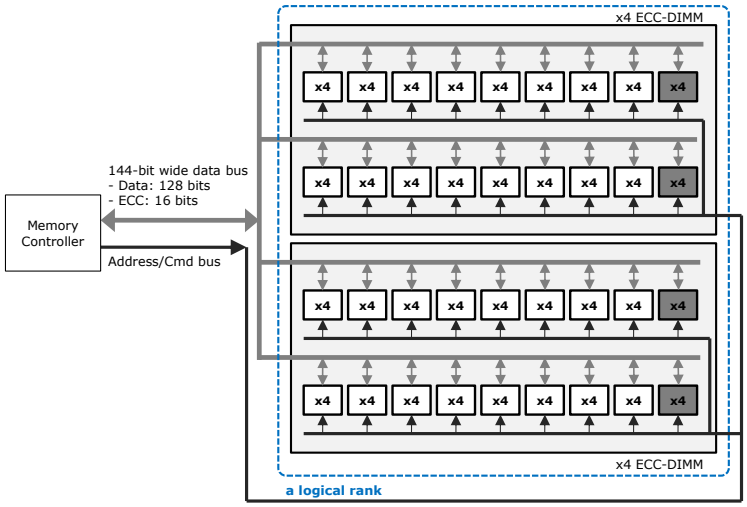


Figure 2.2: Baseline chipkill correct DRAM configuration (gray DRAMs are dedicated to ECC storage).

level reliability, where a DIMM is required to function even when an entire chip in it fails. Chipkill-correct "spreads" a DRAM access across multiple chips and uses a wide ECC code to allow strong error tolerance [15, 37, 116]. The error code for chipkill-correct is a SSC-DSD code; the SNC-DND code explained in Section 2.3.3 is widely used.

The SNC-DND code results in a 144-bit wide data path (128 bits for data and 16 bits for redundant information); this wide data path is implemented using two ×4 ECC DIMMs in parallel as shown in Figure 2.2. This organization is used by the Sun UltraSPARC-T1/T2 [116] and the AMD Opteron [15].

The downside of chipkill-correct is that it either increases a memory access granularity, requiring more energy and restricting possible DRAM config-

urations, or increases the required level of redundancy, which, again, increases cost [13]. For instance, the chipkill-correct memory system shown in Figure 2.2 works well with DDR2 using minimum burst of 4; the minimum access granularity is 64B (4 transfers of 128bits). It is, however, problematic with DDR3 or future memory systems; longer burst combined with a wide data path for chipkill-correct leads to a larger access granularity [13].

The chipkill-correct level protection is, unfortunately, needed in systems that are increasingly sensitive to energy and cost, such as large-scale servers that demand high availability and protection guarantees. For example, large installations have reported that system outages due to DRAM errors are 600 times higher if chipkill is not used [56]. Providing chipkill protection in current DRAM packaging technology is expensive and requires the use of $\times 4$ DRAM configuration. These narrow chips consume roughly 30% more energy for a given total DIMM capacity as more efficient $\times 8$ configurations [17]. This extra overhead is added to all the memory in these large capacity systems, which may be multiple tera bytes if a memory extension appliance is used [73, 122].

Wider DRAMs ($\times 8$ and $\times 16$) can use the 3-check-symbol code, since the maximum codeword length, $2^b + 2$ symbols, increases with 8- or 16-bit symbols. Supporting chipkill-correct with $\times 8$ and $\times 16$, however, is impractical. First of all, chipkill-correct using $\times 16$ DRAMs is not possible unless we design a custom $\times 16$ ECC DIMM, which is expensive. Even for $\times 8$ DRAMs, it requires trading off storage overhead with DRAM access granularity, which may lower performance. Maintaining the same access granularity for a $\times 8$

configuration increases the fraction of redundant data to at least 18.5% [13] (128-bit data and 24-bit ECC), and it also requires a custom ×8 ECC DIMM having 16 ×8 DRAMs for data and three ×8 DRAMs for ECC, that then increases cost. Maintaining the same (or less) 12.5% ECC storage overhead, on the other hand, doubles the DRAM burst size (256-bit data and 24-bit ECC). Note that burst 4 in DDR2 increases the access granularity to 128B for a 256-bit data path and that DDR3 with burst 8 makes it 256B.

Other Chip-Failure Tolerating Schemes. Some designs use a relaxed protection level for tolerating chip failures. Such systems use a single symbol-error correct (SSC) RS code and erasures (i.e., errors with known locations) [87, 88]. A chip failure is detected and corrected using an SSC code, after which the memory controller remembers the location of the chip failure so that the SSC code can correct the chip failure (an erasure) and can detect a second chip failure. This scheme, however, cannot detect two simultaneous chip failures. In this dissertation, we only consider aforementioned chipkill-correct that can correct a chip failure and detect a two-chip failure.

Chapter 3

Mechanisms for Efficient and Flexible Reliability

In this chapter, we develop a set of mechanisms for the proposed memory resiliency described in Chapter 4–6. We first introduce *two-tiered protection* in Section 3.1, then describe *flexible* reliability through *virtualizing* redundant information within the memory namespace in Section 3.2. The proposed cache protection (Chapter 4) and memory protection (Chapter 5) leverage two-tiered protection as well as ECC storage virtualization. Section 3.3 explains how to manage *fine-grained* data in caches and DRAM; the adaptive granularity memory system (Chapter 6) uses the fine-grained data management techniques as well as ECC storage virtualization.

3.1 Two-Tiered Protection

As an alternative to uniform ECC, we develop two-tiered protection in this section. Two-tiered protection is a generalization of previously suggested decoupled detection/correction approaches [64, 71, 99]. We base two-tiered protection on the following observations:

- While memory errors cannot be ignored at any memory hierarchy level, error events are still expected to be extremely rare as discussed in Section 2.1 when compared to the processor cycle time. For instance, the mean time to failure (MTTF) of a 32MB cache is around 155 days assuming 10^{-3} FIT/bit.

- Every read requires error detection to ensure that no error has occurred. Error detection latency, therefore, is critical to application performance, and error detection energy is important to overall system power.

- Every write is accompanied by computing information required for error correction as well as for detection. The latency of this operation, however, is unimportant as long as required write throughput can be supported. Additionally, write operations are typically less common than read operations.

- Given that error rate is low, the latency and the complexity of error correction is not important. Even an error correction latency of 10 ms is acceptable, since it would occur once every several weeks or months on a given processor.

- As discussed in Section 2.3, the complexity of ECC-encoding or detecting-only logic is relatively simple even with very strong ECC codes.

- Much of the data stored in caches and main memory is replicated through the memory hierarchy (clean cache lines or clean data pages). Replicated

data is inherently error tolerant, because correct values can be restored from a copy. Thus, it is enough to detect errors in clean data, which can be done with lower cost than error correction.

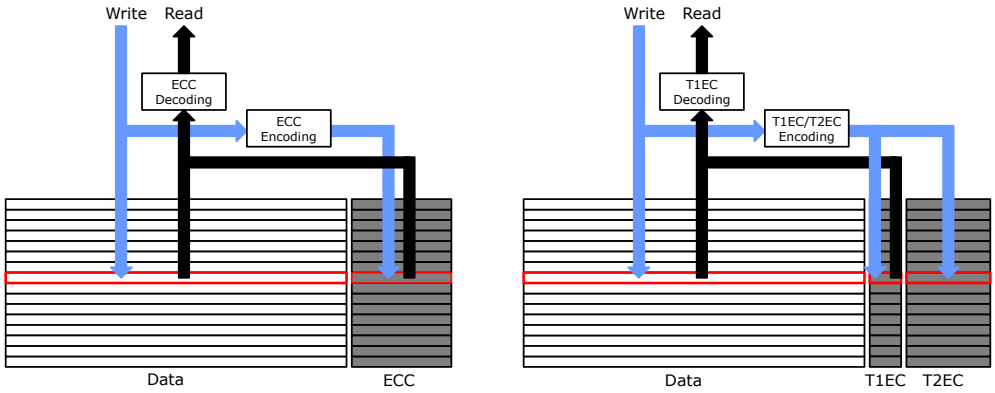
In the rest of this section, we review the previously suggested decoupled detection/correction approaches in Section 3.1.1, then develop two-tiered protection in Section 3.1.2–3.1.4.

3.1.1 Decoupled Detection/Correction

The Punctured ECC Recovery Cache (PERC) [99] uses parity for error detection and single bit-error correcting (SEC) code for error correction. PERC applies this parity-SEC code combination to L1 caches; a read operation accesses only data and parity to save energy and improve latency, and a write operation updates data, parity, and an SEC code. The same parity-SEC mechanism is also used to save static energy consumption [71] and reduce cache area [64] (we will revisit these cache reliability studies later in Section 4).

3.1.2 Two-Tiered Protection

We generalize the simple parity-SEC combination used in the decoupled detection/correction approaches and develop two-tiered protection. Two-tiered protection consists of a *tier-1 error code* (T1EC) and a *tier-2 error code* (T2EC): The T1EC is an ECC code for the common case read operations and provides error detection or light-weight error correction; and the T2EC is a



(a) Uniform ECC (b) Two-tiered protection

Figure 3.1: Uniform ECC and two-tiered protection.

strong error-correcting code that is used for error correction on rare T1EC DUE (detected, but uncorrectable error) events.

Figure 3.1 compares uniform ECC and two-tiered protection using simplified memory array organizations of data and ECC codes. Uniform ECC extends each data line with an ECC code that detects and corrects errors as shown in Figure 3.1(a). In addition to area and leakage power increases, this consumes more dynamic power to read and write redundant information as well as data. A memory array with T1EC and T2EC is shown in Figure 3.1(b). In two-tiered protection, the mechanisms for detecting errors (with correcting light-weight errors also) and correcting errors are essentially split as explained.

Virtualized T2EC. The T2EC, however, is computed only on a write into the memory array, because errors on clean data lines can be recovered from a different level of the memory hierarchy. Just as importantly, the T2EC is only read and decoded on a rare event that the T1EC detects an error that

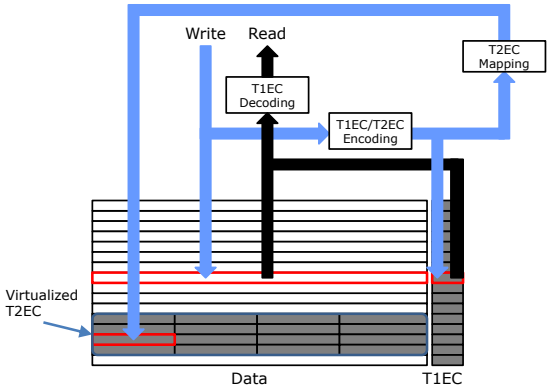


Figure 3.2: Two-tiered protection with virtualized T2EC.

it cannot correct (recall that it is possible to architect the T1EC with light-weight error correction). Therefore, the T2EC does not need dedicated storage and can be stored within the memory namespace as shown in Figure 3.2. This technique can be generalized as *virtualizing* redundant information, which we discuss further in Section 3.2.

Storing T2EC within the memory namespace increases write latency and traffic, since a data write is split into two operations: to write data and T1EC and to write memory-mapped T2EC as data. In order to mitigate latency and traffic increases, many existing microarchitectural solutions such as caching or write coalescing can be combined with two-tiered protection. We discuss the detailed mechanisms in Chapter 4–5.

3.1.3 Advantages of Two-Tiered Protection

Two-tiered protection has many advantages over uniform ECC. First, the T1EC minimizes the common case penalty for memory resiliency. The

T1EC, by its definition, is simpler than an ECC code at the same error tolerance level, reducing storage, latency, and energy overheads of read operations. Second, storing the T2EC within the memory namespace eliminates all static costs (additional area, interconnection wires, and leakage power) of the dedicated T2EC.

Two-tiered protection also enables stronger protection at low cost. As mentioned earlier, uniform ECC requires more dedicated resources in order to meet the stronger protection demands of future systems. The additional overheads are added to the already tight bandwidth and power budgets. The two-tiered protection mechanism, however, makes it simple to accommodate stronger ECC codes; the cost of T1EC with strong error-detecting (and lightweight error-correcting) capability can be even lower than that of traditional ECC codes, and the memory-mapped T2EC provides strong reliability guarantees, but at no dedicated storage for it. Note that the ECC-encoding or detecting-only logic has relatively low complexity even with very strong ECC codes as discussed in Section 2.3 and that the complex error-correcting operation using T2EC is invoked only upon a rare T1EC DUE event.

Another advantage with the two-tiered approach is the flexibility in choosing and adapting the error tolerance level based on dynamic application, user, and system needs. Section 3.2 further discusses the flexible reliability with virtualized redundant information.

3.1.4 Design Considerations for Two-Tiered Codes

We list the design considerations for the T1EC and T2EC as follows.

- The T1EC should be chosen to minimize dedicated storage overhead as it is stored along with every data line.
- The T1EC should be computed with low latency and maximize error detecting capability. Even though the strong T2EC can correct errors, it should be first detected by the T1EC.
- The T2EC determines overall protection capabilities, but large T2ECs increase traffic to update virtualized T2EC information, which degrades the performance of some applications.

While we can design two-tiered codes based on the above design considerations, caches and DRAM have different requirements and access properties; hence, we defer the discussion of the actual two-tiered code examples in Chapter 4 (for caches) and Chapter 5 (for DRAM).

3.2 Flexible Reliability through ECC Storage Virtualization

The second mechanism we develop is *virtualizing* redundant information within the memory namespace. We already discussed storing T2EC within the memory namespace in Section 3.1 and further generalize this concept to support even one-tier approaches in Section 3.2.1. Then, we describe how

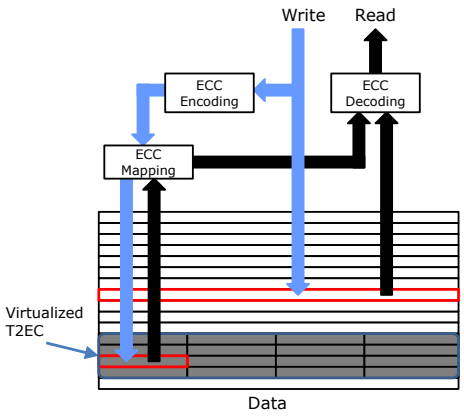


Figure 3.3: One-tiered protection with virtualized ECC.

to enable *flexible* memory resiliency with ECC storage virtualization in Section 3.2.2.

3.2.1 Memory Protection with ECC Storage Virtualization

In two-tiered protection, we store the T2EC within the memory namespace; hence, we dedicate resources only to the T1EC, reducing the hardware overheads, while guaranteeing the strong protection with the virtualized T2EC. We can further extend the idea of virtualizing redundant information to support traditional one-tiered protection, that then completely eliminates dedicated storage for ECC.

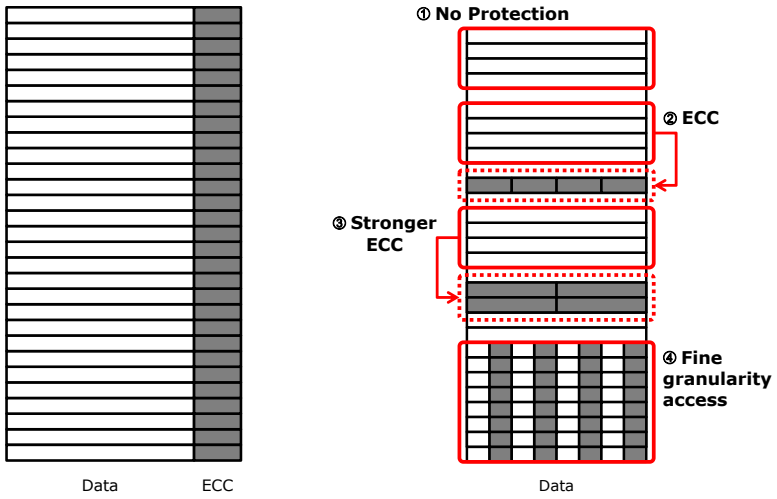
Memory Protection with No Dedicated ECC Storage. Figure 3.3 illustrates a memory array with virtualized one-tiered ECC. Error detection and correction operations in this architecture are same to those in uniform ECC, but the redundant information is stored within the memory namespace, instead of storing ECC to dedicated storage. As a result, every memory op-

eration (even a read) becomes a two-step operation: to access data and to access ECC. This increases latency as well as traffic, but we can leverage microarchitectural techniques such as caching or write coalescing (more details in Chapter 5). Note that we can send requests to fetch data and the associated ECC in parallel to mitigate the latency increase.

Unlike two-tiered approach, this architecture also impacts read latency that is more critical to application performance. So, the one-tiered approach with virtualized ECC does not look attractive at the moment. We can, however, selectively apply this technique only to those applications that need high reliability levels in a system without dedicated ECC storage. We discuss more on this topic in Chapter 5.

3.2.2 Flexible Reliability with ECC Storage Virtualization

So far, we focused on how to avoid dedicated storage for redundant information through virtualizing ECC (either two-tiered or one-tiered). ECC storage virtualization has a more important advantage; it enables flexible reliability so that single hardware can concurrently support different protection schemes based on dynamic application, user, and system demands. For example, critical applications can run with maximum reliability guarantees using stronger (virtualized) ECC, while the performance of non-critical applications can be maximized by turning off ECC protection. Also, a system can adapt error tolerance levels to device wear-out status or environmental parameters.



(a) Uniform ECC (b) Virtualized ECC

Figure 3.4: Flexible reliability through virtualized redundant information.

Figure 3.4 illustrates how virtualizing redundant information enables flexible reliability compared to traditional uniform ECC. Uniform ECC, as shown in Figure 3.4(a), forces a fixed error tolerance level and a fixed access granularity; hence, a system designer must choose an error tolerance level based on worst-case error propensity and an access granularity for average applications. Figure 3.4(b) shows a system with virtualized ECC. In this system, single hardware can have different protection schemes: (i) to maximize performance by not using ECC; (ii) to enable ECC for important data; (iii) to use stronger ECC for mission-critical applications; and (iv) to apply ECC at a finer access granularity for application with low spatial locality. In Chapter 5–6, we leverage this technique, virtualizing redundant information, for flexible reliability and provide the detailed mechanisms, tradeoffs, and evaluation.

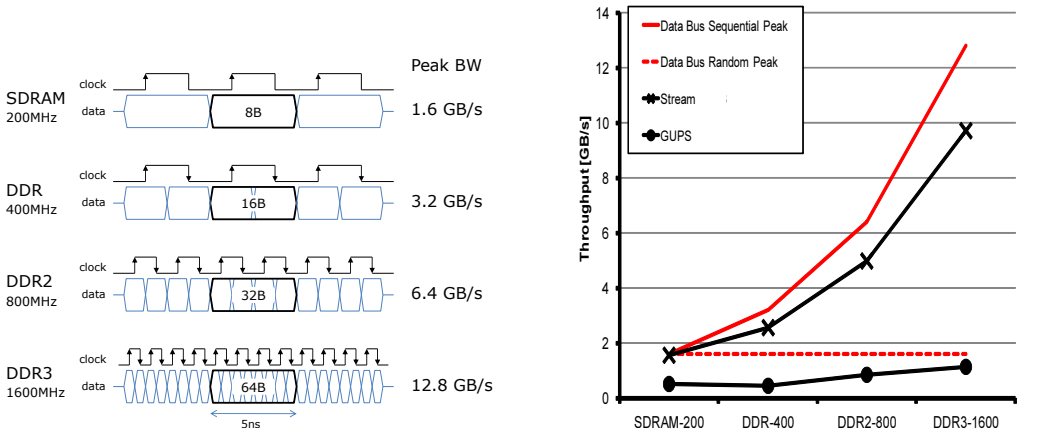
3.3 Fine-Grained Data Management in the Memory Hierarchy

In Chapter 6, we describe the adaptive granularity memory system that utilizes flexible reliability through virtualizing redundant information (described in the previous section) and the ability to manage fine-grained data in the memory hierarchy. Modern memory systems, however, are optimized for coarse-grained accesses; they rely on spatial locality to reduce off-chip accesses and miss rate and to lower control and storage overheads.

In this section, we describe mechanisms that enable fine-grained data accesses in modern memory systems including DRAM and caches. Section 3.3.1 describes a *sub-ranked* DRAM system that enables fine-grained memory accesses, and Section 3.3.2 reviews a *sector* cache, an old design paradigm that allows partially-valid cache lines.

3.3.1 Sub-ranked Memory Systems

Modern memory systems do not provide truly-uniform random access. They are instead optimized for capacity first and for high bandwidth for sequential access second. In order to keep costs low, the DDRx interface relies on high spatial locality and is requiring an ever increasing minimum access granularity. Regardless of how much data is required by the processor, the off-chip DRAM system returns a certain minimum amount, often referred to as a *DRAM burst*. Therefore, in DDRx memory systems, the overall *minimum access granularity* is a product of the DRAM chip minimum burst length and



(a) Timing

(b) Effective throughput (useful data only)

Figure 3.5: Timing diagrams and effective throughput of several DRAM generations.[1]

the data width of a channel. For example, a 64-bit wide DDR3 DRAM channel with a burst of 8 accesses, which is typical in contemporary DDR3 DRAM modules, has a minimum access granularity of 64B. Therefore, even if the processor needs only a single word, the entire 64B chunk around it has to be read from DRAM and transferred to the memory controller.

Module and Interface Tradeoffs. Figure 3.5(a) shows simplified timing diagrams of data bus usage for several DRAM generations: single data rate SDRAM, DDR, DDR2, and DDR3. While the high density required from DRAM practically limits its core clock frequency to 200MHz, effective I/O data rates have increased up to 1600MHz for the latest DDR3. Note that newer DRAM generations transfer larger chunk of data in the same time win-

[1]Note that there are no 200MHz SDRAM products, but we present it here for comparison.

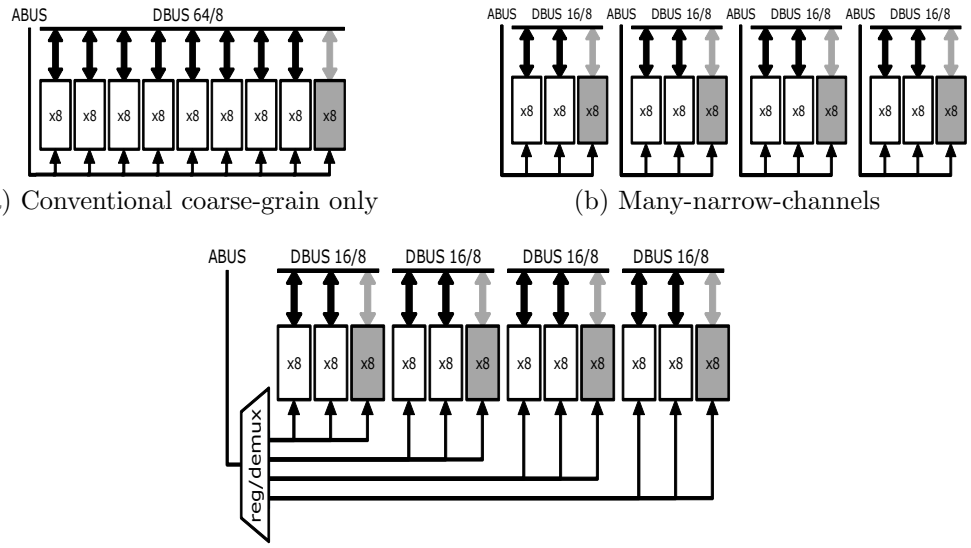
dow (5ns in this example), doubling the peak bandwidth in each generation. This increase in bandwidth is achieved by employing n-bit prefetch and a burst access: n is 2 in DDR, 4 in DDR2, and 8 in DDR3. As a result, the minimum access granularity is increasing: 8B in SDRAM, 16B in DDR, 32B in DDR2, and 64B in DDR3 with a typical 64-bit wide channel. Thus, as DRAM technology advances, the relative cost of fine-grained accesses increases.

We carried out a simple experiment that compares the effective DRAM throughput of sequential accesses and fine-grained random accesses using DRAMsim [123]. We used `STREAM` and `GUPS` of the HPC Challenge Benchmarks [2] to represent sequential and random access patterns, respectively. As Figure 3.5(b) shows, near-peak bandwidth is easily achieved with sequential accesses (in `STREAM`), whereas fine-grained random accesses (in `GUPS`) have poor performance that barely improves over DRAM generations. Note that we provisioned enough banks and ranks (8 banks per rank and 8 ranks) so that bank-level parallelism can hide the penalty of the frequent bank conflicts of random accesses. Hence, the relatively low throughput of `GUPS` in DDR2 and DDR3 is primarily due to their large minimum access granularity. Only a fraction of the data transferred is actually used, lowering effective throughput.

A narrower data path can be used to decrease the minimum access granularity because the granularity is a product of minimum burst length and channel width. The burst length is dictated by the DRAM technology, but the channel width is a system design parameter. We can find such implementations in vector processors like the Cray Black Widow [11]: Its memory subsystem

has many 32-bit wide DDR2 DRAM channels, providing 16B minimum access granularity. We refer to this style of memory subsystem as *many-narrow-channels*. Although the many-narrow-channels approach reduces minimum access granularity, such a design is inherently very expensive. The many-narrow-channels approach increases the control overhead, since each channel requires its own address/command pins. Compared to conventional memory systems with one address/command bus for every 64-bit data path, Black Widow's memory subsystem needs twice the number of address/command pins. This is a critical problem, since future systems will be severely pin limited.

An alternative approach to this supercomputing-based many-narrow-channels design is to use DRAM sub-ranking. Recently, there have been multiple proposals for memory systems that can control individual DRAM devices within a rank: Rambus's micro-threading [124] and threaded memory module [8, 125]; HP's MC-DIMM (multi-core dual in-line memory module) [13, 14]; Mini-rank memory system [140]; and Convey's S/G DIMM (scatter/gather dual in-line memory module) [25]. In this dissertation, we collectively refer to these techniques as *sub-ranked* memory systems. Figure 3.6 compares a conventional coarse-grain-only memory system, a many-narrow-channels approach, and a sub-ranked memory system similar to MC-DIMM. In Figure 3.6, the many-narrow-channels and sub-ranked approaches provide 16B access granularity with DDR3, and gray boxes and arrows represent ECC storage and transfers.



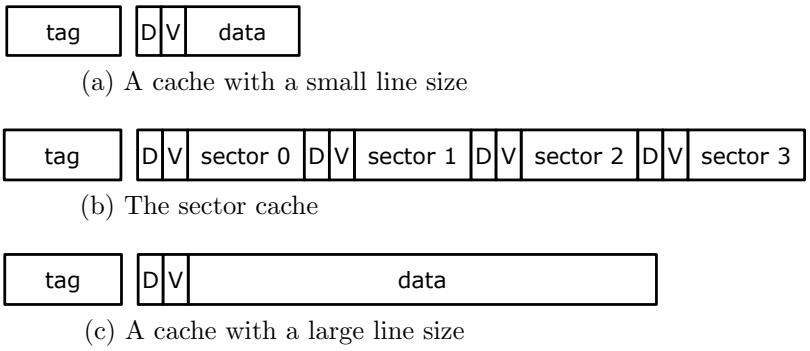
(a) Conventional coarse-grain only

(b) Many-narrow-channels

(c) Sub-ranked

Figure 3.6: Comparison of memory systems. *ABUS* represents address/command bus, and *DBUS X/Y* represents data bus, where X bits are for data and Y bits are for ECC.

Most sub-ranked memory proposals [13, 14, 125, 140] focus on energy efficiency of coarse-grained accesses by mitigating the "overfetch" problem. The ability of sub-ranked memory to support fine-grained accesses is briefly mentioned in [25, 124], but the tradeoffs are neither discussed nor evaluated. Compared to the many-narrow-channels approach, sub-ranked memory systems are less expensive in pin cost, since only one address/command bus is required per wide data path. In Chapter 6, we design a memory system that provides efficient fine-grained accesses and dynamic reliability tradeoffs unlike prior work.



(a) A cache with a small line size

(b) The sector cache

(c) A cache with a large line size

Figure 3.7: Comparison of cache line organizations.

3.3.2 Sector Caches

Similar to modern DRAMs, recent cache designs are also tuned to coarse-grained accesses; a cache line of 64B or larger. Large cache lines reduce not only *tag* and ECC storage but also cache misses by prefetching spatially adjacent data. The adaptive granularity memory system we design in Chapter 6, however, allows fine-grained main memory accesses; hence, it is essential to support fine-grained data management in caches also. Like the many-narrow-channels approach in DRAM, caches with small cache lines (e.g., 8B) are unacceptable since this increases tag and ECC overheads significantly.

The sector cache is a design originally used in the IBM 360/85 system [75] and initially aimed to reduce tag overhead by dividing each cache line into multiple sectors and providing each sector with its own dirty and valid bits. Figure 3.7 illustrates caches with a small line and a large line as well as the sector cache. D and V represent a dirty bit and a valid bit, respectively. The

sector cache can manage fine-grained data by allowing partially-valid cache lines, while not increasing tag overhead significantly.

Chapter 4

Efficient Cache Memory Protection

It is important to minimize the impact on area, power, and application performance of strong resiliency mechanisms because of the continuing increase in soft error propensity. The combination of growing last-level cache (LLC) capacity, shrinking SRAM cell dimensions, decreasing critical charge (Q_{crit}), and increasing fabrication variability that reduces error margins is leading to a higher SER [79, 106, 110]. In particular, recent trends show that multi-bit errors in the array are becoming significant: Many memory cells can fall under the footprint of a single energetic particle strike [79, 90, 103] as semi-conductor design rules shrink; and the likelihood of faulty bit rates grow rapidly as V_{CC} of a cache array is lowered to reduce power consumption [10, 35, 127].

Because of these trends, the need to account for high LLC SER and also tolerate multi-bit errors in SRAM arrays is becoming acute [31]. Researchers have already observed up to 16-bit errors in SRAM arrays and are predicting potentially higher counts in the future [16, 79, 90]. The more powerful protection mechanisms required to correct large numbers of bit flips come at a cost of storing more redundant information or modifying physical designs, as well as increased energy requirements [63, 92].

In this chapter, we aim to reduce the overheads of providing increased protection against soft errors in large SRAM arrays and present two LLC protection mechanisms: Memory Mapped ECC (MME) [132] and ECC FIFO [131]. Both mechanisms apply two-tiered protection described in Chapter 3, but exploit different schemes for managing T2EC information in main memory. MME stores the T2EC as addressable data within the memory hierarchy itself, while ECC FIFO uses a FIFO structure in main memory.

We evaluate MME and ECC FIFO using full-system cycle-based simulation of memory-intensive applications from the SPLASH2 [129], PARSEC [21], and SPEC CPU 2006 [109] suites. We show that the proposed architectures outperform prior techniques for reducing LLC protection overheads on nearly every metric. We estimate $15-25\%$ and $9-18\%$ reductions in area and power consumption of LLC, respectively, while performance is degraded by only 0.7% on average and by no more than 2.8%.

In the remainder of this chapter, we first develop two-tiered protection for LLC in Section 4.1, then explain the proposed LLC protection schemes: Memory Mapped ECC in Section 4.2 and ECC FIFO in Section 4.3. Section 4.4 discusses LLC protection tradeoffs enabled by the two-tiered approach, Section 4.5 evaluates MME and ECC FIFO, Section 4.6 describes related work, and Section 4.7 summarizes this chapter.

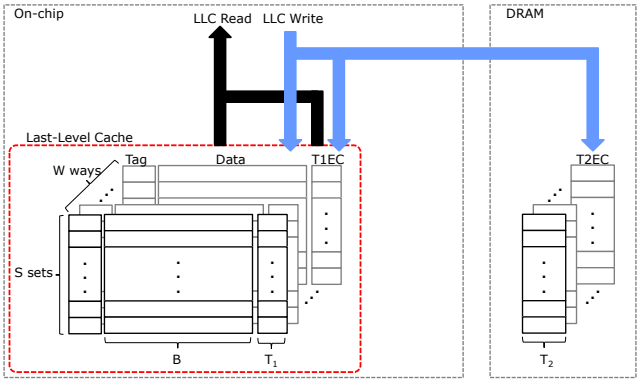


Figure 4.1: Two-tiered protection for LLC with dedicated on-chip storage for the T1EC, while the the T2EC is stored in off-chip memory.

4.1 Two-tiered Protection for LLC

In this section, we develop two-tiered protection mechanisms for LLC. As discussed in Chapter 2, two-tiered protection consists of the T1EC and the T2EC. We provide dedicated on-chip storage for the T1EC in every cache line and use it exclusively for error detection or for light-weight error correction, while we off-load the T2EC to main memory namespace to avoid dedicated on-chip storage for the T2EC.

On-chip Storage Overhead. As shown in Figure 4.1, the on-chip ECC storage overhead of two-tiered LLC protection is for T1EC only and is equal to $S \times W \times T_1$ bytes, where S is the number of LLC sets, W is the LLC associativity, and T_1 is the number of bytes required for the T1EC corresponding to an LLC line of B bytes (see Table 4.1 for notations). For example, an 8-way interleaved parity T1EC that detects up to a 15-bit burst error requires just 1B of storage for a 64B cache line. This is a much smaller overhead than in the

Table 4.1: Definitions and nominal parameters used for evaluation.

LLC parameters		
number of sets	S	2048 sets
associativity	W	8 ways
line size	B	64B
LLC size	$S \times W \times B$	1MB
the way within a set in which address x is cached	*way(x)*	
the set within the LLC in which address x is cached	*set(x)*	
Two-Tiered Protection		
T1EC size per cache line	T_1	see Section 4.4
T2EC size per cache line	T_2	see Section 4.4
MME		
memory address in which the T2EC bits protecting address x are stored	*T2EC_addr(x)*	
base pointer to the T2EC region	*T2EC_base*	
multiple T2ECs that are mapped to a single LLC line	*T2EC line*	
ECC FIFO		
Tag size	t	$\log_2(S \times W) = 14$ bits
the coalesce buffer size	m	$\lfloor \frac{B}{t+T_2} \rfloor = 6$
T2EC FIFO size	k	see Section 4.3.3
Others		
DRAM bandwidth	BW	5.336GB/s
eager write-back period	T_{ewb}	10^6 cycles
processor clock	*clock*	3GHz

conventional approach of uniformly providing an 8-way interleaved SEC-DED code requiring 8B for each cache line. Both the 8-way interleaved parity T1EC and the conventional 8-way interleaved SEC-DED have similar burst-error detection capability (up to 16-bit bursts for SEC-DED and 15-bit bursts for the T1EC), although the SEC-DED code can detect a larger number of non-burst bit errors. To provide a matching correction capability, the T2EC is an 8-way interleaved SEC-DED that is stored in the off-chip DRAM. Section 4.4 describes more examples of two-tiered codes with error detection/correction capabilities and area/power overheads.

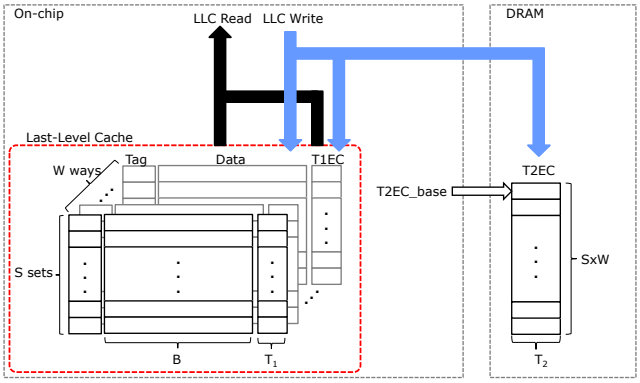


Figure 4.2: Memory Mapped ECC architecture that stores the T2EC information as addressable data within the memory hierarchy.

4.2 Memory Mapped ECC

Memory Mapped ECC (MME) is based on two-tiered protection and stores T2EC information within the memory hierarchy as addressable data. Thus, the memory-mapped T2EC information can be cached in the LLC and eventually stored in low-cost off-chip DRAM. Figure 4.2 illustrates how MME organizes T1ECs and T2ECs in the LLC and DRAM.

Storing the T2EC bits as cacheable data, rather than in dedicated storage, offers several important advantages. First, by allowing the redundant information to be stored in DRAM and cached on chip, we transparently and dynamically adjust the on-chip T2EC storage to the number of dirty cache lines using the existing cache management mechanism. We do not limit the number of dirty lines in a cache set or the cache overall. Thus, we eliminate all SRAM leakage and area overheads of dedicated T2EC storage, improving on the savings suggested in the past [64, 71]. The second main advantage is

the generality of our technique and the flexibility it provides in choosing the T2EC (see Section 4.4).

We now describe how to integrate MME within the memory hierarchy including interfaces with the main memory namespace and the LLC controller in Section 4.2.1–4.2.2, followed by a discussion of error detection and correction procedures in Section 4.2.3. We also discuss how to support write-through L1 caches in Section 4.2.4.

4.2.1 T2EC in Main Memory.

MME requires a region of the memory namespace (the T2EC region) to be allocated for T2EC storage. Our current implementation is to map this region to physical DRAM addresses and avoid address translation when reading and writing T2EC. An on-chip register, (*T2EC_base*), points to the start of the T2EC array, which is only $T_2 \times S \times W$ bytes in size, where T_2 is the number of bytes required for the T2EC corresponding to an LLC line of B bytes (see Table 4.1 for notations). For example, using interleaved SEC-DED codes, the T2EC array is just 128KB of physical DRAM for every 1MB of an LLC with 64B lines. Each physical LLC line is associated with a T2EC in physical memory, as follows (See Table 4.1 for notations):

$T2EC\_addr(x) = T2EC\_base + (way(x) \times S + set(x)) \times T_2$.

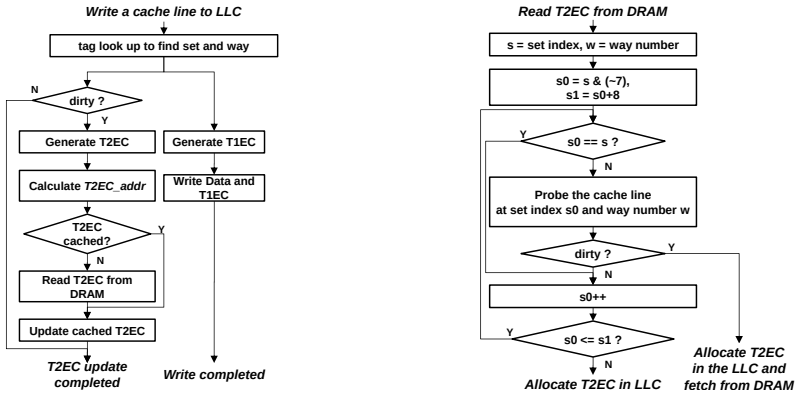
Using this mapping for the interleaved SEC-DED T2EC on 64B LLC lines, eight T2EC entries are mapped into 64B and form a *T2EC cache line*.

We define the T2EC region in DRAM to be cacheable in the LLC. This has two significant advantages over writing T2EC information directly out to DRAM. First, it reduces the DRAM bandwidth required to support MME by exploiting locality in T2EC addresses and accessing T2EC data in the cache when possible. We map LLC lines from consecutive sets to be adjacent in T2EC DRAM storage to increase the likelihood that accesses to arrays in the program will have their T2EC data placed in the same T2EC line. A T2EC cache line is fetched to the LLC, modified in the LLC, then evicted to DRAM like a normal cache line. In other words, the LLC is dynamically partitioned to data and T2EC cache lines, and T2EC storage overheads are eventually off-loaded to DRAM.

The second advantage of caching T2EC data is in matching the T2EC access granularity to DRAM burst size. A typical T2EC will require a small number of bytes for every LLC line, and writing at such a fine granularity to modern DRAM systems is inefficient and wastes limited DRAM bandwidth. By caching T2EC data, the LLC acts as a write-combining buffer for T2EC bits, and all writes to DRAM are performed at the granularity of one or more DRAM bursts.

4.2.2 LLC operations

LLC Read. We assume the cache level preceding the LLC (L1, if the LLC is L2) is a write-back cache; hence, the LLC is always accessed at a coarse granularity of a cache line. When a cache line is read from the LLC (fill



(a) LLC write (b) Allocate a T2EC line

Figure 4.3: LLC write and T2EC update in LLC with MME.

into preceding level or write-back into DRAM), the T1EC is used to detect and potentially correct errors. The detailed error detection and correction procedures are described in Section 4.2.3.

LLC Write. The procedure for writing a line into the LLC is summarized in Figure 4.3(a). In all write cases, the T1EC is computed and stored in the T1EC portion of the cache line. A T2EC, however, is only computed for dirty lines that are written back into the LLC from a previous-level cache (e.g., L1). This newly generated T2EC is mapped to a cacheable DRAM address. If this T2EC address is already in the cache, it is updated with the newly computed T2EC. If the T2EC address is a cache miss, a line is allocated for it in the LLC and the T2EC is fetched from DRAM (see the next paragraph for more details). Note that a T2EC is generated only for a dirty cache line and that T2EC encoding / caching is concurrent with the LLC data write.

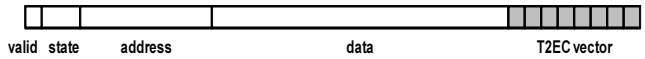
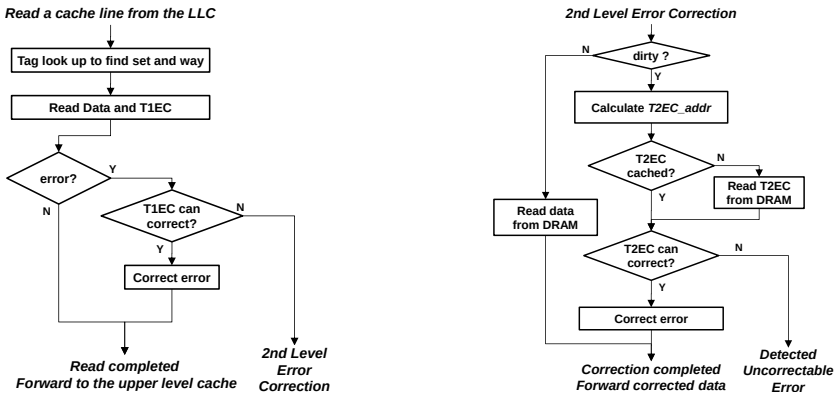


Figure 4.4: Extended MSHR entry. The bit vector length is equal to the number of T2EC entries mapped to a single LLC line.

T2EC Write. The MME architecture provides an optimization for the allocation and population of T2EC lines in the LLC. Because a T2EC is just an ECC for a dirty cache line, we can often simply allocate a T2EC line in the LLC without first fetching it from DRAM. If all the cache lines that are mapped to a single T2EC cache line are clean, the corresponding T2EC information in DRAM is stale and unnecessary, and it is wasteful to read it. The optimized procedure for allocating and populating a T2EC line is illustrated in Figure 4.3(b). T2EC cache line fills occur only when there is at least one dirty cache line among the cache lines mapped to the T2EC cache line.

Filling a T2EC line from DRAM is similar to a regular data line fill, and the read request is first placed in an MSHR (Miss Status Handling Register). The MSHR is responsible for merging the newly computed T2EC with the T2EC data corresponding to other cache lines that is being fetched from DRAM. This merging capability is similar to that provided by the MSHRs between two cache levels where the granularity of writes is smaller than the cache line size (e.g., when L1 line size is 16B and L2 line size is 64B). To accomplish the T2EC merging, each MSHR entry is extended with a bit vector that indicates what portion of the T2EC line has just been calculated and what portion is being fetched from DRAM (Figure 4.4). This bit vector enables



(a) LLC read (b) 2^{nd} level error correction

Figure 4.5: LLC read and error correction in LLC with MME.

additional pending T2EC writes to the same T2EC line to proceed, while the line is being fetched.

4.2.3 Error Detection and Correction

Error Detection. When properly designed, the error detection using a T1EC, which is the most common operation, is less complex and consumes less energy per access than those of the conventional ECC. For instance, detecting a single-bit error is much simpler using a parity based T1EC than with a SEC-DED code. We do not exploit this advantage in our architecture when evaluating MME in Section 4.5 and refer the reader to the PERC paper [99], which uses decoupled error detection and error correction to improve the performance and energy of a first-level cache.

Error Correction. When the T1EC detects an error, the correction mechanism will attempt to correct the error using the T1EC. If possible, the cor-

rection will be done with low energy and latency. If the chosen T1EC is not strong enough for correction, and the cache line is clean, error correction simply involves re-fetching the line from DRAM with the corresponding energy and latency penalty. For dirty lines, the correction mechanism accesses the T2EC data and decodes the redundant information. Thus, the maximum penalty for correction is a single LLC line fill from DRAM and invoking T2EC decoding. Given the rare occurrence of an error on a dirty line that cannot be corrected by the T1EC, this overhead is entirely negligible. The error correction procedure using the T1EC and T2EC is described in Figure 4.5(a) and Figure 4.5(b).

4.2.4 Supporting Write-Through L1

So far we assumed L1 (or the cache level preceding the LLC) is write-back (WB), where a dirty eviction to L2 is always at a coarse granularity of a cache line. Supporting write-through (WT) L1 that is used in many commercial designs, however, requires no fundamental changes in the MME architecture. Each store instruction in a processor with WT L1 caches writes the data to an L1 cache line, then eventually updates a word in an L2 cache line. In MME, each update of an L2 cache line updates T2EC information also; hence, the fine-grained and more frequent L2 updates (in a system with WT L1) can potentially lead to more frequent T2EC cache misses. Since we do not read T2EC information unless an error occurs, the more frequent T2EC misses can be easily tolerated with the MSHR described in Section 4.2.2.

As a result, the MME architecture supporting WT L1 should be able to support more concurrent T2EC misses, and this may impact performance and power due to potentially higher MSHR/LLC utilization. Although we do not evaluate MME with WT L1, the relatively low activity factor of the LLC and the MME's reduced LLC area can improve power/energy efficiency, since LLC power consumption, as shown in Section 4.5, largely depends on the leakage power.

4.3 ECC FIFO

The second LLC protection mechanism we propose is ECC FIFO. Similar to MME, ECC FIFO also uses two-tiered protection, but the mechanism of off-loading T2EC overheads is different; a FIFO in main memory off-loads T2EC storage overheads.

In ECC FIFO, every time a T2EC is generated, the redundant information is pushed into the T2EC FIFO along with a *Tag* that indicates the corresponding dirty physical LLC line. Thus, when the T1EC detects an error that it cannot correct, T2EC FIFO can be searched starting from the newest entry until a matching tag is found and the T2EC information can be retrieved. The FIFO is simple to implement and allows us to easily identify the T2EC corresponding to an error with very low management overhead and with efficient transfer of T2EC data to DRAM.

Figure 4.6 illustrates ECC FIFO architecture; on-chip ECC overhead is the same as that of MME, but T2EC FIFO and the coalesce buffer manage

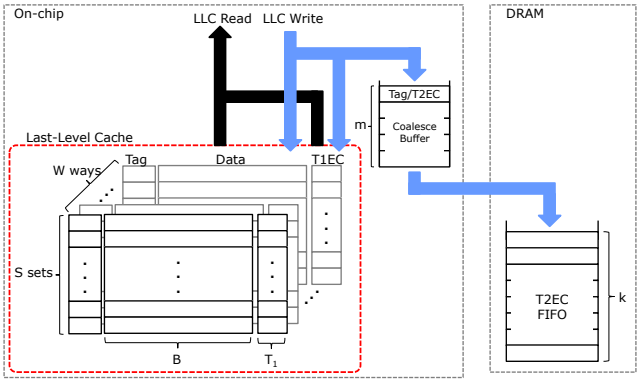


Figure 4.6: ECC FIFO architecture that stores the T2EC information into the T2EC FIFO in DRAM.

T2EC information. Unlike MME, ECC FIFO does not cache T2EC as data; hence, ECC FIFO does not affect LLC caching behavior.

4.3.1 LLC operations and Error Detection/Correction

Since ECC FIFO has similar on-chip organization of uniform T1EC as MME, LLC read/write operations in ECC FIFO are almost the same as those operations in MME except T2EC information management as well as error correction procedure using T2EC.

LLC Write and T2EC Write. In ECC FIFO, T2EC is encoded only with dirty line eviction to the LLC as in MME. We assume that the cache level preceding the LLC (e.g., L1, if the LLC is L2) is a write-back cache in ECC FIFO. Supporting write-through L1 in ECC FIFO is impractical since frequent fine-grained writes to L2 generate too much T2EC traffic. The encoded T2EC is packed with a *Tag* , which is a pointer to the corresponding

physical data line in the LLC. A Tag is composed of the set number and the way number of the cache line so that the T2EC can later be associated with a detected error. The Tag requires $\log_2(S \times W)$ bits, e.g., 18 bits for a 16MB LLC with 64B cache lines. The packed Tag/T2EC pair is pushed into a coalesce buffer and then written into the FIFO with DRAM-burst granularity. The coalesce buffer is necessary to achieve high DRAM throughput, because a single Tag/T2EC is only 10.5 bytes in the example above (8 bytes for an interleaved SEC-DED T2EC and 2.5 bytes for the Tag), smaller than the minimum burst size of modern DRAM controllers, which are architected for LLC-line granularity (e.g., 64B). Using the parameters above, we can coalesce up to 6 Tag/T2EC pairs into a single DRAM write-back.

LLC Read, Error Detection, and Error Correction. Error detection mechanism of ECC FIFO is the same as MME; the uniform T1EC detects (and possibly correct light-weight errors). Once T1EC detects, but cannot correct errors in a dirty LLC line, the first step required to correct such an error using T2EC is to identify the FIFO entry that contains the T2EC data. The FIFO is searched sequentially to find the Tag, which corresponds to the LLC physical line in which the error was found. The search starts from the newest FIFO entry, which can still be in the on-chip coalesce buffer, and proceeds from the current FIFO tail towards the head. Once the most recent matching pair is found, the T2EC (and the T1EC, if needed) attempts to correct the errors. Thus, the worst case penalty of T2EC error correction is the required time to read and compare all Tag/T2EC pairs from the FIFO.

This takes roughly $k \times (B/m)/BW$ seconds, where k is the size of the FIFO in number of pairs, m is the number of pairs in the coalesce buffer and BW is the expected throughput of the DRAM channel in bytes/sec (see Table 4.1 for notations). For instance, the worst case correction latency is around 1.66 ms when k is one million entries, m is 6, B is 64 bytes, and BW is 6.4GB/s. This overhead, however, is entirely negligible given the rare occurrence of an error that requires T2EC for correction, which we estimate at one error every 155 days for a 32MB cache in today's technology [106, 110]. Note that errors on clean cache lines will be corrected by re-fetching data from DRAM as in MME.

4.3.2 T2EC FIFO

We propose to use a large circular buffer in the main memory space as the T2EC FIFO. The FIFO has two significant advantages over other T2EC storage options: (i) a FIFO allows arbitrary coalescing with a trivial and small on-chip buffer that can still maximize DRAM throughput; and (ii) a FIFO is easy to manage in software and provides a clear way to identify the most recent update to an LLC physical line's T2EC data. Information on the FIFO size and base address is stored within the LLC controller and can be set by privileged hypervisor or OS instructions, or through the BIOS.

One caveat to using a circular buffer is that a T2EC push into the FIFO overwrites the oldest entry. If the physical cache line that corresponds to the overwritten T2EC FIFO entry has not been modified or evicted from

the LLC, then its T2EC information is overwritten and lost. At that point, the line becomes *T2EC unprotected*, and the system will not be able to recover from an error within it.

4.3.3 T2EC Overwrite and Unprotected Lines

To analyze the effects of the T2EC FIFO size on T2EC protection capability, we define P_{unprot} as the probability that a T2EC cannot correct the LLC line due to a T2EC overwrite in the circular buffer. Even though the buffer is finite, the probability of unprotected lines is small for two main reasons: (i) inherent memory access patterns in applications that limit the lifetime of dirty lines in the LLC through reuse and capacity/conflict evictions; and (ii) limiting dirty line lifetime by programming the LLC controller to periodically clean dirty lines.

Reused and Evicted Dirty Lines. In many applications, the natural access pattern leads to relatively short lifetime of dirty lines in the LLC. Cache lines are often re-written by the application as new results are generated, rendering earlier computed T2EC data stale and unnecessary. In other cases, LLC lines are evicted to make room for new lines being fetched, and again any existing corresponding T2EC data in the FIFO becomes obsolete. We denote the fraction of T2EC entries that correspond to reused lines in the LLC as F_{reuse} and the fraction of entries corresponding to evicted lines as F_{evict}.

For example, `lbm` from the SPEC CPU 2006 suite [109] has streaming memory access patterns; hence, the vast majority of dirty lines are written-

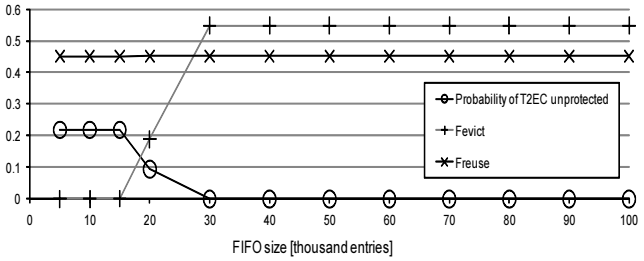


Figure 4.7: P_{unprot}, F_{reuse}, and F_{evict} in the `lbm` application.

back into DRAM before a T2EC overwrite occurs. Simulation results (parameters as described in Section 4.5) show that F_{evict} goes up rapidly as the FIFO size is increased from 15K to 30K entries; the probability of T2EC unprotected reaches 0 when the FIFO is larger than 30K entries, which is 313KB of DRAM space (Figure 4.7).

Periodically Cleaning Dirty Lines. Although some applications (such as `lbm`) do not suffer from T2EC unprotected with a reasonable FIFO size, other applications may cause overwrites leading to unprotected lines regardless of the FIFO depth. We utilize the previously proposed *eager write-back* technique [69] to bound the lifetime of a dirty line in the LLC. We explain this in detail and derive a model to analyze the T2EC unprotected probability with eager write-back.

The original eager write-back technique, proposed in [69], opportunistically writes dirty lines into DRAM when a dirty line is in the least-recently used position and a DRAM issue slot is available. This reduces pressure on DRAM when demand fetches are necessary and increases performance. We utilize a more predictable approach, which retains the performance advantages, in

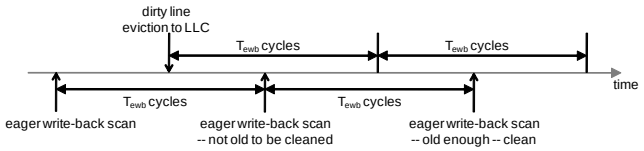


Figure 4.8: An example time-line of dirty line eviction to the LLC and eager write-back.

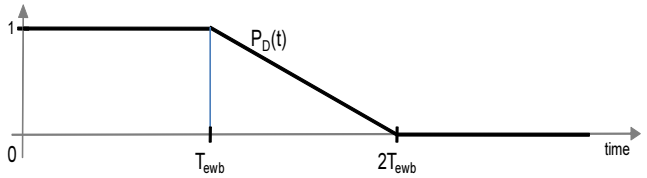
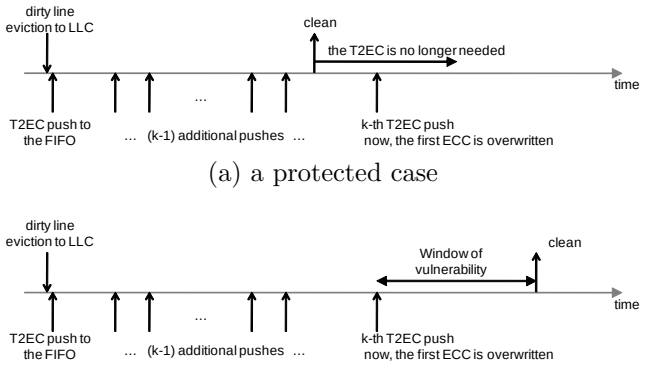


Figure 4.9: Probability that a dirty cache line remains dirty in time.

which lines are periodically probed and written back if dirty. This is similar to the policy used in cache reliability studies [64, 71] and decay caches [60]. Our eager write-back implementation scans each cache line with a predetermined period, T_{ewb} cycles, and eagerly writes a dirty line older than the period.

Figure 4.8 shows an example time-line of a dirty line in the LLC, from the time it is evicted into the LLC until the time it is cleaned by an eager write-back. Each cache line is scanned once per T_{ewb} cycles so that a dirty line is eagerly written back to DRAM within a maximum of $2\times T_{\text{ewb}}$ cycles after the line is evicted into the LLC. Based on this observation and the fact that the periodic scanning is independent of the eviction times, we can define $P_{\text{D}}(t)$ as the expected probability that a dirty cache line that was written into the LLC at time $t = 0$ remains dirty at time t, as shown in Figure 4.9. Since eager write-back does not clean dirty lines younger than T_{ewb} cycles, $P_{\text{D}}(t) = 1$ for the first T_{ewb} cycles, after which $P_{\text{D}}(t)$ decreases linearly until it reaches 0 at time



(a) a protected case

(b) an unprotected case

Figure 4.10: Time-lines of dirty lines with T2EC pushes to the FIFO.

$t = 2 \times T_{\text{ewb}}$ cycles. Figure 4.10 illustrates the two potential time-lines of a dirty LLC line that is neither evicted nor reused with respect to being cleaned by an eager write-back or overwritten in the T2EC FIFO. Figure 4.10(a) depicts the case that a line is cleaned by an eager write-back before its T2EC data is overwritten and is thus fully protected, whereas Figure 4.10(b) shows a window of vulnerability opening, if the T2EC is overwritten before the line is written back.

To summarize the discussion above, the factors determining whether a line becomes T2EC unprotected are the period of time in which a T2EC entry is required to protect the cache line (the T2EC entry's *valid-time*) and the time for the T2EC to be overwritten in the FIFO (its *overwrite-time*); when a T2EC entry's valid-time is longer than its overwrite-time, the cache line becomes T2EC unprotected. The valid-time is the property of the LLC and memory access pattern; the time it takes for the line to be cleaned by an eager write-back determines the valid-time unless the line is reused or evicted

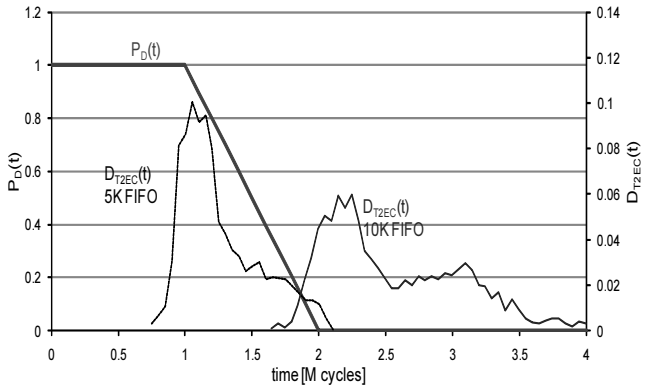


Figure 4.11: Examples of $D_{\mathrm{T2EC}}(t)$ for 5k and 10k entry FIFOs from the `omnetpp` application overlaid on $P_{\mathrm{D}}(t)$ when T_{ewb} is 1M cycles.

before it is cleaned. The overwrite-time is a function of the FIFO depth and the rate at which dirty lines are written into the LLC (property of memory access pattern and parameters of the caches closer to the core). Because the valid- and overwrite-times are application-specific and dynamic, we model their impact on unprotected lines using a probability density function of the overwrite-time, $D_{\mathrm{T2EC}}(t)$. $D_{\mathrm{T2EC}}(t)$ is the distribution of T2EC overwrite-time that a T2EC entry that was computed at time $t = 0$ is overwritten at time $t > 0$. Figure 4.11 shows examples of $D_{\mathrm{T2EC}}(t)$ from `omnetpp` with different FIFO sizes. A larger FIFO shifts $D_{\mathrm{T2EC}}(t)$ towards longer overwrite times so that eager write-back cleans nearly all the dirty lines before the T2EC is overwritten.

We can now write the probability of an unprotected line as in Equation 4.1.

$$P_{\mathrm{unprot}} = (1 - F_{\mathrm{reuse}} - F_{\mathrm{evict}}) \times \int_0^\infty P_{\mathrm{D}}(t) \times D_{\mathrm{T2EC}}(t)\, dt \qquad (4.1)$$

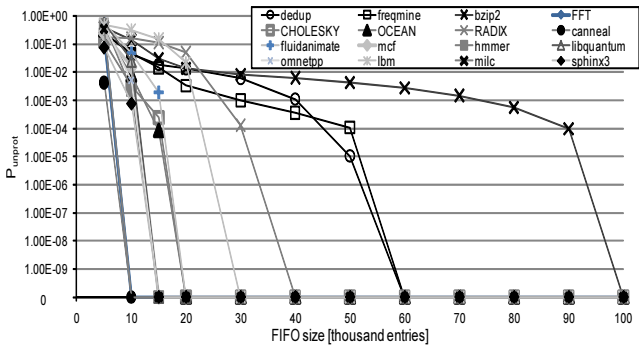
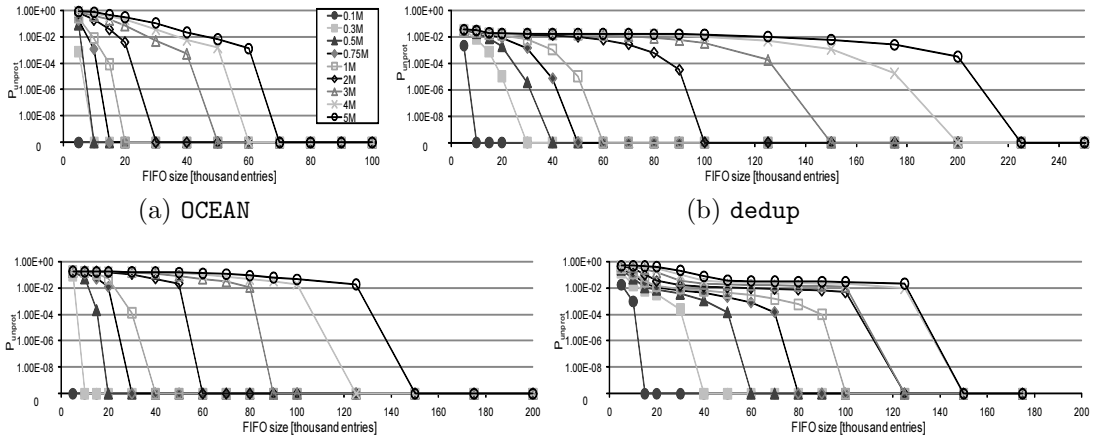


Figure 4.12: Probability of T2EC unprotected when T_{ewb} is 1M cycles. A FIFO larger than 40k entries is required only in three applications: `dedup`, `freqmine`, and `bzip2`.

We used a detailed simulator, described in Section 4.5, to collect T2EC overwrite-time information for a variety of benchmark applications and a range of FIFO sizes. We then applied the model of Equation 4.1 and compared the resulting expected fraction of unprotected LLC lines to that from the cycle-accurate simulation. Although we do not present the detailed comparison results in this dissertation, the model and the simulation agreed to within 1%.

Impact of FIFO size and Eager Write-Back Period. Figure 4.12 shows the decrease in probability of T2EC unprotected as the size of the FIFO increases for a range of applications when the eager write-back period is 1M cycles. All but 3 of the applications (`dedup` `freqmine`, and `bzip2`) are fully protected with a FIFO of 40K entries requiring only 417KB of DRAM storage. The deepest FIFO required to avoid overwrites with $T_{\text{ewb}} = 1M$ cycles is 100K-entries for `bzip2`, requiring about 1MB of DRAM storage. Figure 4.13 shows the effects of varying T_{ewb} on a few representative applications. All but



(a) `OCEAN` (b) `dedup`

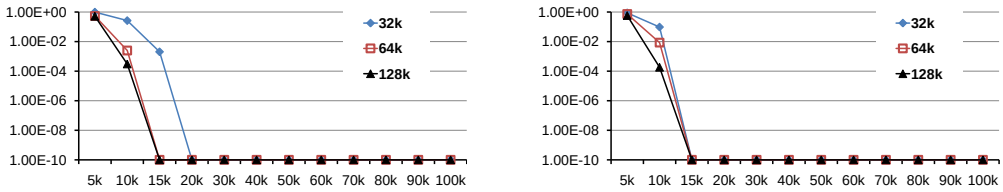
(c) `RADIX` (d) `bzip2`

Figure 4.13: Probability of T2EC unprotected varying T_{ewb}.

4 of the applications evaluated behaved similarly to `OCEAN` of the SPLASH2 suite, requiring a FIFO smaller than 100K entries even with $T_{\text{ewb}} = 5M$ cycles. `RADIX`, `bzip2`, `freqmine`, and `dedup` required deeper buffers, with `dedup` requiring a 220K-entry (2.5MB) buffer with $T_{\text{ewb}} = 5M$ cycles.

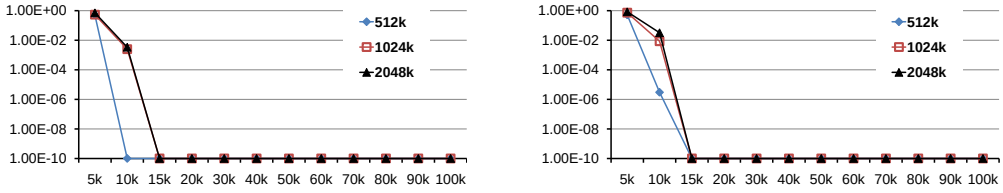
Guaranteeing No T2EC Unprotected Lines. If the T2EC FIFO is sufficiently large, then eager write-back will always clean a dirty line before the corresponding T2EC is overwritten. With eager write-back, the maximum T2EC valid-time is $2{\times}T_{\text{ewb}}$ cycles so we need to choose a FIFO deep enough to make the minimum T2EC overwrite-time longer than the maximum valid-time. The minimum T2EC overwrite-time is k/R, where R is the maximum FIFO fill rate, and a FIFO greater than $2{\times}T_{\text{ewb}}{\times}R$ prevents T2EC unprotected.

The maximum FIFO fill rate is essentially the maximum possible rate of dirty line writes into the LLC. At worse, every store instruction can cause



(a) `hmmer` (b) `omnetpp`

Figure 4.14: P_{unprot} with varying L1 size.



(a) `hmmer` (b) `omnetpp`

Figure 4.15: P_{unprot} with varying L2 size.

a dirty line eviction into the LLC. Alternatively, the fill rate may be limited by the total DRAM write bandwidth (to fill the FIFO) or the bandwidth of the LLC controller. In our simulated system (see Table 4.4), the DRAM write bandwidth sets the tightest bound. The FIFO size required is $2 \times T_{\text{ewb}} \times m/B \times BW/clock$, where *clock* is the clock frequency of eager write-back operations. This corresponds to a FIFO of $346,070$ entries requiring 3.6MB. While this is a fairly large buffer, its size is very small compared to today's main-memory capacities and replaces valuable on-chip area.

Sensitivity to Cache Size. Intuitively, a larger L1 or a smaller L2 will reduce the number of T2EC unprotected lines; a larger L1 reduces the rate of dirty evictions into the LLC, and as a result, T2EC overwrite-time increases. Similarly, a smaller L2 will cause cache lines to be replaced more often, due

Table 4.2: Baseline and two-tiered ECC codes assuming 64B cache line (baseline codes are regarded as T1EC without T2EC).

	T1EC		**T2EC**		**T1EC burst error correction**	**T1EC burst error detection**	**T2EC burst error correction**
	code	size	code	size			
baseline							
S	8 way SEC-DED	8B	N/A	N/A	8	16	N/A
D	8 way DEC-TED	16B	N/A	N/A	16	24	N/A
two-tiered error protection							
PS	8 way parity	1B	8 way SEC-DED	8B	N/A	15 bits	8 bits
PN	16 way parity	2B	4 way SNC-DND	8B	N/A	31 bits	4 nibbles
PB	32 way parity	4B	8 bit symbol RS	8B	N/A	63 bits	4 bytes
SD	8 way SEC-DED	8B	8 way DEC-TED	8B	8bits	16 bits	16 bits

to capacity constraints, which then decreases T2EC valid-time. We ran simulations varying the L1 size between $32 - 128$KB and the L2 size between $512 - 2048$ KB. The behavior of T2EC unprotected is as expected and described above. The overall impact of cache sizes is, however, relatively small, and we omit the full results for brevity. The conclusion is that the FIFO depth required to eliminate the possibility of T2EC unprotected lines is unchanged in most applications; it changes by $5 - 10$K entries only in four of the eight applications we tested (SPEC CPU 2006 applications), including `hmmer` and `omnetpp`. Figure 4.14 and Figure 4.15 present P_{unprot} with different L1 and L2 sizes in `hmmer` and `omnetpp` applications.

4.4 Error Protection Tradeoffs

In this section, we evaluate improvements in protection capabilities and discuss new tradeoffs enabled by the two-tiered approach. One of the main advantages of our two-tiered protection is the flexibility it provides in choosing the T1EC and T2EC.

With the design considerations presented in Section 3.1.4, Table 4.2 describes the configurations and burst error detection/correction capabilities of a number of one-tiered baseline ECC codes as well as two-tiered ECC codes for MME and ECC FIFO. Other error detecting codes, such as cyclic redundancy coding (CRC), can be combined and yield stronger random error detecting capability. However, we choose interleaved parity and SEC-DED as the T1EC considering both burst error detection capability and decoding latency. All of the two-tiered codes have identical T2EC size (8B) per 64B cache line; hence, the impact on performance in MME and ECC FIFO will be roughly the same across the presented two-tiered coding schemes. This leaves the tradeoff to be between the protection capability (T1EC detection and T2EC correction) and on-chip area (T1EC size).

The PS configuration (parity T1EC and SEC-DED T2EC) allows us to directly compare our architecture to the baseline one-tier 8-way interleaved SEC-DED codes (S), which are commonly used in related and prior work. The two-tiered PS provides the same bursty error correction capability (up to 8-bit bursts) as the one-tier SEC-DED (S) with reduced on-chip area. The number of detectable errors is different, however. The interleaved parity can detect bursts up to 15 bits long, whereas the conventional interleaved SEC-DED codes can detect up to 16-bit bursts (8-way interleaved double-bit errors). The codes also differ in their detection capability for non-burst multi-bit errors, which we evaluate later in this section using error injection.

Table 4.3: Area, leakage power, and array energy per read access.

	baseline			**MAXn**			**two-tiered protection**			
	no ECC	**S**	**D**	**MAX1**	**MAX2**	**MAX4**	**PS**	**PN**	**PB**	**SD**
Leakage Power (W)	1.4	1.6	1.8	1.5	1.5	1.6	1.4	1.5	1.5	1.6
Energy per Read (nJ)	2.0	2.4	2.9	2.1	2.1	2.7	2.1	2.1	2.2	2.4
Area (mm^2)	10.0	12.0	14.1	10.3	10.4	10.6	10.2	10.5	10.9	12.0
ECC area overhead (%)	-	20.7	41.7	3.4	4.2	6.5	2.4	4.9	9.8	20.7

The PN (parity T1EC and nibble-based SNC-DND T2EC) and PB (parity T1EC and byte-based RS T2EC) configurations can correct even longer bursts. Correction capability is up to 4 nibbles (13–16 bits) and 4 bytes (25–32 bits), respectively, while the on-chip overhead is still very low; even the 32-way parity T1EC of PB requires only 4 bytes, half the space of the baseline (S).

The SD configuration uses a DEC-TED code, which is separable into an 8-bit SEC-DED, with an additional 8 bits providing detection and correction for one more bit error [91]. In our two-tier SD, the 8-bit SEC-DED portion is the T1EC that corrects nearly all errors, up to 8-bit bursts, with very low latency. The T2EC DEC-TED, which is constructed from the 8 bits stored on chip in the T1EC and the 8 bits that are retrieved from the T2EC, is used only in the very rare case of an error that was detected by the T1EC but cannot be corrected by it. Note that the T1EC has only half the on-chip storage overhead of the one-tier DEC-TED configuration (D).

Storage, Energy, and Latency Overheads. Table 4.3 compares area overheads, leakage power, and array energy per access of the two-tiered protection, MAXn, and baseline schemes. MAXn is a generalization scheme of the area-efficient cache architecture [64]. The MAXn architecture uses decoupled

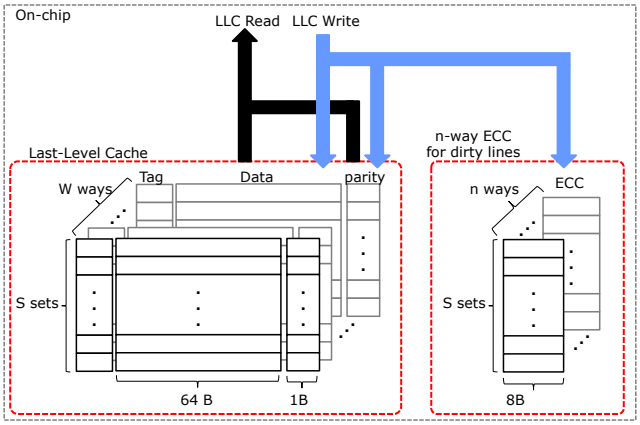
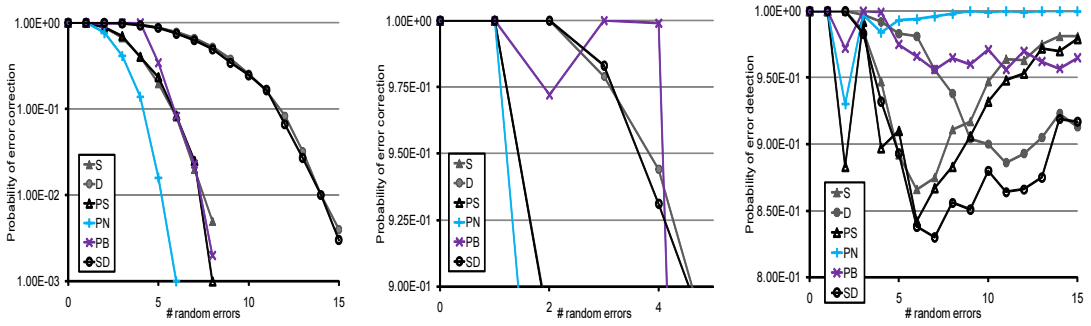


Figure 4.16: MAXn architecture.

error detection/correction approach. Each cache line (64B) has parity (1B) for error detection, and a separate n-way set associative dirty line ECC cache provides error correction using SEC codes. As a result, MAXn allows only n dirty lines per set, and any violation to this constraint will force an oldest dirty line to be evicted, increasing off-chip traffic. Figure 4.16 illustrates the MAXn architecture.

We use CACTI 5 [119] to report the properties of a 1MB LLC and related ECC in a 45nm process. Our architectures only dedicate on-chip storage to the T1EC and significantly reduces the area and leakage overheads of strong ECC. For up to 8-bit burst error correction (S, MAXn, and PS), the two-tiered error protection has much lower area overhead (2.4%) compared to the one-tier SEC-DED baseline (20.4%) and MAXn ($3.4\% - 6.5\%$ for $n = 1 - 4$). With the increased burst error protection of PN and PB, the area overhead is very small (4.9% for PN, and 9.8% for PB). In addition, 16-bit burst error correction in



(a) Error correction (log scale) (b) Error correction (zoomed) (c) Error detection

Figure 4.17: Random error detection/correction capabilities of baseline and two-tiered error codes.

the two-tiered (SD) has the same overhead of the baseline S, whereas it is only supported with a 41.7% area overhead using a one-tier equivalent code (D).

Random Error Behavior. In order to further assess the error protection capability of the two-tiered codes, we use random-error injection and measure error protection and detection capabilities. For each error scenario presented in Figure 4.17, we generate 1000 random cache lines and inject random bit flips to each line, including varying the number of erroneous bits from $1 - 15$ for each protection scheme. Figure 4.17 depicts the probability of errors corrected and detected in the presence of multi-bit random error.

Figure 4.17(a) shows the probability that the ECC codes detect and correct the injected errors. We make two observations regarding the results: (i) in general, two-tiered error protection matches the correction capabilities of its one-tiered counterparts; and (ii) the symbol-based PN and PB error codes perform relatively poorly with respect to random errors, even though they

have superior burst-error correction capabilities (Table 4.2). Also, note that PB cannot correct all double- and quad-bit errors, even though it uses byte-oriented RS, because the parity-based T1EC cannot detect all even-numbered random errors (Figure 4.17(b)).

Figure 4.17(c) depicts the rate of correctly detected errors, including errors that are correctable. The behavior of the one-tier and two-tier approaches is quite different, because the stronger T2EC only comes into play if T1EC detects an error. The parity-based T1EC schemes, perform poorly in detecting two random errors, but the greater interleaving degree of PN and PB have very good detection rates for a large number of errors. Both the one-tier and two-tier SEC-DED and DEC-TED based codes perform well with a small number of errors, but cannot handle a larger number of errors. One conclusion we draw is that further study of error detection/correction techniques is necessary if a large number of random errors become a possibility, which is the case of low-V_{CC} caches [10, 35, 127].

4.5 Evaluation

In this section, we evaluate the cost and performance impact of MME and ECC FIFO. We use the Simics full system simulator [78] with the GEMS [80] toolset. We use the OPAL out-of-order processor model of GEMS to simulate a SPARC V9 4-wide superscalar core with a two-level write-back exclusive cache hierarchy implemented in the Ruby module of GEMS. To accurately account for the impact our technique has on memory bandwidth and performance, we

Table 4.4: Simulated system parameters.

Processor Core	SPARC V9 ISA 4-wide superscalar (3GHz)
L1 Cache	split I/D caches, each 64KB 2-way set associative, 64B cache line write-back, 1 cycle latency
L2 Cache	a unified 1MB cache, 8-way set associative L1 exclusive, 64B cache lines eager write-back ($T_{\text{ewb}} = 10^6$ cycle) L2 latency is 12 cycles including ECC encoding/decoding
DRAM	single channel DDR2 DRAM (5.336GB/s) 667MHz 64-bit data bus open page, Read and Instruction Fetch First policy

integrate DRAMsim [123] into GEMS. We use applications from the SPEC CPU 2006 [109], PARSEC [21], and SPLASH2 [129] benchmark suites that stress the memory subsystem. We do not present results for applications that are not memory intensive, as they are not sensitive to the T2EC traffic. Because accurate full-system simulation is very slow, we augment the simulation results using PIN emulation [77] to study the behavior of applications with large datasets. We believe that simulating a single core with the given parameters proves the utility of MME and ECC FIFO, and conclusions can be drawn on the performance and overheads if implemented within a multi-core processor.

Table 4.4 describes the baseline system configuration. As mentioned in Section 4.3.3, we use eager write-back [69] to limit the lifetime of dirty lines in the LLC and to improve the baseline performance; eager write-back improves the baseline performance by 6–10% in most applications and 26% in `libquantum`. Note that we applied a different eager write-back period for `fluidanimate` (0.25M) to optimize the performance of the baseline scheme.

4.5.1 Workloads

We use a mix of the SPLASH2 [129], PARSEC [21], and SPEC CPU 2006 [109] workloads. We concentrate on applications with large working sets that stress the memory hierarchy and highlight the differences between our architectures and prior work. For the detailed cycle-based simulations, we run the applications from SPLASH2 and PARSEC to completion using a single thread and small to medium problem sizes: tk15.O for `CHOLESKY`; 64K samples for `FFT`; a 514×514 grid for `OCEAN`; 1M samples for `RADIX`, and the `simsmall` inputs for all PARSEC applications. For the SPEC CPU 2006 workloads, we use the reference input dataset and a representative region of 200M instructions, as indicated by Simpoint [49]. We use much larger datasets and execute all applications to completion for our PIN-based evaluation in Section 4.5.3.

4.5.2 Performance/Power Results and Analysis

In this subsection, we evaluate overall performance results and compare the impact of MME and ECC FIFO to that of MAXn schemes [64]. We analyze LLC power consumption in various coding schemes discussed in Section 4.4 and show the potential of saving LLC power consumption by off-loading the T2EC to DRAM. We evaluate the impact of additional DRAM traffic needed for off-loading the T2EC to DRAM, which only degrades performance if it competes with demand fetches and analyze sensitivity to DRAM bandwidth to assess the performance impact of MME and ECC FIFO on CMPs, where DRAM bandwidth is more scarce.

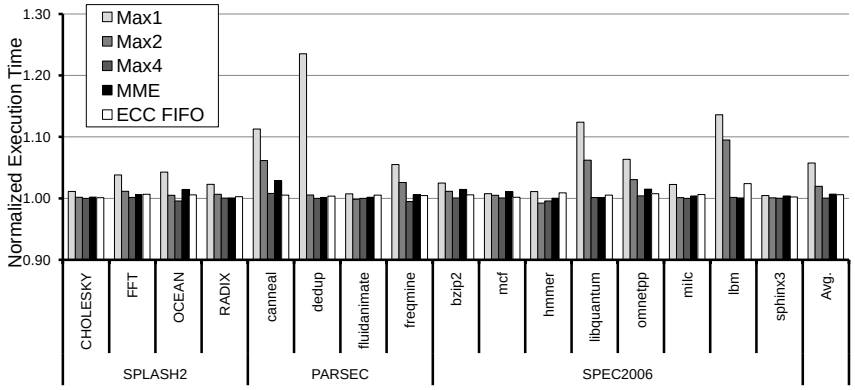


Figure 4.18: Normalized execution time.

4.5.2.1 Impact on Performance

Figure 4.18 compares the execution times of MAXn, MME, and ECC FIFO normalized to the execution time of the baseline configuration. As shown, the impact on application performance of both MME and ECC FIFO is minimal, with an average performance penalty of less than 0.7% both in MME and ECC FIFO and a maximum degradation of just 2.8% in MME and 2.3% in ECC FIFO. These results are encouraging because of the benefits our schemes provide in reducing on-chip area and leakage power by minimizing dedicated on-chip ECC storage. The MAXn technique, on the other hand, requires a significant tradeoff between performance loss and area gains. MAX1 averages over 5.7% performance loss with several applications experiencing $10 - 23\%$ degradation, MAX2 degrades 2% on average, while `libquantum` is significantly degraded by 6%, and MAX4 performs slightly better than MME and ECC FIFO. MAX4, however, requires much more area than MME and ECC FIFO as shown in Table 4.3.

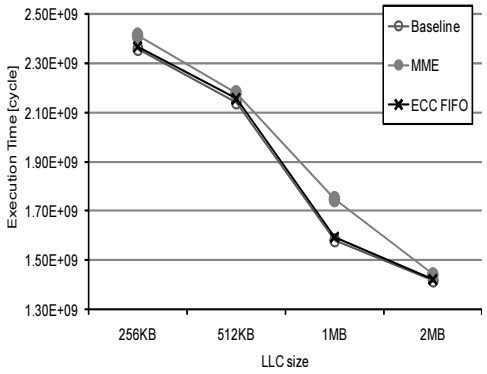


Figure 4.19: `OCEAN` 258 × 258 performance of the baseline, MME, and ECC FIFO with varying LLC size.

The anomalous behavior of `OCEAN`, `fluidanimate`, `freqmine`, and `hmmer`, where the performance of MAX2 and MAX4 is slightly better than baseline, is a result of increased eager write-backs for lines that are being forced clean in order to guarantee protection. We also note that applications with small working sets, such as `WATER-NSQUARED` (SPLASH2) and `blackscholes` (PARSEC) experience no performance drop with MME and ECC FIFO as well as MAXn, and we do not report their results.

Sensitivity Analysis. Note that the area / error protection tradeoff in MME is the same as that of ECC FIFO because both schemes use two-tiered protection and off-load T2EC to DRAM. MME, however, can impact performance significantly when the working set of an application closely matches the LLC size. Figure 4.19 compares the execution times of `OCEAN` with a 258 × 258 grid of baseline, MME, and ECC FIFO configurations as the LLC size is varied. As the LLC size grows from 512KB to 1MB, the baseline perfor-

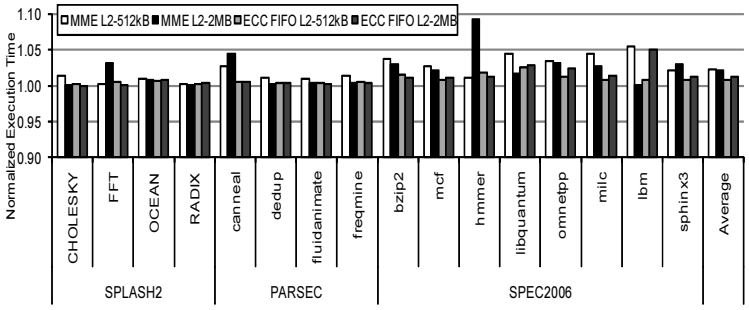


Figure 4.20: MME and ECC FIFO's sensitivity to L2 size.

mance is improved by 26%. ECC FIFO's performance is consistently less than 1% across all LLC sizes. MME, however, degrades performance significantly (10%) with a 1MB LLC since the effective LLC size is reduced by sharing the LLC between data and T2EC information. We also evaluate MME and ECC FIFO's sensitivity to L2 size in Figure 4.20. The relative performance degradations in 512KB and 2MB L2 caches are $1-2\%$ on average. MME's penalty is, however, sensitive to L2 size in applications such as `FFT`, `hmmer`, `libquantum`, and `lbm`, where performance penalty is still less than 5% except `hmmer` with 2MB. Compared to MME, ECC FIFO is insensitive to L2 size since the T2EC management bypasses the LLC.

In addition, we evaluated MME and ECC FIFO varying many system parameters: different eager write-back periods, an in-order core, and different L1 sizes. MME and ECC FIFO's performance degradation is consistently low (less than 2%) across all different configurations, showing that MME and ECC FIFO are not sensitive to system parameters.

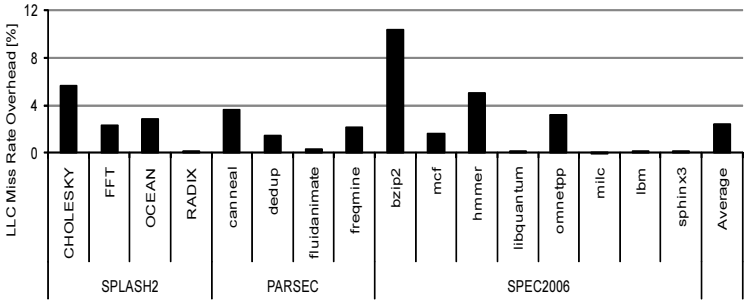


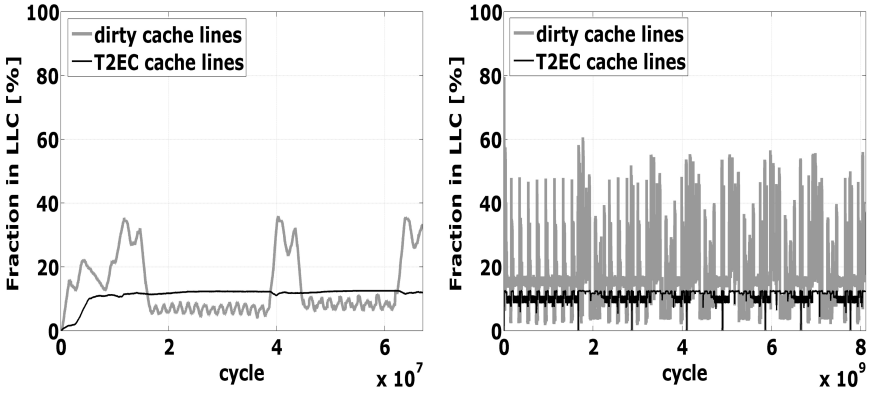
Figure 4.21: LLC miss rate overhead in MME normalized to the miss rate of the baseline

4.5.2.2 Impact of MME on LLC behavior

In this subsection, we analyze the impact of MME on the behavior of the LLC. Note that ECC FIFO bypasses the LLC to access T2EC information, and thus does not change the caching behavior and only impacts the operation of DRAM system.

Figure 4.21 shows that the sharing of LLC resources between T2EC information and data does not significantly impact the LLC data miss rate, which on average increases by only 2% and no more than 11% (`bzip2`).

Figure 4.22 illustrates the dynamic adjustment of T2EC lines to the number of LLC dirty lines in the `FFT` and `OCEAN` applications. In `FFT`, we can see the spatial locality of placing multiple T2EC words in a single cache line. Even as the number of dirty lines changes, there is no need to allocate additional cache lines for T2EC information. `OCEAN` demonstrates how MME transparently adjusts to the rapid changes in the number of dirty lines in the LLC.



(a) FFT (b) OCEAN

Figure 4.22: Fraction of dirty and T2EC cache lines over time.

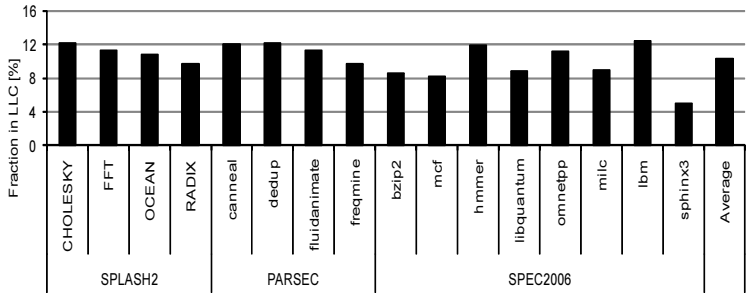


Figure 4.23: Average fraction of T2EC cache lines in the LLC.

Figure 4.23 shows that the average fraction of the LLC that is occupied by T2EC lines is roughly 10%. It is possible that we can reduce the total number of T2EC lines by changing the cache policy for T2EC lines, such as inserting T2EC lines in the least recently used (LRU) position. We do not pursue such techniques in this dissertation because the performance overheads of MME are already low.

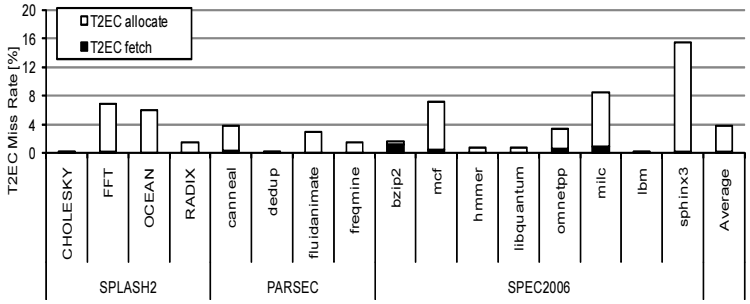
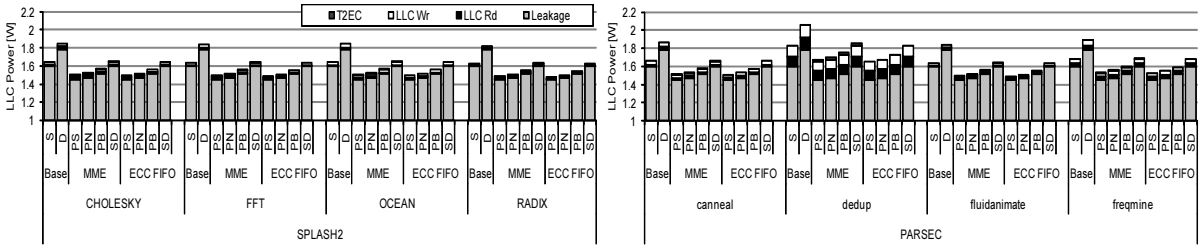


Figure 4.24: T2EC miss rate.

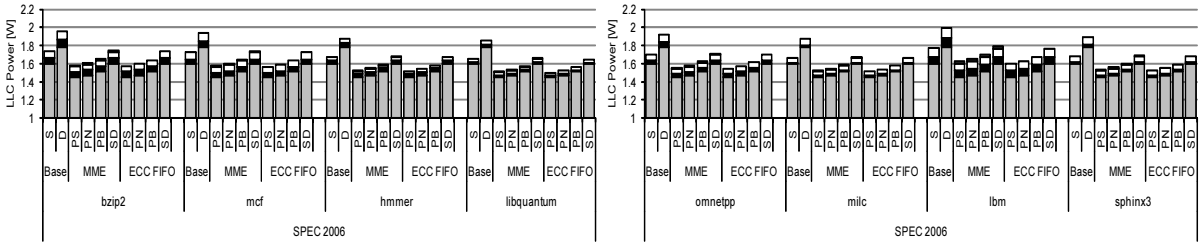
Figure 4.24 presents *T2EC miss rate*; the fraction of accesses to T2EC data not found in the LLC. *T2EC fetch* is the rate of T2EC fetches from DRAM and *T2EC allocate* is the fraction of fetches avoided when all addresses that map to the T2EC lines are clean (see Figure 4.3(b)). Note that nearly all T2EC fetches from DRAM are avoided on average. The low actual T2EC miss rate (T2EC fetch), less than 0.5% on average, indicates that T2EC caching effectively captures the locality of T2EC accesses.

4.5.2.3 Impact on LLC power

Figure 4.25 compares LLC power consumption of the baseline (S and D) and various coding schemes (PS, PN, PB, and SD) of MME and ECC FIFO estimated using CACTI 5 [119] for the cache parameters in Table 4.4 in 45nm process technology. In MME, PS saves 8.6% of LLC power consumption compared to S, while SD, PN, and PB consume 10.5%, 15.2%, and 17.5% less power than D, respectively. ECC FIFO achieves similar gains: PS saves 9.2% over S, SD, PN, and PB consume 11.1%, 15.8%, and 18.1% less than D, respec-



(a) SPLASH2 and PARSEC



(b) SPEC CPU 2006

Figure 4.25: LLC power consumption estimated using CACTI 5 for a 8-way 1MB LLC with 64B cache line.

tively. MME's power consumption is slightly higher than that of ECC FIFO since MME accesses the LLC to manage T2EC while ECC FIFO writes T2EC directly to the FIFO in memory via the coalesce buffer, leading more traffic increases as described in Section 4.5.2.4. Note that PS and SD provide similar error protection as S and D, respectively, and PN and PB can protect against even longer error bursts, even compared to D.

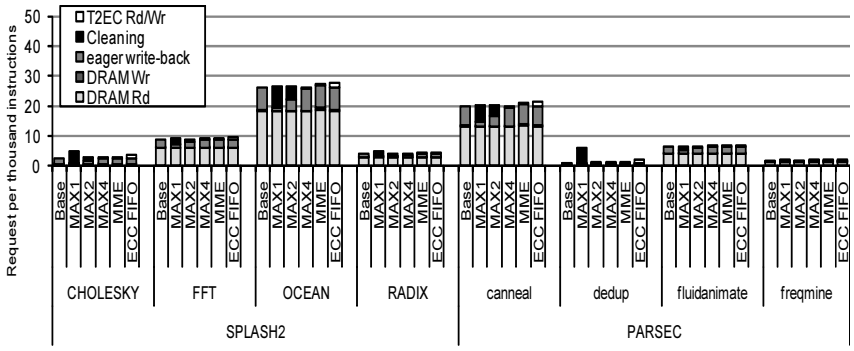
4.5.2.4 Impact on DRAM Traffic

Figure 4.26 shows the total DRAM traffic broken down into components of data reads and writes of LLC lines (*DRAM Rd* and *DRAM Wr*), eager write-backs of cache liens (*eager write-back*), MAXn writes for cleaning lines (*Cleaning*), and T2EC traffic of MME and ECC FIFO (*T2EC Rd*

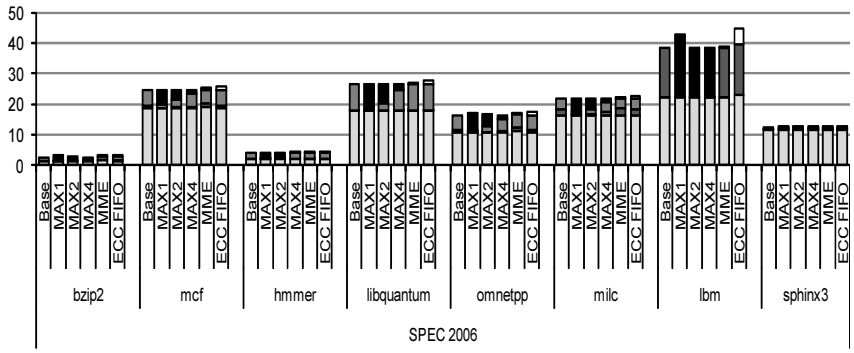
and *T2EC Wr*). Although MAXn does not increase overall memory traffic by much in most cases, the type of traffic and its timing differ significantly from the baseline. Unlike eager write-back traffic, which is scheduled during idle DRAM period [69], MAXn's cleaning writes compete with demand fetches and degrade performance as shown in Figure 4.18. Compared to the baseline, MME increases memory traffic by 2% and ECC FIFO increases by 9% on average. MME's low impact on memory traffic is because it takes advantage of locality in T2EC access by caching T2EC information. The relatively large traffic increase in ECC FIFO is, however, not critical to performance in that T2EC write-back is one-way traffic and can be scheduled during idle DRAM period.

4.5.2.5 Impact on DRAM Power

Although MME and ECC FIFO only marginally increase DRAM traffic, the increase in DRAM power consumption may cancel out MME and ECC FIFO's LLC power savings. To measure the actual DRAM power consumption in MME and ECC FIFO, we use a power model developed by Micron Corporation [6] that is embedded within DRAMsim. Figure 4.27 presents DRAM power consumption of baseline, MME, and ECC FIFO. On average, MME only increases DRAM power consumption by 40 mW (2.8%); this is much lower than MME's LLC power saving (147 mW in PS compared to S). In ECC FIFO, the increase in DRAM power (143 mW) is comparable to LLC power saving (152 mW in PS compared to S). ECC FIFO's LLC power savings,



(a) SPLASH2 and PARSEC



(b) SPEC CPU 2006

Figure 4.26: DRAM traffic comparison.

however, are much larger than DRAM power increase in other configurations: 331 mW in PN, 293 mW in PB, and 206 mW in SD compared to D. In addition, recent systems have even larger LLC size (e.g., 24MB); MME and ECC FIFO can potentially achieve much more power savings.

4.5.2.6 Multi-Core Considerations

We have so far discussed and evaluated MME and ECC FIFO in the context of a single core and now briefly discuss multi-core and multi-processor

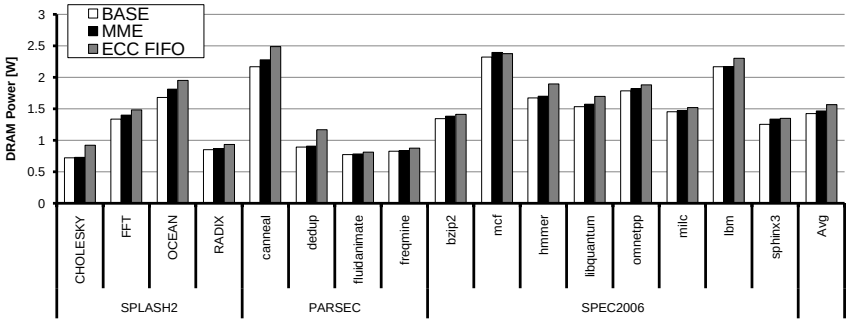


Figure 4.27: DRAM power consumption in Baseline, MME, and ECC FIFO.

implications. Both MME and ECC FIFO are easily integrated with any cache coherence mechanisms because the T2EC region of MME and T2EC FIFO in ECC FIFO are fixed and inherently private; the mapping is between physical cache lines and physical memory. A potential problem with MME and ECC FIFO is that the increased traffic due to T2EC accesses can hurt performance given the relatively lower memory bandwidth per core of a multi-core processor. We evaluate MME and ECC FIFO in a system with low DRAM bandwidth of only 2.667GB/s, which is half the bandwidth of the baseline system. As Figure 4.28 shows, the relative performance of MME and ECC FIFO is not sensitive to memory bandwidth. Compared to MME, ECC FIFO is slightly more degraded with the lower memory bandwidth; it is due to ECC FIFO's increase in traffic, whereas MME takes advantage of locality in T2EC accesses by caching it.

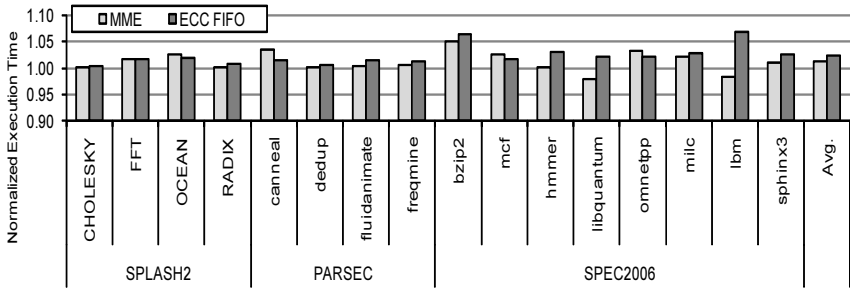


Figure 4.28: Normalized execution time (2.667 GB/s DRAM BW).

4.5.2.7 Larger LLC with MME and ECC FIFO

Our evaluation of MME and ECC FIFO assume the same LLC capacity across all configurations, showing LLC area and power savings. We can, however, design a larger LLC using the saved area in MME and ECC FIFO. Though it is not a fair comparison (neither same area nor same power budget), we evaluate MME and ECC FIFO with a 9-way set associative 1.1MB LLC. This LLC, 10% increase in data capacity, requires comparable on-chip area, though not exactly same, to the baseline with uniform ECC. Most applications are not sensitive to this 10% increase in LLC size; hence, MME and ECC FIFO with the larger LLC perform only slightly better than those with 1MB LLC, but the performance of `mcf` is improved by more than 2%, even compared to the baseline with 1MB LLC. We believe that the saved on-chip area in MME and ECC FIFO, especially in a system with a large LLC, can give more opportunity to improve overall system energy efficiency.

4.5.3 PIN-based Emulation

Full-system cycle-based simulations can provide accurate performance prediction and measurements, but are very slow, limiting the number and lengths of experiments. Hence, we simulated small to medium problem sets with the SPLASH2 and PARSEC benchmarks, and only representative regions of SPEC applications. To understand how larger datasets affect MME and ECC FIFO, we use PIN [77] to qualitatively evaluate LLC behavior for applications with larger datasets: tk29.O for `CHOLESKY`; 4M samples for `FFT`; a 1026×1026 grid for `OCEAN`; 8M samples for `RADIX`; the `simlarge` inputs for all the PARSEC applications; and the complete SPEC CPU 2006 application runs. We implement eager write-back, MAXn, MME, and ECC FIFO schemes in a two-level cache hierarchy PIN tool. Note that our PIN-based emulation is only used to collect statistics on LLC behavior and does not utilize a timing model, an instruction cache, stall and re-try due to finite resources such as MSHRs, or a DRAM model.

The results are presented in Figure 4.29 and follow similar trends to those seen with cycle-based simulation (Figure 4.26). The most significant difference is that the traffic overhead of MME is smaller when larger datasets are used, and we expect to have an even smaller performance degradation than the 0.7% average experienced with small datasets (Section 4.5.2.1). We repeated the experiments, while changing the LLC size between $1 - 16$MB, and the results and trends are similar to those reported in Figure 4.29.

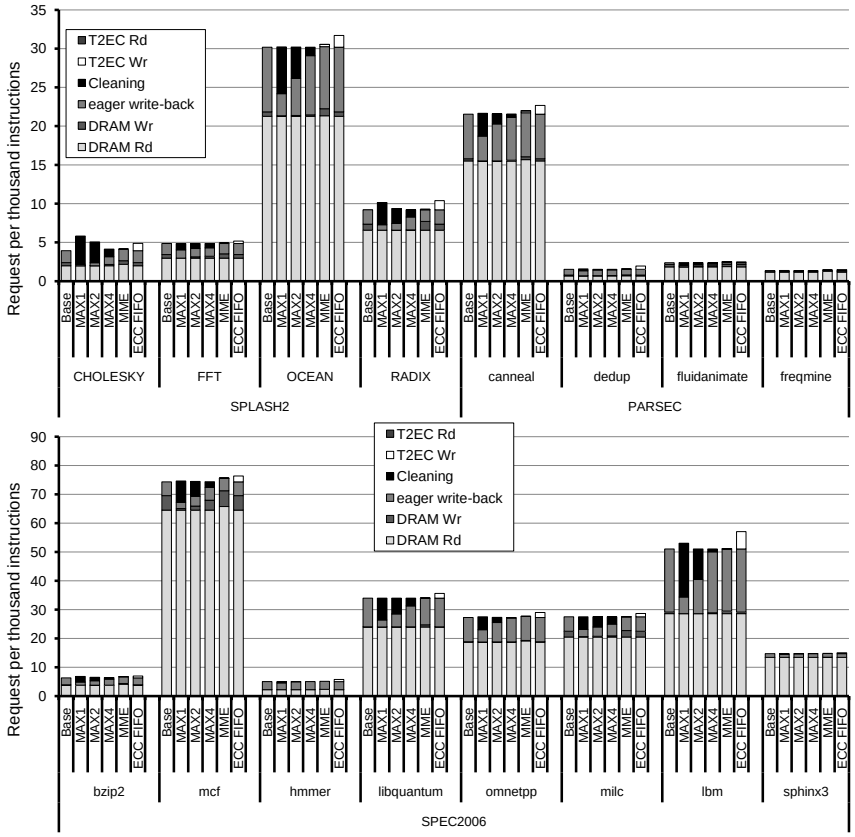


Figure 4.29: PIN-based emulation.

4.6 Related Work

Prior work on reducing on-chip ECC overhead generally breaks the assumption of uniform protection of all cache lines and either sacrifices protection capabilities, when compared to uniform ECC protection, or utilizes different mechanisms to protect clean and dirty cache lines.

Parity caching [65] compromises error protection to reduce area and energy costs of ECC by only protecting most recently used cache lines in every

cache set. Selective data protection [70] implements two cache arrays: one array uses uniform ECC and stores critical data; and the other one, without ECC, is for non-critical data. Another scheme in this first category is In-Cache Replication (ICR) [138]. ICR increases error protection for a subset of all cache lines, which are accessed frequently, by storing their replicas in place of cache lines that are predicted to be "dead" and no longer required. Not all cache lines are replicated leading to a potentially higher uncorrectable error rate than with the baseline uniform ECC.

An approach to save energy and latency rather than area was proposed in Punctured ECC Recovery Cache (PERC) [99]. The PERC architecture decouples error detection from error correction and utilizes a low-latency and low-cost parity for reading, while traditional ECC is computed and stored on every write. A similar idea was taken further in [71], where the SEC ECC portion of clean cache lines is power-gated to reduce leakage power, leaving only parity active.

The area efficient scheme [64] is a method to decrease area in addition to energy by trading-off performance. The idea is to allow only one dirty cache line per set in a 4-way set associative cache. If a larger number of lines in a set require ECC (more than one dirty line), a write-back is forced to make space for the new ECC. In our experiments that accurately model the DRAM system (Section 4.5), we found that this additional memory write traffic can significantly degrade performance. The performance impact can be reduced at an area and energy cost if more ECC-capable lines are provided in each

set. We refer to this generalization scheme where n lines per set have ECC as MAXn.

Other work proposes a fully-associative *replication cache* (R-Cache) that utilizes replication to provide error protection [137]. The R-Cache, however, increases energy consumption and can be used only with small L1 caches due to its fully associative structure.

Multi-bit tolerating 2D error coding [63] extends 2-dimensional parity checking [28, 62] and can tolerate multi-bit errors in a scalable way. Interleaving horizontal and vertical parity codes together provides higher error protection with only a modest increase in ECC storage. 2D coding, however, forces every write being read-modify-write.

In recently proposed low-V_{CC} caches, tolerating multi-bit errors is much more important. Word disabling and bit fix [127] trade off cache capacity for reliabilty in low-V_{CC} operation. These techniques result in 50% and 25% capacity reductions, respectively. Multi-bit Segmented ECC (MS-ECC) [35] uses Orthogonal Latin Square Codes (OLSC) [54] that can tolerate both faulty bits in low-V_{CC} and soft errors, sacrificing 50% of cache capacity. Other research [10] studies performance predictibility of low-V_{CC} cache designs using subblock disabling.

Finally, eager write-back is proposed as a way to improve DRAM bandwidth utilization [69]. It eagerly writes dirty cache lines back to DRAM so that dirty evictions do not contend with demand fetches. Many cache relia-

bility studies (e.g., [64, 71]), including ours, use eager write-back as a way to reduce the average number of dirty cache lines in the LLC.

4.7 Summary

This chapter presents novel architectures, Memory Mapped ECC and ECC FIFO, that provide strong error protection for a last-level cache with minimal hardware and performance overheads. Both architectures leverage two-tiered error protection that minimizes the need to dedicate SRAM resources to maintain ECC information by off-loading the overhead of strong ECC codes to memory namespace, achieving both low cost and high reliability in an LLC.

MME places T2EC information within the memory hierarchy and reusing existing cache storage and control, while ECC FIFO uses a simple FIFO structure that does not affect LLC caching behavior. With the minimized dedicated on-chip ECC storage, both the leakage power and energy per access are reduced leading $9-18\%$ LLC power reduction in addition to $15-25\%$ of area saving, while performance is degraded only by 0.7% in both MME and ECC FIFO on average and no more than 2.8% (MME) and 2.3% (ECC FIFO). We have also shown that the probability of T2EC unprotected due to FIFO overwrite in ECC FIFO can be managed to be very small with a reasonably sized FIFO.

Chapter 5

Flexible Main Memory Protection

Applications in all computing segments require ever-larger main memory capacities, which increases the cost of memory systems. This is particularly true for commercial servers, supercomputers, and other shared environments. One aspect that drives the cost up is the need to tolerate errors and faults in DRAM with minimal impact on performance, efficiency, and availability. Recent trends show an increased relative likelihood of entire DRAM chips malfunctioning [38, 101], which necessitates more stringent protection than just tolerating single-bit errors, as discussed in Chapter 2. To make matters worse, this stringent protection is needed in systems that are increasingly sensitive to energy and cost, such as large-scale servers.

In this chapter, we explore cooperative operating-system and hardware techniques that maintain or improve current error tolerance levels, while reducing cost and energy. We propose *Virtualized ECC* (V-ECC) [133, 134] that virtualizes the storage of redundant information, using mechanisms that are similar to virtual memory management [105]. We explore mappings in which some or all of the redundant information shares the same physical address space as the data it protects. This approach gives us great flexibility in ac-

cessing and manipulating ECC information and enables a single system design to be tuned and adapted to a particular usage scenario, or even a particular application or data array.

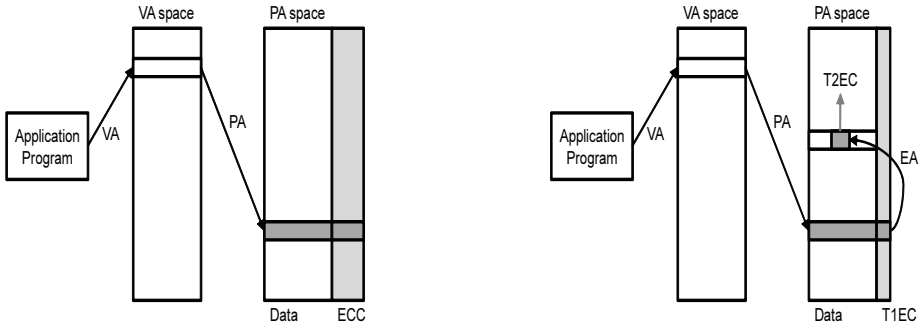
To demonstrate the potential of the V-ECC approach, we describe schemes ranging from low-cost protection, which uses Non-ECC DIMMs, up to double chipkill-correct techniques for high-availability systems. We also show how ECC virtualization enables us to maintain protection capability even when varying the DRAM configuration between $\times 4$, $\times 8$, and $\times 16$ chips. We evaluate each configuration in detail and discuss its impact on performance and improvements in power consumption and *energy-delay product* (EDP). Performance is degraded because effective data bandwidth is lower when the same pins are shared to transfer both data and redundant information. Our evaluation uses applications from the SPEC CPU 2006 [109] and PARSEC suites [21] that have high memory access demands, as well as targeted micro-benchmarks. V-ECC with ECC DIMMs improves system EDP by 12%, and the performance is hardly impacted, dropping by only $1 - 2\%$. V-ECC with Non-ECC DIMMs, on the other hand, degrades performance by $3 - 9\%$ on average and no more than $10 - 24\%$, while improving system EDP by up to 12% when using $\times 8$ and $\times 16$ DRAMs.

The remainder of this chapter is organized as follows: Section 5.1 develops the V-ECC architecture, Section 5.2 presents evaluation results, Section 5.3 describes related work, and Section 5.4 summarizes this chapter.

5.1 V-ECC Architecture

The main innovation of V-ECC is that it allows great flexibility in choosing the protection method and level, even dynamically. The idea is to enable the tuning of the protection scheme based on the needs of a specific system configuration, changes in environmental conditions, or potentially different requirements of different applications or data. V-ECC offers an opportunity to tailor memory protection levels and avoid the need to uniformly pay the overhead for worst-case scenarios [126]. We start by giving a high-level overview and then go into the details of the various components, including the interaction with the cache hierarchy (Section 5.1.1), examples of possible protection techniques (Section 5.1.2), and the OS virtual memory interface (Section 5.1.3).

There are two basic mechanisms underlying V-ECC: an augmented *virtual memory* (VM) interface that allows a separate virtual-to-physical mapping for data and for its associated redundant ECC information; and a generalization of DRAM ECC into a two-tiered protection mechanism that is explained in Section 3.1; a *tier-one error code* (T1EC) is used to detect errors on every access and a *tier-two error code* (T2EC) is only needed when an error is actually detected [99, 131, 132]. Figure 5.1 compares traditional VM, with its fixed relation between data and ECC, and the decoupled two-tier approach of V-ECC. Traditional VM (Figure 5.1(a)) translates a virtual address from the application namespace to a physical address in DRAM. A DRAM access then



(a) Conventional architecture (b) V-ECC architecture

Figure 5.1: High-level view of memory accesses in a conventional virtual memory with uniform ECC, and V-ECC with a two-tiered flexible protection scheme.

retrieves or writes both the data and the ECC information, which is stored aligned with the data in the dedicated ECC DRAM chips.

Figure 5.1(b) gives an example of a flexible mapping enabled by V-ECC, in which a portion of the redundant information, the T1EC, is aligned with the data, but the T2EC part is mapped to a different physical address that shares the same namespace and storage devices as the data. The OS and hardware *memory management unit* (MMU), maintain the pair of mappings and ensure that data and ECC are always matched and up to date (Section 5.1.3). Thus, less total data is accessed on a read in V-ECC than in the conventional approach, because T2EC is only touched on the very rare event of an error.

Data writes, however, may have higher overhead in V-ECC, because the ECC data needs to be updated; hence, requiring a second DRAM access, and the reduced system cost comes at a potential performance overhead. To

mitigate this detrimental effect, we propose to utilize the processor cache to reduce the amount of ECC traffic and discuss this in detail in the following subsection. Another advantage of the decoupled mapping and two-tiered approach is that different memory pages can have different protection types. For example, clean pages do not require any T2EC storage, and thus the overall degree of redundancy in the memory system can be adjusted dynamically, increasing the effective memory capacity.

5.1.1 Cache-DRAM Interface

The cache filters requests from the core to the DRAM system and can also help in improving the performance of V-ECC. Because we store redundant information in the same physical namespace as data, we can cache ECC information on the chip and improve ECC access bandwidth using the same principles that make caches advantageous for data.

Unlike application data, however, ECC is only accessed by the memory controller when it needs to address off-chip DRAM and is not shared among multiple processor cores. Thus, the redundant information is stored in the cache level to which the memory controller has direct access – the *last-level cache* (LLC) bank to which it is attached. Because of this arrangement, ECC information does not participate in any coherence protocol and is kept up to date by the memory controller.

V-ECC does not require significant changes from the existing cache interface, with the additional hardware being the ECC address translation

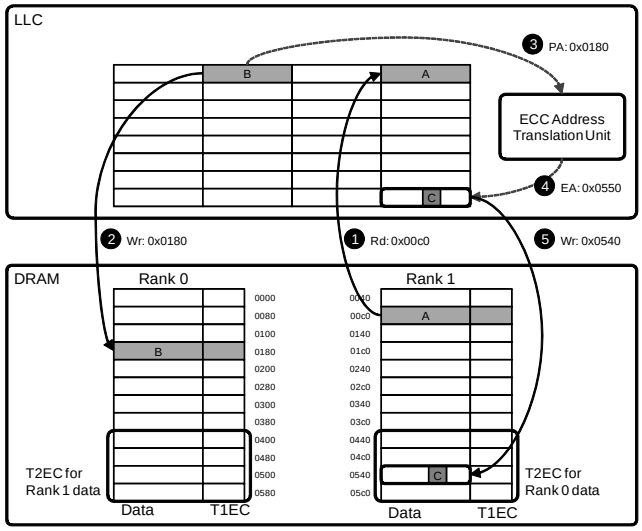


Figure 5.2: Operations of DRAM and LLC for accessing a two-tiered V-ECC configuration.

unit (described in Section 5.1.3) and the ability to maintain and write back partially valid cache lines. The latter property is necessary because the cache has up to date ECC information only for data that is generated on-chip (in two-tiered V-ECC). We describe this property in greater detail below and address the different operations needed for two-tiered protection and for implementing ECC with Non-ECC DIMMs.

5.1.1.1 Two-Tiered V-ECC

Figure 5.2 shows our two-tiered V-ECC on top of a generic memory system configuration with a last-level cache connected to two ranks of DRAM with dedicated ECC chips. We use ECC chips to store a T1EC code, which can detect all errors of interest, but cannot correct them without the additional

information of the T2EC. The T2EC is mapped to the data DRAM chips such that data and its associated T2EC are in two different DRAM ranks.

Circled numbers in the text below refer to operations shown in Figure 5.2. Handling a fill into the LLC on a cache miss follows the same operations as in a conventional system; a data burst and its aligned T1EC are fetched from main memory, and error detection is carried out (①). The difference from a conventional system is that any detected errors cannot be immediately corrected.

Evicting a dirty line and writing it back to DRAM (②), however, requires additional operations when compared to a conventional hierarchy. The memory controller must update the T2EC information associated with the evicted line, which starts with translating the data address to the location of the *ECC address* (EA) (③ and ④). If the translated EA is already in the LLC, it is simply updated in place. [1] Otherwise, we allocate an LLC line to hold the T2EC. We do not need to fetch any information from memory, because T2EC is only read when an error is detected. Any writes, render the prior information obsolete; thus, we compute the new T2EC and write into the LLC along with a mask that indicates what portion of the LLC line contains valid T2EC information.

[1] This requires a partial write to a LLC line. If the LLC is a sector cache [75], updating a part of an LLC line is inherently supported. Otherwise, we can either perform read-modify-write operations or modify the LLC to support partial writes.

As we explain later, our coding schemes use T2ECs that are $16 - 128$ bits long, and thus require very few additional valid bits. Depending on the exact ECC used to protect the LLC itself, it may even be possible to re-purpose the LLC ECC bits to store the valid mask. We can ignore errors in a T2EC line in the LLC because there is no need to add a third level of redundancy and protect T2EC information from errors. The errors on the valid mask can be tolerated by the ECC for tag and meta-data (valid, dirty, and other bits). This mask is used when a T2EC LLC line is evicted back to DRAM (⑤) as invalid portions of the line must not overwrite T2EC data in DRAM. To do so, we extend each MSHR entry with a valid bit vector, which the DRAM controller uses for communicating the write mask to the DRAM chips [111, 112].

When an error is detected by T1EC, which can only happen upon a read, the correction is carried out using the corresponding T2EC. If the T2EC is not in the cache, correction requires an additional DRAM access to fetch the redundant information. The additional latency, however, does not impact performance because errors in a particular memory channel are very rare. We can also employ, rather complex, software error correction handlers.

Very frequent errors indicate a hard-fault and can be mitigated by data migration, as suggested by Slayman [107].

5.1.1.2 V-ECC Interface with Non-ECC DIMMs

Even if physical memory does not provide ECC storage, we can use V-ECC to protect memory against errors. In the Non-ECC DIMM configura-

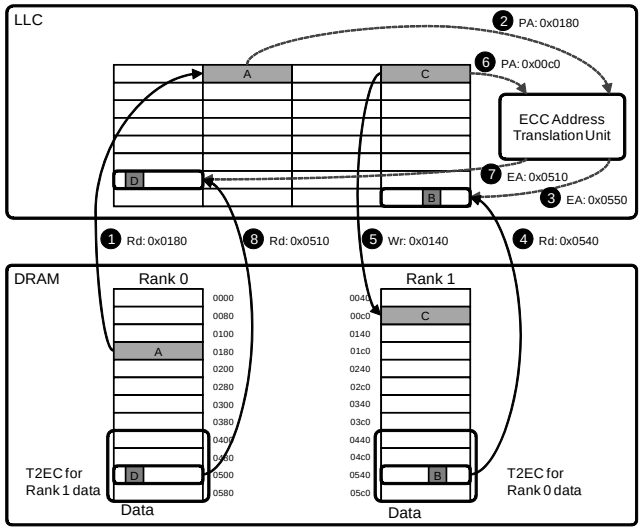


Figure 5.3: Operations of DRAM and LLC for accessing V-ECC in a Non-ECC DIMM configuration.

tion, we cannot store an aligned T1EC and, instead, place all the redundant information in the virtualized T2EC (we still refer to this as T2EC to keep the notation consistent).

The cache and memory behavior for this scheme is shown in Figure 5.3 which the circled numbers below refer to. When data is read from main memory (①), we use the ECC address translation unit to find its EA (② and ③). A T2EC LLC miss will fetch the T2EC from main memory (④), because without ECC DIMMs, the information is required to detect errors and not just for correction. Unlike the two-tiered scenario, we fetch an entire cache-line's worth of T2EC data on a miss to amortize the DRAM access, and expect spatial locality to reduce the cost of following memory accesses. We only return data to the cache controller after the ECC information is fetched and the data

is verified.[2] On a dirty write-back (⑤), the PA is translated to an EA (⑥ and ⑦), and a T2EC is fetched from main memory if the EA is not already cached (⑧), again, expecting spatial locality to amortize this access cost.

5.1.2 V-ECC Protection Schemes

We now discuss possible DRAM configurations for V-ECC, assuming a memory system that is representative of servers that require chipkill-correct level protection. The baseline memory system is composed of a 128-bit wide DDR2 DRAM channel with an additional 16 bits of dedicated ECC as described in Chapter 2. We use DDR2 in this study because of a readily available power model [123]. Our techniques work as well, or better, with DDR3 [112] because it uses longer bursts and limits use of traditional chipkill techniques [13].

In the rest of this subsection we describe memory protection mechanisms that are enabled by V-ECC, which maintain the storage overhead and access granularity of chipkill and can even increase protection guarantees.

5.1.2.1 V-ECC with ECC DIMMs

While traditional chipkill-correct uses a 4-check-symbol code as explained in Chapter 2, V-ECC, with two tiered protection and T2EC storage virtualization, enables a more efficient 3-check-symbol code [34]. In two-tiered

[2]Speculative data forwarding is possible, but did not significantly improve performance in our experiments as shown in Section 5.2.2.3.

error protection, the first two check symbols of the 3-check-symbol code construct a T1EC that can detect up to 2 symbol errors, while the T2EC is the third check symbol of the 3-check-symbol code. If the T1EC detects a single symbol error, it is corrected using all 3 check symbols of both tiers. Our scheme uses 8-bit symbols for $\times 4$ and $\times 8$ DRAM configurations and 16-bit symbols with $\times 16$ chips (alternatively, we can use a 2-way interleaved 8-bit symbol ECC to reduce the complexity of GF(2^{16}) arithmetic). In the $\times 4$ system, we use two consecutive transfers of 128 bits so that we have an 8-bit symbol from each DRAM chip for the 8-bit symbol based ECC code. This effective 256-bit access does not actually change the DRAM access granularity, which is still 64B as in the baseline DDR2-based chipkill system.

The other key point is to use V-ECC to reduce the system cost, or to increase error tolerance to two malfunctioning chips, rather than just one. We describe these mechanisms for $\times 4$ and $\times 8$ configurations below, and discuss $\times 16$ cases in the next subsection. The configurations are summarized in Table 5.1, which also presents the details of the baseline chipkill technique and mechanisms that rely on modified DIMMs and higher-overhead codes to improve access granularity and power [13].

ECC $\times 4$ uses $\times 4$ chips, but utilizes the two-tiered approach to improve energy efficiency. We store two 8-bit check symbols in 2 ECC DRAM chips (in an ECC DIMM) that serve as a T1EC that can detect chip errors (up to

[3]Non-ECC configurations of V-ECC cannot detect 2 chip failures.

Table 5.1: DRAM configurations for chipkill-correct of the baseline system, MC-DIMMs that are non-standard DIMMs introduced in [13] to improve DRAM power consumption, and V-ECC.[3]

	DRAM type	# Data DRAMs per rank	# ECC DRAMs per rank	Rank organization	T2EC access		T2EC per cache line
					on read	on write	
Baseline Chipkill Correct							
Baseline x4	×4	32	4	2 ECC DIMMs	N	N	N/A
MC-DIMM [13] with 128bit channel and 4 rank subsets							
MC-DIMM x4	×4	32	12	2 MC-DIMMs	N	N	N/A
MC-DIMM x8	×8	16	12	2 MC-DIMMs	N	N	N/A
MC-DIMM x16	×16	8	12	2 MC-DIMMs	N	N	N/A
V-ECC							
ECC ×4	×4	32	2	1 ECC DIMM and 1 Non-ECC DIMM	N	Y	2B
ECC ×8	×8	16	2	2 ECC DIMMs	N	Y	4B
Non-ECC ×4	×4	32	N/A	2 Non-ECC DIMMs	Y	Y	4B
Non-ECC ×8	×8	16	N/A	2 Non-ECC DIMMs	Y	Y	8B
Non-ECC ×16	×16	8	N/A	2 Non-ECC DIMMs	Y	Y	16B

2 chip failures). The third check symbol is the T2EC, which is stored in the data chips.[4] Thus, ECC ×4 only requires 2 ECC chips instead of the 4 chips of the conventional approach, saving 8 pins and associated costs of storage, power, and bandwidth.

ECC ×8 is an efficient scheme for chipkill protection using ECC DIMMs with ×8 chips. We use the two ECC chips in a rank for the 2-symbol T1EC and store the third check symbol in data memory as the T2EC. Thus we access 16 ×8 data chips and two additional ECC chips on every read, for the same 64-byte access granularity and redundancy overhead of the conventional chipkill approach. Without virtualizing T2EC, an additional DRAM chip to hold the third symbol would be touched on every access, increasing power and

[4]We omit the full details of the error correction coding scheme, such as the parity check matrix and syndrome description

pin requirements and redundancy to a fixed 18.5% [13], as well as requiring non-standard DIMMs.

The above two-tier chipkill schemes can further be extended by adding a second check symbol to T2EC; if T1EC detects two chip errors (two erroneous symbols) the combined 4 check symbols from both tiers can be used to tolerate two bad chips. The details of this approach are summarized in the double-chipkill column of Table 5.2). While two simultaneous bad chips are unlikely today, future energy-saving techniques may require such a protection level because of the following two trends: Transient errors that manifest as dead chips are growing in likelihood [38]; and energy-efficient server systems that increasingly operate with narrower noise margins due to reduced cooling and voltage levels.

5.1.2.2 V-ECC with Non-ECC DIMMs

Another advantage of V-ECC is the ability to add ECC protection to systems that use Non-ECC DIMMs. We suggest schemes that are based on a 2-check-symbol Reed Solomon (RS) code [94], which can detect and correct one symbol error – in our case, a code that can tolerate any number of bit errors as long as they are confined to a single chip.

The details for this type of scheme using $\times 4$, $\times 8$, and $\times 16$ DRAM chips are also summarized in Table 5.1. All three Non-ECC DIMM configurations have the same protection capability, but the access properties differ. The wider symbols needed for $\times 16$ DRAMs imply that fewer T2EC words fit into

Table 5.2: V-ECC configurations for flexible protection achieved by varying T2EC size.[5]

	T2EC size			
	No Protection	chipkill detect	chipkill correct	double-chipkill correct
ECC ×4	N/A	0B	2B	4B
ECC ×8	N/A	0B	4B	8B
Non-ECC ×4	0B	2B	4B	8B
Non-ECC ×8	0B	4B	8B	16B
Non-ECC ×16	0B	8B	16B	32B

an LLC line. Recall that unlike the two-tiered techniques with ECC DIMMs, both LLC write-backs and demand fetches require the DRAM controller to access both the data in DRAM and the T2EC information to perform error checking and correcting. The T2EC is cached and may not require a second DRAM access, but the different symbol widths used result in different caching behavior (see Section 5.2.2.1).

5.1.2.3 Flexible Protection Mechanisms

An exciting feature of V-ECC is that it enables flexibility in choosing and adapting the error protection used based on dynamic application, user, and system needs. A single hardware with virtualized T2EC can support different levels of error tolerance by varying the T2EC size (Table 5.2). Supporting flexible protection tuning requires that the memory controller be able to compute and decode different codes, as well as a way to identify which ECC technique to use for any given DRAM access. An elegant method to achieve

[5] *Chipkill-detect* and *chipkill-correct* can detect up to 2 chip failures in ECC configurations, but they can detect only 1 chip failure in Non-ECC configurations.

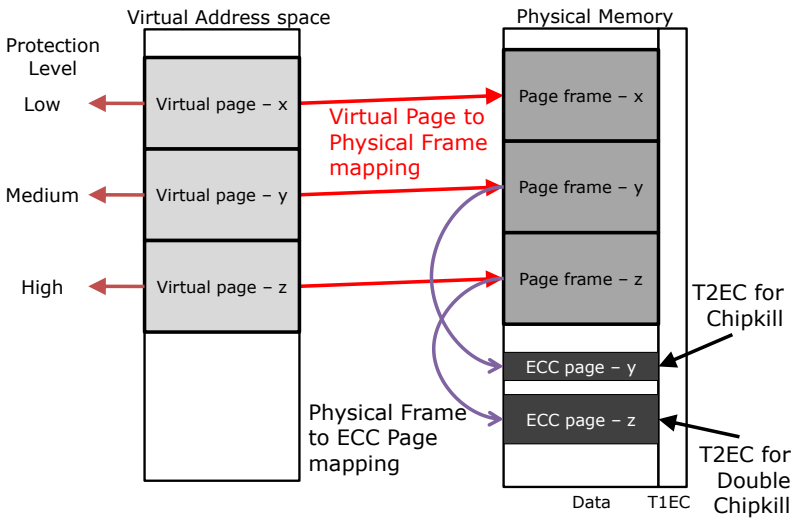


Figure 5.4: Adaptive protection example.

the latter is to augment the OS virtual memory page table and hardware TLB to include protection information for each page.

Figure 5.4 illustrates an example of per-page adaptive protection with V-ECC. A physical frame "x" is not associated with an ECC page; hence, we can only detect errors in this page using the T1EC. Physical frames "y" and "z", on the other hand, are associated with ECC pages, and errors detected by the T1EC can be corrected by using the T2EC. Furthermore, the physical frame z is associated with stronger T2EC. This costs more storage and bandwidth, but provides higher error tolerance level, double chipkill-correct. For instance, we can only detect double chip-failures in physical frames y, but can correct such failures in physical frame z.

We do not explore the benefits of protection tuning in this dissertation, but list two potential scenarios that we will explore in future work. The first is an opportunity to reduce system power by protecting more critical data with stronger codes and potentially leaving some data unprotected. Another potential use is to adapt the protection level to changes in environmental conditions, such as higher temperature or a higher energetic particle flux (while in an airplane, or if the system is located at high altitude). V-ECC offers opportunities for new tradeoffs, such as the double-chipkill technique described above or reduced power consumption by turning ECC off.

5.1.3 Managing T2EC Storage in DRAM

In this subsection, we discuss how the OS manages T2EC storage in DRAM and how ECC address translation is performed to retrieve the T2EC associated with a particular address. We present a solution that is based on current virtual memory approaches, but other alternatives are possible.

5.1.3.1 T2EC Allocation

Managing DRAM storage is the responsibility of the OS, and we extend the OS memory manager to account for T2EC information as well. Any time the OS allocates a new physical page, the OS also allocates a T2EC section that is large enough to accommodate redundant information for the entire data page. The size of this T2EC data depends on the protection configuration chosen for the newly allocated page. Note that only dirty pages require T2EC

storage in two-tiered protection, because clean pages can be recovered from secondary storage if T1EC detects an error. Many modern OSs already only allocate a physical page when the virtual page is first written (copy-on-write allocation policy), and hence, will inherently only allocate T2EC for dirty pages.

The T2EC allocation algorithm can follow the same approach as the data memory allocator of the OS, such as using a slab or buddy allocator [105], but at a finer granularity than data pages because the T2EC storage overhead is a fraction of the data size. Within a T2EC section, EAs are mapped to their associated data PAs such that two paired addresses are always on different ranks. This level of mapping is handled by the memory controller when it accesses T2EC data (see Section 5.1.3.2). Accounting T2EC increases the overhead of allocating a physical page, but prior work has shown that augmenting a copy-on-write allocator has negligible impact on application performance [12].

The OS is also responsible for freeing T2EC space when it is no longer needed for data protection. This happens when a data page is freed by the application or when a page is swapped out to secondary storage. For the two-tiered schemes, T2EC can also be freed when a data page is preemptively cleaned and written back to secondary storage even if it is still stored in DRAM, because it is then inherently redundant in the system.

One caveat of storing T2EC information in data memory is that it competes with data and decreases the effective memory capacity (albeit, while reducing overall cost). This may impact application performance if the sharing

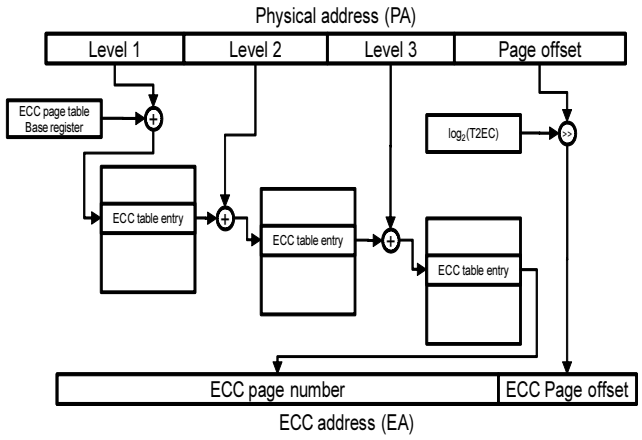


Figure 5.5: PA to EA translation using a three-level hierarchical ECC table mapping, similar to virtual memory address translation.

of space increases page replacement when the working set of applications is large. A possible approach to mitigating this problem is to spill T2EC memory sections to secondary storage to dynamically adjust the memory overhead of ECC. Again, parallels to conventional virtual memory management can be drawn, where data pages can be proactively cleaned and moved to secondary storage. In this dissertation, however, we always maintain active T2EC data in DRAM.

5.1.3.2 ECC Address translation

When the memory controller needs to access T2EC information, it translates the data physical address (PA) into its associated T2EC ECC address (EA). The translation requires information from the OS about the T2EC mapping, and we follow the same techniques as virtual to physical address translation. We maintain a PA to EA translation table in the OS in addition

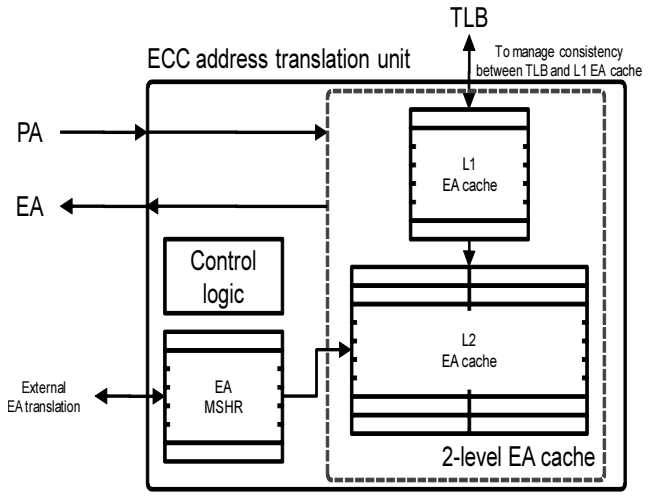


Figure 5.6: ECC address translation unit with a two-level EA translation cache.

to the VA to PA page table. Figure 5.5 shows an example of a hierarchical PA to EA translation table that uses the same translation scheme as an x86 virtual page table, where each translation entry points to the starting physical address of the T2EC region associated with the data page.

It is also possible to use other translation structures such as those suggested in prior work on maintaining memory metadata [128]. Note that the OS only manages a single translation table because the translation is from physical addresses. The hardware then computes an offset within the T2EC region based on the T2EC size and rank interleaving.

To reduce the overhead of translations, we also employ a hierarchical TLB-like acceleration structure in the ECC address translation unit, as shown in Figure 5.6. The L1 EA cache is maintained with the processor's TLB such

that the EA translation entries are filled and evicted in conjunction with the TLB. This guarantees that PA to EA translation on a read miss will always hit in the L1 EA cache, because a VA to PA translation was performed to access DRAM. Updating the EA cache increases the overhead of a TLB fill, because a second page walk is required to fetch the EA translation information. This overhead is small because of the low EA cache miss rate, which we report in Section 5.2. Note that EA translation is only required when there is a miss in, or write-back from, the LLC and is much less frequent than a virtual address translation.

To reduce the overhead even further, a second level EA cache can be used to store victim translation of L1 EA cache evictions. A write-back, in general, does not have as much locality as read operations, but the L2 EA cache can make the EA translation overhead trivial; write-back addresses had been translated (VA to PA translation) so the EA translation entries for writes are likely to be kept in either the L1 EA cache or a reasonably large L2 EA cache. We evaluate the effectiveness of the TLB-like EA cache in Section 5.2.2.2. The EA translation unit also contains MSHRs to allow concurrent overlapping translations to occur. Both levels of translation cache need to support invalidations from the OS when T2EC data is freed or re-mapped.

5.2 Evaluation

We evaluate V-ECC using a combination of detailed cycle-based simulation to determine performance and power impact and the PIN binary instru-

Table 5.3: Simulated system parameters.

Processor Core	SPARC V9 ISA 4-wide superscalar (2GHz)
L1 Cache	split I/D caches each 32KB 2-way set associative 64B cache lines write-back 1 cycle latency
L2 Cache	a unified 1MB cache 8-way set associative L1 exclusive 64B cache lines 12 cycle latency
DRAM	single-channel 4-rank DDR2 DRAM 800MHz 128-bit data bus (12.8GB/s) 16bit SSC-DSD ECC for ×4 chipkill correct open page, Read and Instruction Fetch First XOR based rank/bank interleaving [139] is used

mentation tool [77] to collect statistics on T2EC OS management and caching behavior for applications with large datasets. For the cycle-based simulation, we use the Simics full system simulator [78] with the GEMS [80] toolset: the OPAL out-of-order processor model to simulate a SPARC V9 4-wide superscalar core and the Ruby two-level exclusive write-back cache hierarchy model. We integrate DRAMsim [123] with GEMS to accurately account for the impact of V-ECC on memory bandwidth, performance, and power. Table 5.3 describes the baseline system configuration, which uses uniform ECC for chipkill-correct with ×4 DRAMs (as explained in Chapter 2).

We estimate DRAM power consumption using a power model developed by Micron Corporation [6] that is embedded within DRAMsim. For processor power analysis, we use WATTCH [26] to model out-of-order processor energy consumption. We use updated energy parameters to 45 nm technology based on technology scaling rules. We also estimate processor power consumption

Table 5.4: Application characteristics.

Application		IPC	MPKI	LLC miss rate [%]	Power consumption [W]			
					Processor	LLC	DRAM	Total
SPEC CPU 2006	`bzip2`	1.9	1.1	25	15.4	1.3	8.3	25.0
	`hmmer`	2.0	0.9	14	14.5	1.3	7.9	23.9
	`mcf`	0.4	18.5	32	3.5	1.3	11.3	16.1
	`libquantum`	1.0	4.9	71	6.5	1.3	7.1	15.0
	`omnetpp`	1.0	3.5	18	7.4	1.3	8.9	17.6
	`milc`	1.0	3.2	68	8.6	1.3	8.3	18.1
	`lbm`	0.6	8.0	56	5.8	1.3	10.0	17.0
	`sphinx3`	0.7	4.4	47	7.4	1.3	8.2	16.9
PARSEC	`canneal`	0.4	13.2	72	3.8	1.3	11.1	16.3
	`dedup`	1.6	0.5	4	11.0	1.3	7.5	19.9
	`fluidanimate`	0.7	4.0	60	6.9	1.3	8.3	16.5
	`freqmine`	1.8	1.3	24	12.5	1.3	8.0	21.8
Micro-benchmarks	`STREAM`	0.3	35.0	99	2.8	1.3	10.1	14.2
	`GUPS`	0.2	92.8	52	1.4	1.3	19.5	22.2

using IPC-based linear scaling of the maximum power consumption of a 45nm Intel Xeon [68], as suggested in [13]. The estimates of the two power models closely matched one another. Finally, we estimate cache power consumption using CACTI 5 [119].

5.2.1 Workloads

We use a mix of several SPEC CPU 2006 [109] and PARSEC [21] applications, as well as the STREAM [82] and GUPS [42] micro-benchmarks. We select only applications that stress the memory system and that are potentially significantly impacted by V-ECC.

The `STREAM` micro-benchmark processes a 2M-element vector and performs copy, scale, add, and triad operations on each element in sequence. `STREAM` has very low temporal locality, but high spatial locality and is memory bound with little computation for every memory access. `GUPS` reads and

performs a single addition to update distinct pseudo-random locations in a 64MB array of 64-bit numbers. We use 5120k updates when performing cycle-based simulations and 16M updates when using PIN emulation. `GUPS` is even more memory intensive than `STREAM`, and, in addition, has essentially no spatial locality, intensely stressing the memory system.

For the SPEC CPU 2006 applications, we use the reference input dataset and run cycle accurate simulations for 200M representative instructions, as indicated by Simpoint [49], and run the applications to completion using PIN. We use the `simsmall` dataset for the cycle-based simulations of PARSEC and `simlarge` with PIN. We run all the PARSEC applications to completion using a single thread. Table 5.4 summarizes the IPC, cache miss ratio, and power consumption of the applications on the baseline system.

5.2.2 Results and Analysis

We present our analysis of V-ECC in four parts: the T2EC information storage and translation management (Section 5.2.2.1–5.2.2.2), the overall performance and energy impact for single chipkill ECC (Section 5.2.2.3); the flexible reliability schemes (Section 5.2.2.4); and implications on multicore processors that have a lower effective memory bandwidth for each core (Section 5.2.2.5).

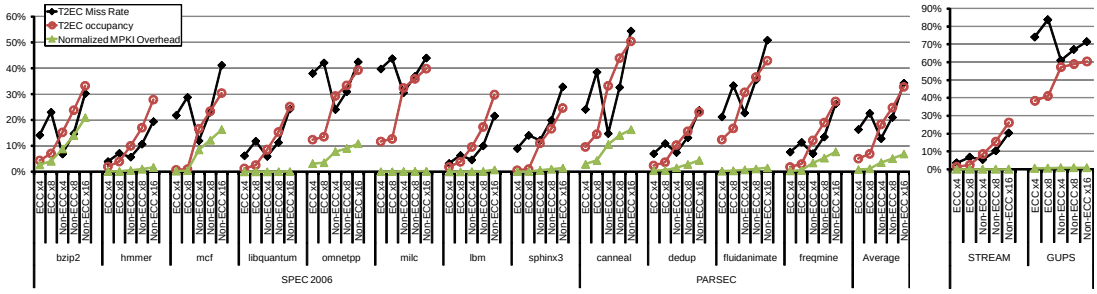


Figure 5.7: Impact on the LLC: T2EC miss rate and occupancy in the LLC as well as impact on application data miss rate.

5.2.2.1 V-ECC Storage Management

Maintaining T2EC information as data implies sharing storage resources between application data and its ECC redundancy, impacting data caching behavior. Figure 5.7 shows the impact on caching of T2EC data, including the average occupancy of T2EC in the LLC, the T2EC miss rate (T2EC accesses that required a DRAM access), and the impact on data MPKI (misses per thousand instructions). In general, T2EC miss rate and occupancy are directly proportional to the size of each T2EC (see Table 5.2). The greater the number of T2ECs in an LLC line, the lower the impact on data cache miss rate and memory traffic.

There is a significant difference in behavior between the configurations that use ECC DIMMs (ECC ×4 and ECC ×8) and those that do not (Non-ECC ×4, Non-ECC ×8, and Non-ECC ×16). With ECC DIMMs, T2EC is only accessed on write-backs to DRAM and the redundant information in the LLC is only maintained for dirty data. Without ECC DIMMs, T2EC

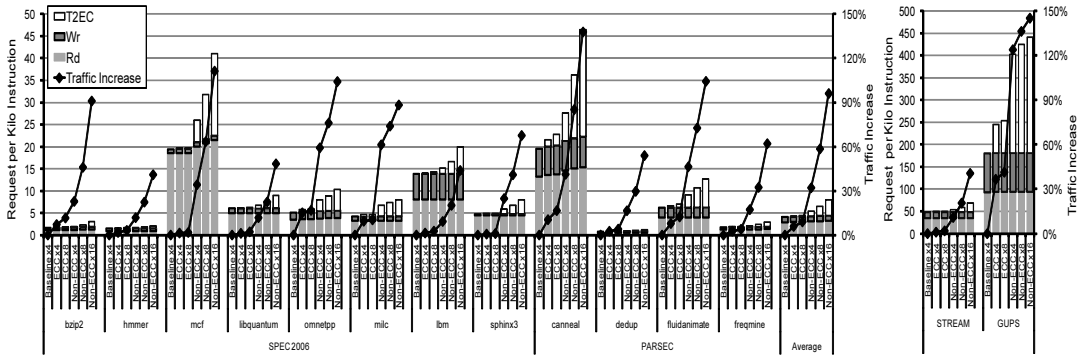


Figure 5.8: Impact of V-ECC on DRAM traffic.

information is accessed on any DRAM read or write and T2EC LLC lines are not partially valid. The impact can be seen in the higher LLC occupancy required for T2EC with Non-ECC DIMMs, as well as in in the lower miss rate. The miss rate is lower because more T2EC accesses are necessary to support detection.

While T2EC data can occupy upwards of 50% of the LLC, its impact on data caching behavior is not as severe. For most applications, MPKI is not significantly affected. A few applications (`bzip2`, `mcf`, `omnetpp`, and `canneal`), suffer an increase of up to 20% when ×16 DRAMs are used. The increase in MPKI results in increased DRAM traffic, which also grows due to T2EC accesses. The increase can be significant as shown in Figure 5.8, but as we discuss in Section 5.2.2.3 and Section 5.2.2.5, impact on performance is not severe and overall power consumption and energy-delay product are improved. In addition, T2EC occupancy and miss rate/traffic increases are modest in many configurations.

5.2.2.2 V-ECC Translation Management

In this subsection, we use PIN [77] to estimate the overheads of address translation and copy-on-write for T2EC physical memory allocation with large datasets. We model a two-level ECC address translation cache as in Figure 5.6 with a fully associative 16 entry L1 cache (same size as the UltraSPARC-III TLB [115]) and a 4K entry, 8-way set associative L2 translation cache. We measured the EA translation cache miss rate relative to the number of dirty LLC evictions (recall that a T2EC access on a memory read always hits in the EA cache because it is coupled to the data TLB).

As shown in Table 5.5, most applications we evaluated have EA cache miss rates that are lower than 1%. The exceptions to this low miss rate are `fluidanimate` (5%), `mcf` (7%), `omnetpp` (50%), `canneal` (52%), and `GUPS` (67%). More important than the actual miss rate is the frequency of misses relative to program execution, or the EA translation cache misses per thousand instructions (EA cache MPKI in Table 5.5), which are only significant in three applications: `omnetpp` (3.7), `canneal` (3.0), and `GUPS` (12.6).

With the exception of the `GUPS` micro-benchmark, we expect that a hardware page walk to fill translations will be able to support the rate of translations necessary, and writing back ECC information does not impact performance as long as write throughput is maintained; EA MSHRs in Figure 5.6 will allow non-blocking ECC address translation. For applications such as `GUPS`, we propose to use coarser-grained allocation of T2EC ranges in memory, similarly to super- or huge-pages used to improve TLB performance of

Table 5.5: EA translation cache behavior (KI: thousand instructions).

Application	EA cache miss rate [%]	Write-backs per KI	EA cache MPKI
bzip2	0.06	1.9	0.001
hmmer	0.005	2.2	0.0001
mcf	6.5	9.6	0.62
libquantum	0.05	9.9	0.005
omnetpp	50.0	7.4	3.7
milc	0.7	7.0	0.05
lbm	0.0003	21.4	0.0006
sphinx3	0.00007	1.2	0.00000008
canneal	52.0	5.9	3.0
dedup	0.9	0.4	0.004
fluidanimate	5.2	0.5	0.02
freqmine	0.04	0.1	0.000005
STREAM	0.0	20.6	0
GUPS	67	18.8	12.6

virtual memory [117]. We will evaluate this approach in future work, and for the performance analysis in the following subsections we roughly estimate for translation overhead by doubling the data TLB miss penalty.

We also measured the frequency of copy-on-write (COW) exceptions in our application benchmarks. On average, the COW rate is only 0.0001 for every 1000 instructions, and not more than 0.05 for every 1000 instructions. Because the frequency is so low, we did not perform a detailed performance evaluation of the additional overhead for allocating T2EC storage and maintaining the translation tables. Even an additional $10,000$ cycles for every COW event would not significantly impact application run time. Note that COW exceptions rarely access slow secondary storage, as most translation and allocation data structures are resident in DRAM.

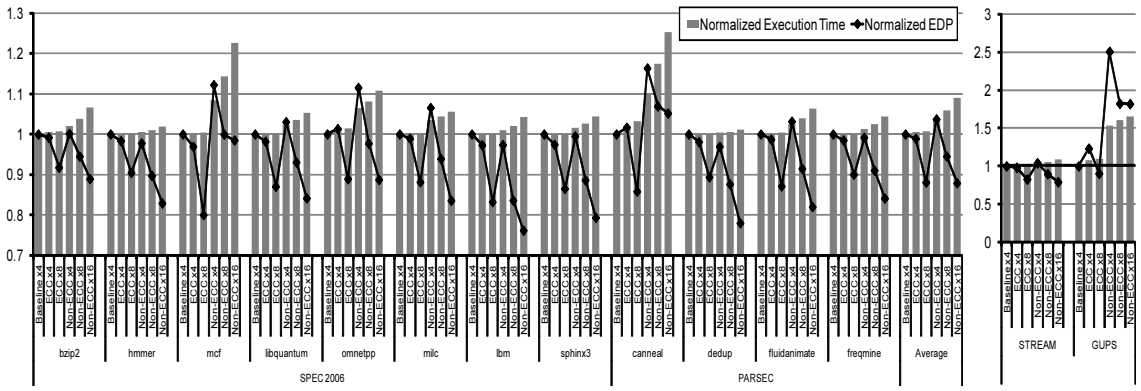


Figure 5.9: Performance and system EDP for baseline and V-ECC chipkill correct.

5.2.2.3 Chipkill Performance and Energy

Figure 5.9 presents the execution time and system energy-delay product (EDP) of the V-ECC configurations described in Table 5.1, normalized to those of the baseline ×4 chipkill-correct. For system EDP, we measured system power consumption as the sum of processor core power, LLC power, and DRAM power.

V-ECC with ECC DIMMs. V-ECC with ECC DIMMs has a very small impact on performance. With the exception of `canneal`, which has 2% and 3% lower performance with ECC ×4 and ECC ×8 respectively, all applications have a lower than 0.5% performance difference. This very low performance penalty is a result of the effective T2EC caching and the fact that the additional DRAM traffic is for writing out T2EC information and not on the computation critical path. Even the very write-intensive `GUPS` micro-benchmark that has no T2EC locality and very low arithmetic intensity only suffers a

$\sim$ 10% reduction in performance. ECC $\times$4, unfortunately, also has little impact on EDP and only improves it by $\sim$ 1%. ECC $\times$8, however, shows a very significant improvement in energy efficiency. DRAM power is reduced by an average of almost 30% and EDP is improved by an average of 12%. EDP improvement is very consistent using $\times$8 DRAMs, with only two outliers: `mcf` and `STREAM` have a 20% and 18% improvement respectively. Both benchmarks place significantly higher pressure on the memory system (see Table 5.4), and thus benefit more from increased memory power efficiency. `GUPS` demands even higher memory performance. While the EDP of `GUPS` is improved by 10% in ECC $\times$8 with more energy efficient $\times$8 DRAMs, it is degraded by 23% in ECC $\times$4, mainly due to the increase in DRAM power consumption (7%). Note that supporting chipkill with $\times$8 DRAMs in conventional systems is not possible unless custom-designed DIMMs with higher redundancy or increased access granularity are used.

V-ECC with Non-ECC DIMMs. V-ECC can also bring DRAM error tolerance to systems that use Non-ECC DIMMs. The extra DRAM accesses required for every read (and not just for writes) results in a larger impact on performance. Even with this extra traffic, however, application performance is degraded by 3%, 6%, and 9% using $\times$4, $\times$8, and $\times$16 chips, respectively, on average. The $\times$4 configuration does not reduce power consumption by much and the performance overhead leads to a 3% degradation in EDP. Wider DRAM configurations, however, improve EDP by 5% ($\times$8) and 12% ($\times$16) when compared to a standard chipkill configuration. In fact, only `canneal`

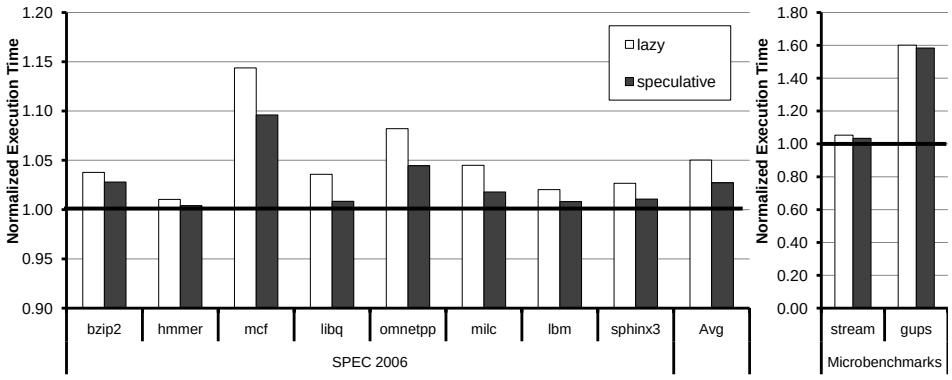


Figure 5.10: Speculative data forwarding.

and `GUPS` have a relatively worse EDP with Non-ECC ×8 and Non-ECC ×16. These two applications are memory bound and have random access patterns that do not make effective use of T2EC caching. The other two bandwidth-bound applications, `mcf` and `STREAM`, are not impacted as much. `STREAM`, especially, shows the effectiveness of caching T2EC data as it has very high spatial locality.

Speculative Data Forwarding. In V-ECC with Non-ECC DIMMs, we assumed that the memory controller returns data to the cache controller only when the virtualized T2EC is fetched and data is verified (lazy data forwarding). To mitigate the penalty with the virtualized T2EC, speculative data forwarding is possible. We compare the lazy scheme and speculative data forwarding in Figure 5.10. Speculative data forwarding provides better performance than lazy data forwarding (2.3% less performance overhead on average) This gain, however, comes at a cost of design complexity: speculative execu-

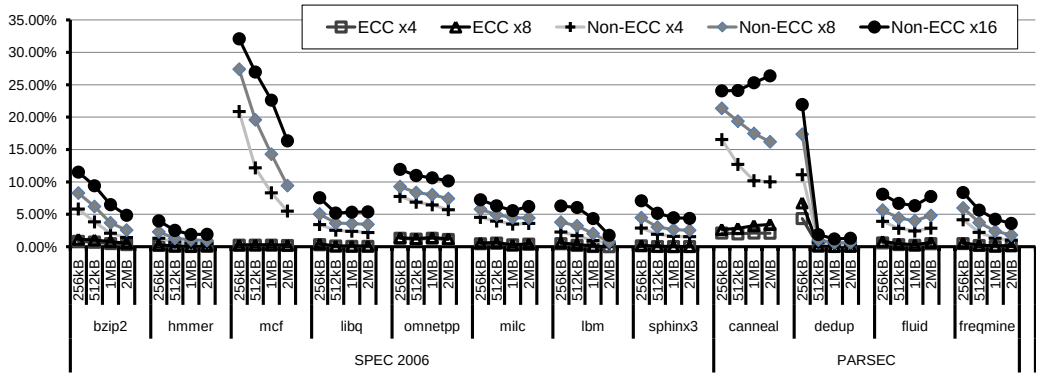


Figure 5.11: V-ECC performance with varying LLC size.

tion using unverified values and roll-back mechanisms to recover the system state when errors are detected.

Sensitivity to LLC size. Since V-ECC uses cache lines for storing the virtualized T2EC, V-ECC affects LLC caching behavior as MME described in Chapter 4. We evaluate V-ECC performance with varying LLC size from 256kB to 2MB and present how V-ECC is sensitive to LLC size in Figure 5.11. The performance impact in V-ECC with ECC DIMMs is stable across the simulated LLC sizes. In most applications (except `canneal` and `fluidanimate` in Non-ECC ×16), V-ECC with Non-ECC DIMMs is not sensitive to LLC size; performance penalty gradually decreases with larger LLC.

However, it is possible that V-ECC can drop performance significantly at a certain LLC size (as MME shown in Chapter 4). When an application has working set close to LLC size, the reduced effective LLC capacity due to caching of the virtualized T2EC in V-ECC can increase LLC miss rate significantly. An example is `OCEAN` (in SPLASH2 [129]) with a grid size of 258×258.

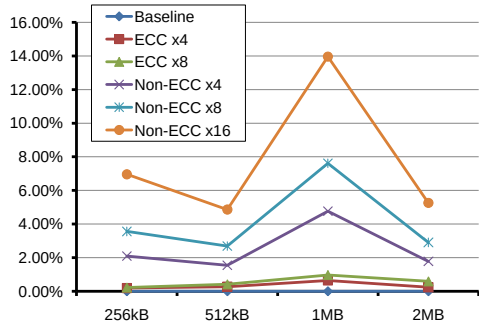


Figure 5.12: V-ECC performance impact on `OCEAN`.

Figure 5.12 presents performance penalty in `OCEAN` (258×258), while varying LLC size from 256kB to 2MB. Since `OCEAN`'s working set is close to 1MB, V-ECC degrades performance significantly only at 1MB LLC. The penalty, however, is very low in V-ECC with ECC DIMMs. Only V-ECC with Non-ECC DIMMs show more than 4% drop in performance.

Impact on Performance with Error Correction. So far, we evaluated V-ECC in error-free condition. In V-ECC with Non-ECC DIMMs, the memory controller always fetches data and its associated ECC; hence, error correction penalty is only additional cycles for invoking error correction steps, which is relatively small compared to the long DRAM latency.

In V-ECC with ECC DIMMs, however, the T2EC is not fetched for reads. Hence, error correction incurs fetching the virtualized T2EC from memory. From the memory error propensity we describe in Chapter 2, SER in DRAM is relatively low for processor execution so we can ignore the occasional T2EC read penalty for error correction.

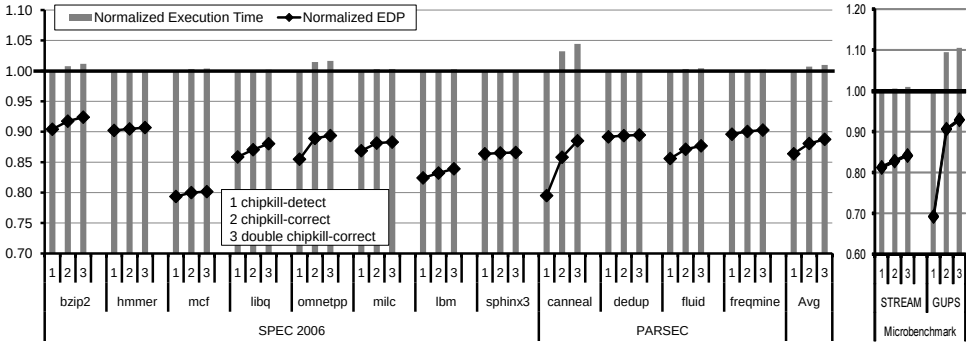
As discussed in Section 5.1.1, frequent errors indicate a chip failure, and we can use techniques such as dynamic data migration [107] for mitigating the penalty with hard failures. On the other hand, the memory controller can identify a faulty rank (after observing frequent errors) and fetch both data and its associated T2EC simultaneously when accessing the faulty rank. In this case, the performance of ECC $\times 4$ and ECC $\times 8$ will be similar to those of Non-ECC $\times 4$ and Non-ECC $\times 8$, respectively.

5.2.2.4 Flexible Protection

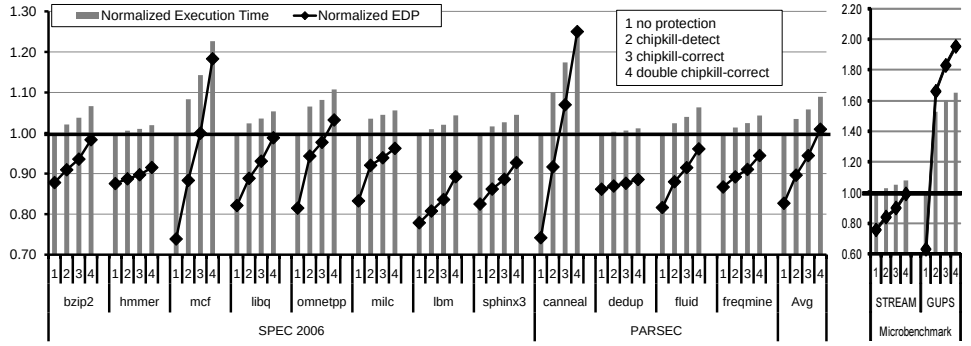
As described in Section 5.1.2.3 the V-ECC architecture enables flexibility in choosing the error protection level based on dynamic application, user, and system needs. To assess a new tradeoff that V-ECC enables, we evaluate the effect of different T2EC sizes, which are summarized in Table 5.2.

Figure 5.13 presents the execution time and system EDP, normalized to those of the baseline system, in ECC $\times 8$, Non-ECC $\times 8$, and Non-ECC $\times 16$ varying T2EC error protection. As expected, increasing the protection level increases EDP and execution time. The impact of adding the capability to tolerate a second dead chip, however, has a fairly small overhead overall when using ECC DIMMs (Figure 5.13(a)). Double chipkill-correct increases execution time by 0.3% at most over single chipkill correct, and system EDP is still $10 - 20\%$ better than with conventional $\times 4$ chipkill.

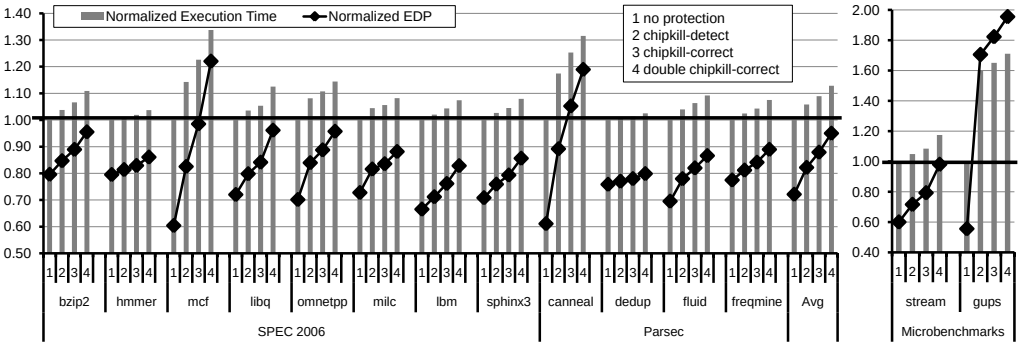
If ECC DIMMs are not available, the overhead of improved protection is more significant; Figure 5.13(b) and Figure 5.13(c) show the normalized



(a) ×8 ECC DIMM



(b) ×8 Non-ECC DIMM



(c) ×16 non-ECC DIMM

Figure 5.13: Performance and system EDP while varying error tolerance levels (execution time and system EDP are normalized to those of baseline ×4 chipkill).

Table 5.6: Summary of flexible error protection with V-ECC.

		Performance penalty	DRAM power reduction	System EDP gain
ECC ×8	chipkill-detect	0%	29%	14.6%
	chipkill-correct	0.7%	27.8%	12%
	double chipkill-correct	1%	27.2%	11.2%
Non-ECC ×8	no protection	0%	37.1%	17.3%
	chipkill-detect	3.4%	32.6%	10.4%
	chipkill-correct	5.8%	30.1%	5.6%
	double chipkill-correct	8.9%	26.7%	−0.9%
Non-ECC ×16	no protection	0%	59.5%	27.8%
	chipkill-detect	5.8%	53.4%	17.8%
	chipkill-correct	8.9%	50.1%	46.2%
	double chipkill-correct	12.8%	46.2%	4.9%

execution time and EDP of Non-ECC ×8 and Non-ECC ×16, respectively. The overall system EDP is better than the baseline in all configurations, with the exception of `mcf`, `omnetpp`, `canneal`, and `GUPS`. Note that the protection method can be varied at a granularity of a single page, which can make the options more attractive in an overall system design optimization process.

Table 5.6 summarizes the average performance penalty and DRAM power/system EDP gains of ECC ×8, Non-ECC ×8, and Non-ECC ×16; ×4 configurations are omitted since they do not show much gain.

5.2.2.5 Implications on Multicore Processors

So far our design and evaluation have focused on a single-core system. As described in Section 5.1.1, T2EC information is exclusively accessed by a DRAM controller and not shared among multiple processor cores. Therefore, it is easy to integrate V-ECC within a multiprocessor or multicore system. In multicore systems, however, the increased traffic due to T2EC accesses and

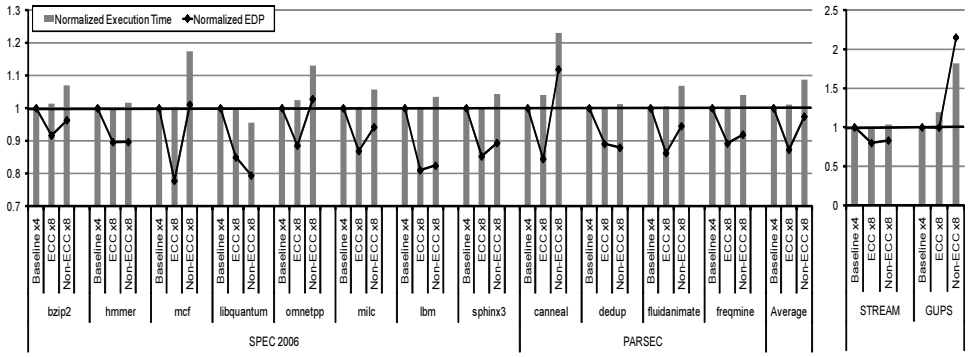


Figure 5.14: Performance and system EDP for ×8 V-ECC in a half-bandwidth (6.4GB/s) multicore-like scenario.

reduced effective LLC size may be more detrimental to performance than in a single core case because of the relatively lower per-core memory bandwidth. We gauge the potential impact of the ×8 configurations of V-ECC on a multicore by simulating a single-core system with half the memory bandwidth (6.4GB/s) and evaluate V-ECC in a 4-core CMP later.

The results of this reduced-bandwidth experiment are shown in Figure 5.14. The relative performance penalty of ECC ×8 and Non-ECC ×8 is only slightly worse than that of the full bandwidth system, and follows the same trends and insights described before. One anomalous result can be seen in `libquantum`, where Non-ECC ×8 slightly improves performance. This anomaly is a result of changes in the timing of DRAM accesses. The T2EC information in the LLC changes the eviction pattern that acts as an eager write-back mechanism, which has been shown to improve performance in some situations [69].

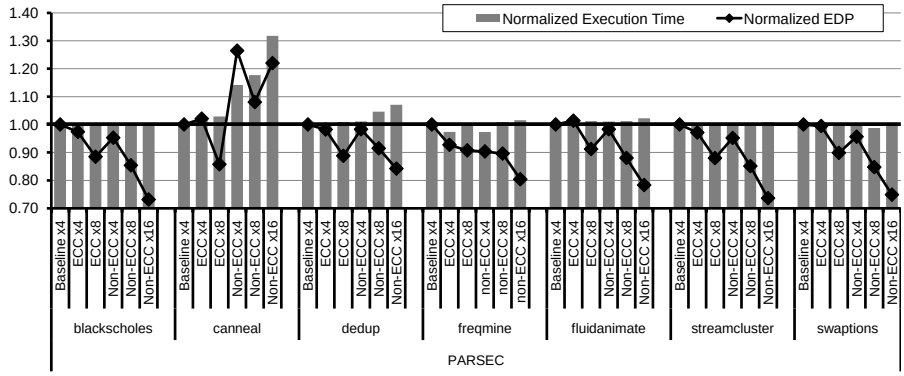


Figure 5.15: 4-core system.

We also evaluate V-ECC in multi-core environments in Figure 5.15. In this experiment, we use a 4-core CMP with a MESI directory protocol and two 128-bit wide memory channels (with baseline chipkill), implement V-ECC at LLC, and evaluate PARSEC applications with 4 threads. As shown in Figure 5.15, V-ECC with wider DRAM configurations ($\times 8$ and $\times 16$) shows improved system energy efficiency and low performance penalty. The only exception is `canneal`; as in single-core evaluation, V-ECC with Non-ECC DIMMs degrades system EDP in `canneal`. Slightly improved performance with V-ECC in some applications is due to the effect of eager write-back in V-ECC.

5.3 Related Work

The proposed memory protection mechanisms build on a large body of prior work, including general work on error coding [33, 34] and chipkill correct [15, 37, 116]. There has also been recent work on how to allocate redundant

information across chips in a DRAM module in ways that account for modern DRAM properties. One approach, like V-ECC, that uses the data chips to also store redundant information is mentioned in [140]. The focus of the paper was on reducing DRAM power consumption by modifying DIMM design and enabling mini-ranks that only access a subset of the chips on a DIMM at one time, and it included a suggestion for embedding redundancy uniformly within the data DRAM chips. There are patents [36, 48] that describe a similar concept of storing redundant information in data memory. This changes the overall data layout and access properties, and its full impact on power, performance, and strong chipkill protection has not been evaluated to the best of our knowledge.

Similar to [140], MC-DIMMs [14] have also been suggested to subset DIMM ranks and improve power consumption. As part of this work, techniques to implement chipkill-correct with MC-DIMMs were developed [13]. These techniques, like the ones in Section 5.1.2, use 3-check-symbol SSC-DSD chipkill-correct. The MC-DIMM approach uses uniform and fixed redundancy, however, and requires dedicating one DRAM chip to each of the three check symbols, which increases system cost. In our work, we focused on adding flexibility to the design and on using standard DIMMs to support strong memory error tolerance in efficient configurations.

Our research is also related to prior work that aims to reduce the overhead of reliable execution by using non-uniform protection. The Duplication Cache [12] can reduce the overhead of providing dual-modular redundancy for

shared-memory multiprocessor reliable systems. Instead of replicating all of memory, only dirty pages are replicated to increase effective capacity. The duplication cache is implemented on top of the virtual memory interface and copy-on-write exceptions to identify dirty pages. Our work is very different and is able to reduce the overhead of all aspects of memory error tolerance by extending ECC schemes to two tiers. Another non-uniform technique was proposed for computation rather than the memory system. Mixed-Mode Multicore (MMM) Reliability [126] proposes a mechanism that enables different reliability modes based on user demands; dual-modular redundant execution for reliable programs or single-mode execution for high performance. MMM emphasizes the need to provide flexible reliability modes in a system based on user and environmental needs. We fully support this approach and complement prior work by introducing flexible memory error-tolerance schemes.

5.4 Summary

In this chapter, we present general mechanisms to virtualize ECC-based memory error tolerance mechanisms. We show how V-ECC adds flexibility to what until now has strictly been a design-time decision. Decoupling the mapping of data from redundancy even allows systems with Non-ECC DIMMs to tolerate potentially significant memory errors. The focus of this chapter has been on developing the fundamental mechanisms and evaluating potential ECC schemes that take advantage of V-ECC. We show promising results on improving memory system efficiency with minimal impact to performance,

including examining near worst-case micro-benchmarks, which have very low arithmetic intensity and are very demanding on the memory system. We believe that the schemes we describe in this chapter are just the starting point for opportunities to take advantage of dynamically tunable memory protection.

Chapter 6

Adaptive Granularity Memory Systems

The continuing improvements in device density and in the potential performance of parallel processors place increased pressure on the throughput, power consumption, and reliability of memory systems. With the very high arithmetic throughput possible with modern processors, accessing off-chip memory is often the performance bottleneck. Moreover, supporting a large number of concurrent threads and applications requires higher memory capacity. The power consumed by high-capacity memory can be a large fraction of total system power [44]. Another issue with increased storage capacity is that the likelihood of memory errors grows, necessitating sophisticated, and often costly, error-tolerance mechanisms. The fundamental problem is that systems require reliable high-capacity memory with high throughput and low power all at the same time, while these parameters are often in conflict.

Memory systems often rely on spatial locality to reduce off-chip accesses and miss rate and to lower control and storage overheads. Usually, caches and DRAM systems use coarse-grained accesses of 64 bytes or larger. This conventional wisdom of amortizing overheads no longer leads to optimal system tradeoffs. When spatial locality is insufficient, two key resources

are squandered: Off-chip bandwidth is wasted, and unnecessary power is consumed on fetching data that will not be referenced. Fine-grained memory accesses used in some vector processors overcome these inefficiencies, but are also sub-optimal because spatial locality is quite common and should be exploited. This is especially true for error-tolerant systems because providing the same level of protection to fine-grained accesses requires more redundancy than it does for coarse-grained accesses.

In this chapter, We present *adaptive granularity* [135] to combine the best of fine-grained and coarse-grained memory accesses. We augment virtual memory to allow each page to specify its preferred granularity of access based on spatial locality and error-tolerance tradeoffs. We use sector caches and sub-ranked memory systems to implement adaptive granularity. We also show how to incorporate adaptive granularity into memory access scheduling. We evaluate our architecture with and without ECC using memory intensive benchmarks from the SPEC, Olden, PARSEC, SPLASH2, and HPCS benchmark suites and micro-benchmarks. The evaluation shows that performance is improved by 61% without ECC and 44% with ECC in memory-intensive applications, while the reduction in memory power consumption (29% without ECC and 14% with ECC) and traffic (78% without ECC and 68% with ECC) is significant.

The remainder of this chapter is organized as follows: Section 6.1 describes a tradeoff between redundancy overhead and access granularity; Section 6.2 presents the proposed adaptive granularity memory system; the eval-

uation methodology is described in Section 6.3; the results and discussion is shown in Section 6.4; Section 6.5 presents related work; then, Section 6.6 summarizes this chapter.

6.1 Error Protection and Access Granularity

In this section, we describe the tradeoff between redundancy overhead and access granularity. The high density of DRAM coupled with the large number of DRAM chips in many systems make memory one of the most susceptible components to errors and can reduce both the reliability and the availability of the system [101]. The most effective method to improve reliability is to tolerate errors using ECC codes [74, 100]. With ECC, every access to memory is accompanied by an ECC operation to ensure that the access is correct. One pertinent characteristic of commonly used ECC is that its overhead grows sub-linearly with the size of the data it protects ($O(\log_2 n)$, where n is the size of the data) as discussed in Section 2.3. Therefore, coarse-grained accesses can use more space-efficient codes that have lower redundancy, while tolerating the same number of errors; the finer-grained the access, the larger the overhead of ECC.

Moreover, since we cannot use an arbitrary width DRAM chip, the actual overhead in terms of chip count can be even larger. This ECC overhead manifests itself in reduced storage efficiency (more DRAM chips for the same data capacity) and lower effective pin bandwidth and power efficiency (more pins and watts for the same data bandwidth). For example, typical overhead

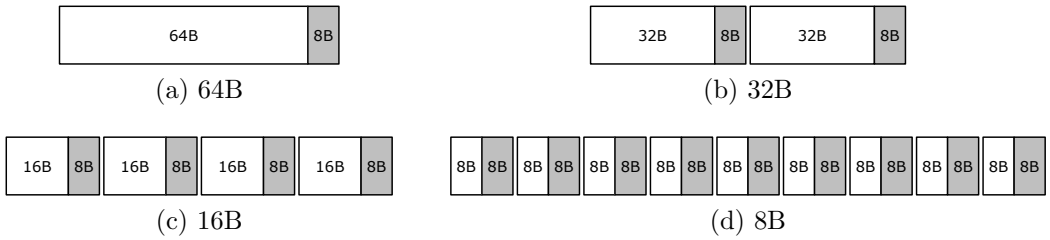


Figure 6.1: Tradeoff between access granularity and redundancy overheads. White and gray boxes represent data and ECC blocks respectively assuming a minimum access granularity of 8B (×8 DRAM chips and burst-8 accesses in DDR3). The finer the granularity, the higher the overhead of redundancy.

with ECC DIMMs is 12.5% with 8 bits of ECC information for every 64 bits of data (or a 16 bits of ECC for 128 bits of data). In Cray's Black Widow [11], many-narrow-channels memory subsystem (illustrated in Section 3.3.1) has 25% overhead: 7-bit SEC-DED for each 32 bits of data, but the actual overhead is 8 bits when using ×8 DRAM chips. Furthermore, providing 8B access granularity with DDR3 using the many-narrow-channels approach requires 100% ECC overhead (a 5-bit ECC provides SEC-DED protection for 8-bit data, but we still need at least one ×8 DRAM chip to store ECC information, resulting in 8 overhead bits per 8-bit of data) as well as additional control overhead. Supporting ECC in a sub-ranked memory system is also very expensive: MC-DIMM requires a 37.5% or higher ECC overhead for chipkill-correct [13], for example. Figure 6.1 summarizes the tradeoff between access granularity and storage efficiency.

In summary, neither a conventional coarse-grained memory system nor a fine-grained memory system, including many-narrow-channels and sub-ranking

schemes, can provide optimal throughput and efficiency. Optimizing the system for coarse-grained accesses sacrifices throughput when spatial locality is low, while tuning for fine-grained accesses makes the overheads of control and/or ECC significant. Modern computing systems, however, require all of these merits: high throughput both for coarse-grained and fine-grained accesses and high reliability and availability levels, and all of these at low power and storage overheads.

6.2 Adaptive Granularity Memory System

We propose the adaptive granularity memory system (AGMS) that combines the best of fine-grained and coarse-grained accesses. The main idea is to use a coarse-grained configuration for memory regions with high spatial locality and a fine-grained configuration for memory regions with low spatial locality. The proposed mechanism is a vertical solution that requires collaboration between several system levels: The application provides preferred granularity information (Section 6.2.1); the OS manages per-page access granularity by augmenting the virtual memory (VM) interface (Section 6.2.2); a sector cache manages fine-grained data in the cache hierarchy (Section 6.2.3); and a sub-ranked memory system and mixed-granularity memory scheduling provide efficient handling of multiple access granularities within off-chip memory system (Section 6.2.4 and Section 6.2.5). We also discuss the tradeoffs in making granularity decisions (Section 6.2.6).

6.2.1 Application Level Interface

As explained in Section 6.1, AGMS requires different memory protection schemes for different access granularities. The degree of redundancy, and thus the memory layout, has to change (see Section 6.2.4 for details). Consequently, the processor cannot adapt the granularity independent of the software. The tuning/adaptation can be done either statically at compile time or dynamically by the OS (we discuss the dynamic case later). In the static approach, the programmer or an auto-tuner provides granularity information through a set of annotations, hints, compiler options, and defaults that associate a specific tolerance mechanism with every memory location.[1] We envision that the programmer will declare a preferred access granularity when memory is allocated. More accurately, we allow the programmer to override the default access granularity using annotations and compiler hints. In Fortran, programmer annotations can take the form of another array attribute; in C, we can add a parameter to `malloc`; in C++, we can overload the `new` operator. The compiler may also select granularity with appropriate analysis.

6.2.2 OS Support

Granularity and protection schemes are applied when physical memory is allocated and are thus closely related to the virtual memory manager. We augment the virtual memory interface to allow software to specify the preferred

[1]Note that a physical memory location can only be protected using a single mechanism at any given time because the protection scheme and redundant information need to be checked and updated consistently.

access granularity for each page. The per-page access granularity is stored in a page table entry (PTE) when a page is allocated. This information is propagated through the memory hierarchy along with requests, miss status handling registers, and cache lines so that the memory controller can use the information to control DRAM channels. Since most architectures have *reserved* bits in a PTE, we can accommodate the preferred access granularity similar to per-page cache attributes in the ×86 ISA [57].

Because the OS manages both access granularity and virtual memory, it is possible to dynamically adapt the granularity without application knowledge. This would require hardware support for determining access granularity, such as the mechanisms proposed for fine-grained cache management [67] as well as the OS to copy (migrate) pages or change granularity when paging. We leave further exploration of this idea to future work and discuss it further in Section 6.2.6.

6.2.3 Cache Hierarchy

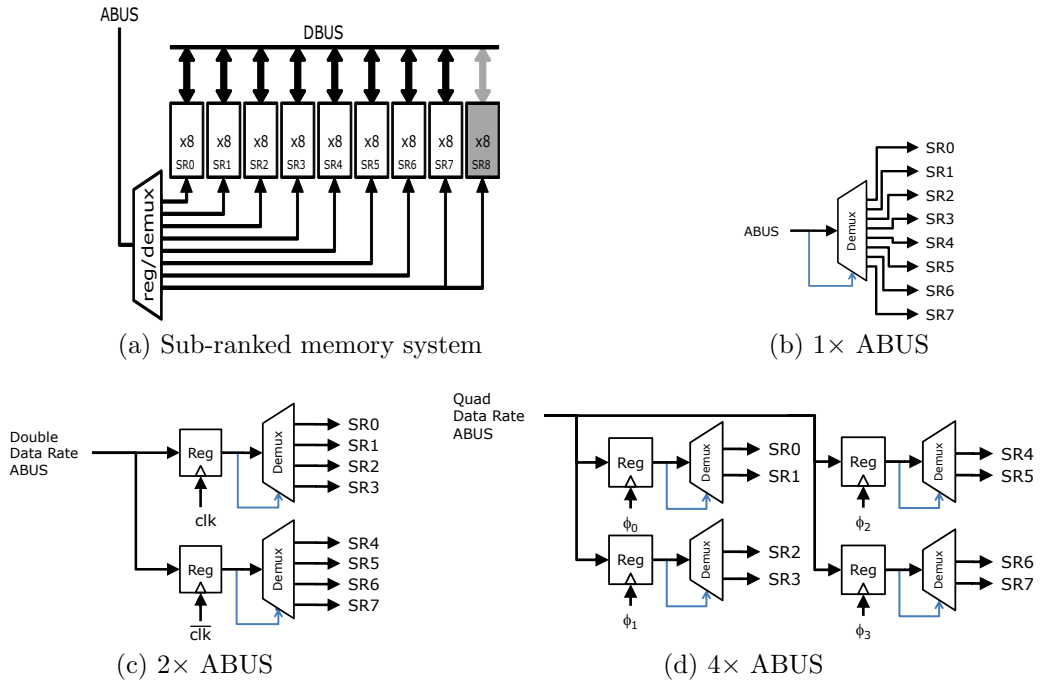
AGMS issues both coarse- and fine-grained requests to main memory and thus needs to manage both granularities of data within the cache hierarchy. The simplest way is to use a cache with a line size equal to the smallest access granularity, e.g. 8B. A better design choice is a sector cache [75, 98] (illustrated in Section 3.3.2). A cache line in a sector cache is divided into multiple sectors, and each sector maintains its own valid and dirty bits, but there is only one tag for each multi-sectored cache line. Since sector caches do not increase address

tag overhead, the additional cost is only the storage for valid and dirty bits: 14 bits per cache line when a 64B cache line is divided into eight 8B sectors.

While the more advanced cache architectures described in Section 6.5 can provide better management of fine-grained data, we choose a simple sector cache in this dissertation to better isolate performance gains from adaptive granularity memory access; the simple sector cache allows a fair comparison between the adaptive granularity memory system and a conventional coarse-grain-only architecture.

6.2.4 Main Memory

Sub-Ranked Memory Systems. We leverage the sub-ranked memory system approach (described in Section 3.3.1) because it enables fine-grained accesses with minimal control overhead. We use a 64-bit wide memory channel with DDR3 DRAMs: eight ×8 DRAMs per rank. Figure 6.2(a) shows the main memory architecture for this study. The minimum access granularity in our system is 8B since we can control each memory device independently. Though sub-ranked memory systems can provide multiple granularities (8B, 16B, 24B, 32B, and 64B), we restrict access granularity to 64B (coarse-grained) and 8B (fine-grained) in this dissertation. A fine-grained 8B request is serviced by a burst of 8 transfers from a single ×8 DRAM chip, and a coarse-grained 64B request is serviced by a burst of 8 transfers from all 8 ×8 DRAM chips in a rank.



(a) Sub-ranked memory system

(b) 1× ABUS

(c) 2× ABUS

(d) 4× ABUS

Figure 6.2: Sub-ranked memory with a register/demux architecture of 1×, 2×, and 4× ABUS bandwidth. We use a register/demux similar to the one suggested in MC-DIMM [14]. Note that we do not use a register with the 1× ABUS configuration.

Address Bus Bandwidth. Because each fine-grained access reads or writes only 8B of data, it requires a factor of 8 more memory requests to achieve the equivalent throughput of coarse-grained accesses. In other words, fine-grained accesses will saturate an address/command bus (ABUS) that is designed for coarse-grained accesses, and greater command signaling bandwidth is required for fine-grained accesses to fully utilize the DRAM bandwidth. Many commercial high-end systems including FB-DIMM [83], Power 7 [59], Cray Black Widow [11] as well as the custom address/command bus in the Convey S/G

DIMM [25] and decoupled DIMM [141] already use or suggest 3 to 4 times, or even more, faster signaling for the ABUS. Increasing ABUS bandwidth is a matter of overall system optimization in terms of cost, power, and performance. We explore the design space of ABUS bandwidth later in this section.

Figure 6.2(b)-(d) show the architectures of the register/demux used in the sub-ranked memory system, and Figure 6.2(c) and Figure 6.2(d) show how 2× and 4× ABUS bandwidths are provided. While the 2× ABUS scheme can be achieved by using double data rate signaling as with the data bus (DBUS) with a relatively simple register/demux, the 4× ABUS scheme, which uses quad data rate signaling, may increase design complexity due to signal integrity issues.

Memory controllers. An interesting challenge in implementing adaptive granularity, compared to fixed granularity memory systems, is the effective co-scheduling of memory requests with different granularities; this includes buffering, scheduling, and providing quality of service. The example in Figure 6.3(a) illustrates how a coarse-grained request is unfairly deferred when there are many fine-grained requests, assuming the commonly used FR-FCFS [95] scheduling policy. The coarse-grained request at *time 2* cannot be immediately issued due to pending fine-grained requests F0 and F3, which were queued at *time 0* and *time 1* respectively. While the coarse-grained request waits in the queue, other fine-grained requests (F1, F2, and F0) arrive and are scheduled quickly since a fine-grained request can be serviced once its single associated sub-rank becomes ready. Coarse-grained requests, on the other

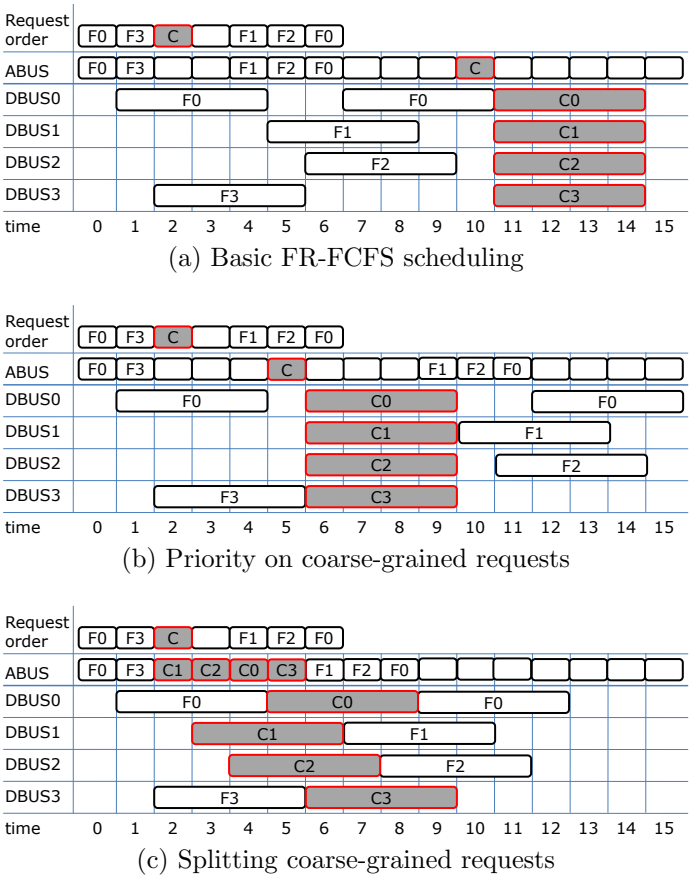


Figure 6.3: AGMS scheduling examples. Fx represents a fine-grained request to sub-rank x, and C represents a coarse-grained request. There are four sub-ranks, and a memory device returns a 4-cycle burst after a request is made. FR-FCFS scheduling is used, and data bus contention is the only constraint in scheduling in this simple example.

hand, can only be scheduled when all sub-ranks are available at the same time. As a result, the coarse-grained request is serviced only after all fine-grained requests are serviced, potentially degrading performance significantly.

We propose two solutions to address this problem: (i) to give higher priority to coarse-grained requests; and (ii) to split a coarse-grained request into multiple fine-grained requests. The first solution prioritizes a coarse-grained request when its service is unfairly deferred due to fine-grained requests. Supporting different request priorities is a common feature of modern out-of-order memory schedulers. As shown in Figure 6.3(b) when the scheduler detects that a coarse-grained request is deferred due to fine-grained requests (at *time 2*), it raises the priority of the coarse-grained request. This prevents fine-grained requests with normal priority from being scheduled. As a result, the coarse-grained request finishes at *time 9*, after which fine-grained requests are serviced.

Our second solution splits a coarse-grained request into many fine-grained requests so that each fine-grained request (belonging to a single coarse-grained request) can be opportunistically scheduled. The original coarse-grained request finishes when all its fine-grained requests are serviced. Figure 6.3(c) shows such an example. At *time 2*, the coarse-grained request is split into four fine-grained requests (C0, C1, C2, and C3), and the coarse-grained request finishes when all of them are serviced (at *time 9*). A caveat to this solution is that it potentially increases ABUS bandwidth requirement. We compare the two solutions in more detail later in this subsection.

(a) Coarse-grained (b) Fine-grained

Figure 6.4: Coarse-grained and fine-grained accesses with ECC

6.2.4.1 Data Layout.

When ECC is not used, the data layout in physical memory is the same for both fine-grained and coarse-grained pages. When ECC is enabled, on the other hand, we need to use a different data layout for fine-grained pages to account for the higher required redundancy. Figure 6.4 compares a coarse-grained and fine-grained accesses with ECC in a DDR3-based system. Note that single hardware with AGMS can provide both coarse-grained and fine-grained configurations unlike conventional memory systems that are either coarse-grain only or fine-grain only. Fine-grained data can provide higher throughput with low spatial locality, but it increases the ECC overhead since every data block needs its own ECC code. The 8B minimum access granularity dictates at least 8B for ECC.

In this dissertation, we use a simple scheme in which a fine-grained physical page is twice as large as a coarse-grained nominal page, e.g. 8kB of storage for a 4kB data page. Each 8B of data is associated with 8B of ECC. Hence, a fine-grained request is serviced by accessing 16B in total. Memory controllers must interpret this change in addressing when making fine-grained data accesses with ECC, and the OS should manage physical memory pages

accordingly. As a result, fine-grained pages have low storage efficiency, but can still provide better throughput than always using coarse-grained accesses. We store data and its associated ECC in different memory devices, providing better reliability than other embedded-ECC designs [36, 48, 140], which store ECC in the same DRAM row as the data it protects. A more generalized and flexible scheme such as Virtualized ECC [133] (described in Chapter 5) can manage ECC without changing physical data layout and can even handle granularities other than 64B and 8B easily, but we leave exploring such configurations to future work. We assume that ECC DIMMs are used, and ECC information for coarse-grained accesses is stored in the dedicated ECC DRAM chips.

6.2.5 AGMS Design Space

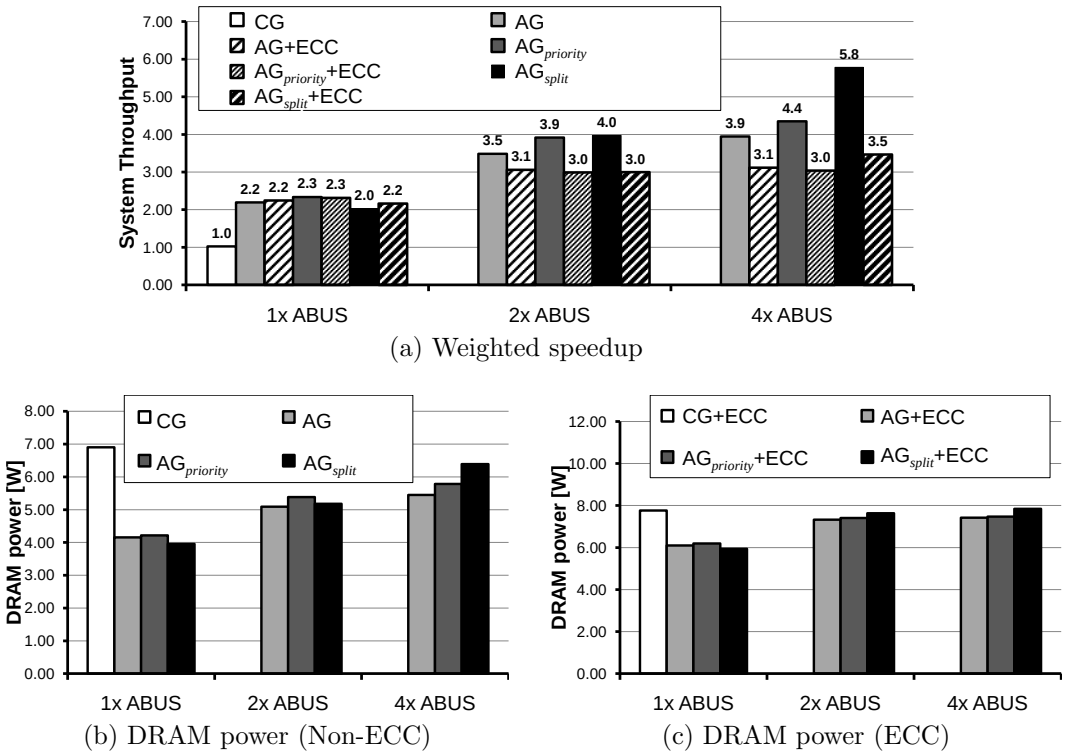
We now explore the design space of AGMS. We describe the details of the simulation settings and system parameters in Section 6.3. We use `GUPS` for exploring the design space because it is very memory-intensive and has many fine-grained requests. Furthermore, `GUPS` is often used as the gold standard for evaluating memory systems and network designs in large-scale systems and is considered to be a challenging benchmark [2]. We later show that AGMS provides significant benefits to real applications in addition to this important micro-benchmark.

`GUPS` performs a collection of independent read-modify-write operations to random locations (8B elements in our experiments). `GUPS` has two buffers:

an index array and a data array. The index array is accessed sequentially. The values stored in the index array are random numbers that are used for addressing the data array to be updated. For adaptive granularity, we define the index array as a coarse-grained region and the data array as a fine-grained region. We simulate a 4-core system with an instance of `GUPS` per core and a single 1067MHz DDR3 channel. We choose this relatively low-bandwidth configuration because it is representative of future systems that are expected to have larger compute to memory bandwidth ratios than current systems. Limited experiments with higher bandwidth yielded very similar results in the case of `GUPS`.

Figure 6.5(a) shows the throughput of various system configurations using weighted speedup [43]. In Figure 6.5, *CG* represents a conventional fixed-granularity coarse-grained system, *AG* is the baseline AGMS with standard FR-FCFS scheduling, $AG_{priority}$ and AG_{split} are AGMS, where the former uses higher priority for deferred coarse-grained requests and the latter splits coarse-grained requests into multiple fine-grained requests. The suffix *+ECC* represents ECC support in each configuration. Note that CG+ECC has identical performance to CG so the throughput of CG+ECC is not shown here.

Compared to CG (coarse-grain-only), the AG schemes improve system throughput significantly: 100−130% with nominal 1× ABUS bandwidth, 150−200% with 2× ABUS, and up to 480% with 4× ABUS. Note that with AG, the index array uses coarse-grained accesses and the data array is accessed with fine granularity. The performance of the AG schemes with 1× and 2× ABUS is



(a) Weighted speedup

(b) DRAM power (Non-ECC)

(c) DRAM power (ECC)

Figure 6.5: `GUPS` 4-core simulation for design space exploration.

limited by ABUS bandwidth: They have almost 100% ABUS utilization, while data bus (DBUS) utilization is only 25% (1× ABUS) and 50% (2× ABUS). With 4× ABUS, DBUS utilization reaches more than 70%. Note that even with 1× ABUS, the effective throughput is still twice as high as that of a coarse-grain-only system.

The throughput of the baseline AG system is limited by coarse-grained requests to the index array that are unfairly deferred due to many fine-grained accesses. This effect is more pronounced with 2× or 4× ABUS. $\text{AG}_{priority}$ and AG_{split} overcome this inefficiency by allowing coarse-grained requests to

complete in a timely manner. $AG_{priority}$ performs slightly better when ABUS bandwidth limits overall throughput (1× ABUS), but AG_{split} is best when ABUS bandwidth is higher (2× and 4× ABUS).

When ECC is enabled, the AG schemes improve performance by $120-130\%$ with 1× ABUS, $200-210\%$ with 2× ABUS, and up to 250% with 4× ABUS. One anomalous case is AG_{split} with 1× ABUS, where the throughput of AG_{split}+ECC is better than that of AG_{split}. The reason is that AG_{split}+ECC makes coarser requests because of the ECC data and thus requires less ABUS bandwidth. The coarse-grained requests are split into only four finer-grained requests (as opposed to eight requests with AG_{split} without ECC). Note that 4× ABUS does not provide any further improvements because the data bus is already heavily utilized (more than 70%) with 2× ABUS due to the extra bandwidth consumed by the redundant ECC information.

Figure 6.5(b) and Figure 6.5(c) compare the DRAM power consumption of the evaluated configurations. In general, the AG schemes consume much less power than CG because they avoid accesses and transfers of unnecessary data and mitigate DRAM "overfetch" [13]. Having higher ABUS bandwidth increases DRAM system utilization and DRAM power consumption. The significant improvements to system throughput, however, compensate for this increased power, leading to superior power efficiency.

Based on the evaluation results shown in Figure 6.5, in the rest of the chapter, we use AG_{split} with 2× ABUS both for non-ECC and ECC configurations. Although 4× ABUS improves `GUPS` significantly, `GUPS`'s access

pattern of almost exclusively fine-grained requests is not common in real applications. Therefore, we choose a 2× ABUS configuration over a 4× ABUS system given its design complexity and power overhead of a 4× ABUS system. Both $AG_{priority}$ and AG_{split} perform equally well with 2× ABUS bandwidth in general, but AG_{split} can utilize bandwidth more efficiently in a few cases.

6.2.6 Access Granularity Tradeoffs

While fine-grained accesses can utilize off-chip bandwidth more efficiently than coarse-grained ones, we should also consider that fine-grained access may degrade performance because of potentially higher average memory access latency. Ideally, bottom-line performance can be used to determine granularity, but the high cost of copying (migrating) a page to change its data layout (when ECC is used) for a different access granularity dictates that such changes be infrequent. As a result, common approaches to "experiment" dynamically and monitor performance are ill-suited for granularity selection. Instead, we propose a heuristic that incorporates metrics of spatial locality and DRAM characteristics.

The first component of our heuristic is cache spatial locality. Fine-grained accesses avoid fetching unnecessary data and utilize scarce off-chip bandwidth resources more effectively. Minimizing traffic generally reduces memory power consumption and can maximize system throughput when memory throughput is the bottleneck. We estimate the spatial locality of a page by averaging the number of words used in each fetched cache line for that page.

Thus, we can estimate the average traffic for fetching a line from a page as follows:

- The cost for a coarse-grained access (64B) is 73B: 64B data, 8B ECC, and 1B for control.
- The cost for a fine-grained access (8B) depends on the number of words referenced. For each referenced word, we need 17B: 8B data, 8B ECC, and 1B for control.

Consequently, fine-grained accesses minimize the amount of traffic if an average cache line in a page has fewer than 4 referenced words (for the configuration parameters used in this study).

Minimizing traffic, however, does not always guarantee higher overall performance or efficiency. The reason is that with fine-grained requests, multiple accesses to the same line will all be treated as cache misses (they hit in the tag array, but miss on the actual data word because of the sector cache design). These misses could have been avoided with coarse-grained accesses. Thus, if memory bandwidth is not the bottleneck, fine-grained accesses can degrade performance. We account for this effect by considering DRAM access characteristics that make fetching additional data relatively cheap.

The second component of the heuristic is DRAM page hit rate. A high DRAM page hit rate reduces the potential gains of fine-grained data access since the relative overhead of fetching an entire cache line is smaller when page

hit rate is high. When the page hit rate is low, fine-grained accesses allow more rapid transitions between pages and also increase the level of parallelism in the memory system (sub-rank level parallelism in addition to rank and bank level parallelism). Thus, if the page hit rate is high, pre-fetching extra data with coarse-grained accesses does not significantly impact effective memory throughput and can improve cache hit rate and average load latency.

We combine cache-line spatial locality and page hit rate by adding a penalty (α) to fine-grained accesses when page hit rate is high: We opt for fine granularity if $\alpha\times$ fine-grained-cost $\leq$ coarse-grained-cost. α is a parameter that penalizes fine-grained accesses and is 1.0 when only one word is referenced for more than 70% of cache lines in a page. Otherwise, α is determined by DRAM page hit-rate: 1.0 if page hit-rate is lower than 60%; 1.3 if lower than 70%; 1.8 if lower than 80%; and 3.0 otherwise. These weights were chosen arbitrarily, and we determined that performance is insensitive to this choice to a large extent.

The two components of the granularity-decision heuristic can be calculated statically, using programmer hints or compiler analyses, or dynamically, by allowing the OS to track access characteristics. In this research, we use an intermediate approach to identify fine-grained data regions using off-line profiling. Instead of modifying the application code and OS, we use the profiler to guide our decisions. We discuss the potential merits of this approach compared to dynamic prediction in the evaluation section, but overall, we determine that once an application starts running, its granularity characteristics are stable.

Note that the OS can always dynamically override granularity decisions and set all pages to coarse-grained if it determines that to be the most beneficial approach. For example, if memory bandwidth utilization is expected to be low leading to fine-grained access increasing memory footprint with no likely benefit to performance.

6.3 Evaluation Methodology

We evaluate AGMS using a combination of PIN-based emulation [77] to collect granularity statistics for full application runs and detailed cycle-based simulations to determine impact on performance and power. For the cycle-based simulations, we use the Zesto simulator [76] integrated with a detailed in-house DRAM simulator that supports sub-ranked memory systems as well as variable ABUS bandwidth as described in Section 6.2.4. The DRAM simulator models memory controllers and DRAM modules faithfully, simulating buffering of requests, FR-FCFS [95] scheduling of DRAM commands, contention on shared resources such as ABUS and DBUS, and all latency and timing constraints of DDR3 DRAM. Banks and sub-ranks are XOR interleaved to minimize conflicts [139].

Workloads. We use a mix of several applications from SPEC CPU 2006 [109], PARSEC [21], Olden [30], SPLASH2 [129], and the HPCS [3] benchmark suites as well as micro-benchmarks such as `GUPS` [42], `STREAM` [82], and `Linked List` [4] (Table 6.1). We choose mostly memory-intensive applications without spatial locality, but also include applications that are not memory intensive

Table 6.1: Benchmark statistics: LLC MPKI (misses per thousand instructions), DRAM page hit rates and IPC are estimated from the baseline cycle-accurate simulation with a single core; average number of used words per cache line is gathered using PIN.

Benchmark	Input size	IPC	LLC MPKI	DRAM page hit rate	Avg. referenced words per cache line	Description
SPEC						
`mcf`	ref	0.24	31.3	19.1%	3.59	Vehicle scheduling using a network simplex algorithm
`omnetpp`	ref	0.49	11.6	47.8%	3.22	Discrete event simulations
`libquantum`	ref	0.45	15.6	98.9%	4.09	Simulates a quantum computer
`bzip2`	ref	0.80	3.2	57.1%	3.63	`bzip2` compression
`hmmer`	ref	0.98	0.87	91.3%	7.93	Protein sequence analysis using hidden Markov models
`astar`	ref	0.87	0.59	44.0%	2.86	A* path finding algorithm
`lbm`	ref	0.56	22.9	82.6%	3.92	Fluid dynamics using Lattice-Boltzmann method
PARSEC						
`canneal`	400k elements	0.32	17.2	14.1%	1.87	Simulated annealing to minimize the routing cost
`streamcluster`	16k points	0.49	14.5	86.8%	7.24	On-line clustering of an input stream
SPLASH2						
`OCEAN`	1026^2 grid	0.54	18.6	92.6%	6.68	Simulates large-scale ocean movements
Olden						
`mst`	2k nodes	0.12	41.6	40.5%	2.30	Minimum spanning tree
`em3d`	200k nodes	0.19	39.4	27.4%	2.62	Simulates electromagnetic wave
HPCS						
`SSCA2`	64k nodes	0.27	25.4	25.5%	2.63	Computes betweenness centrality in a large graph
Micro-benchmarks						
`Linked List`	342 linked lists	0.04	111.9	34.2%	1.99	Linked list traversal
`GUPS`	8M elements	0.08	174.9	10.9%	1.84	Updates random memory locations
`STREAM`	2M elements	0.4	51.9	96.5%	7.99	streaming copy, scale, add, and triad

and/or have high spatial locality. Note that with adaptive granularity, all applications can perform at least as well as on the baseline system because

coarse-grained accesses are used by default, but we report numbers based on the heuristic described in Section 6.2.6.

`mcf` and `omnetpp` are memory intensive and have low spatial locality with less than half a cache line referenced by a processor core on average and low DRAM page hit rate. `libquantum` and `lbm` are memory-intensive and fewer than half the words in a cache line are accessed on average, but have high spatial locality for DRAM (high DRAM page hit rate). `bzip2`, `astar`, and `hmmer` are not memory-intensive. Applications from PARSEC (except `streamcluster`), Olden, HPCS `SSCA2`, and the micro-benchmarks (except `STREAM`) all stress main memory and have very low spatial locality: DRAM page hit rate is low, and only one or two words are referenced in each cache line on average. `STREAM` (micro-benchmark), `OCEAN` (SPLASH2), and `streamcluster` (PARSEC) are memory intensive, but present high DRAM page hit rate and access most words in a cache line. Note that we do not include applications that do not stress memory as the results will be the same for the baseline system and AGMS.

For the cycle-based simulations, we ran a representative region from each application. We use Simpoint [49] to determine the regions for SPEC applications and manually skipped the initialization of the simpler and more regularly-behaved Oldeni, PARSEC, and SPLASH2 benchmark suites, HPCS `SSCA2`, and the micro-benchmarks. The size of each representative region is 200M instructions for the 4-core simulations and 100M instructions for the

Table 6.2: Simulated base system parameters.

Processor core	4GHz x86 out-of-order core (4 or 8 cores)
L1 I-caches	32kB private, 2-cycle latency, 64B cache line
L1 D-caches	32kB private, 2-cycle latency, 64B cache line
L2 caches	256kB private for instruction and data, 7-cycle latency 64B cache line
Last-Level caches (L3)	shared cache, 64B cache line 4MB 13-cycle latency for 4-core systems 8MB 17-cycles latency for 8-core systems 64B cache line
On-chip memory controller	FR-FCFS scheduler [95], 64-entry read queue, 64-entry write queue XOR-based bank, sub-rank mapping [139]
Main memory	1 72-bit wide DDR3-1066 channel (64-bit data and 8-bit ECC) ×8 DRAMs, 8 banks per rank, 4 ranks per channel parameters from Micron 1Gb DRAM [84]

8-core simulations. For the PIN-based experiments, we ran all applications to completion.

Data page profiling. To collect profiling information, we use PIN [77] and emulate a 1-level 1MB 8-way set-associative cache with 64B lines. During program execution, we profile which 8B words are referenced within each cache line. We then aggregate the per-cache line cost across each 4kB page and determine the granularity recommendation that can be overridden dynamically by the OS (see Section 6.2.6), although we do not override the decision in this study. We use this static profiling method to report the ECC storage overheads of fine-grained pages in Section 6.4.1 and to identify fine-grained data pages for the cycle-based simulations (Section 6.4.2–6.4.3).

System Configurations. Table 6.2 gives the parameters of the baseline coarse-grain-only system used in our cycle-based simulations. The cache hierarchy of the base system has an instruction pointer prefetcher for the instruction caches and a stream prefetcher for the data caches.

We use the following configurations for evaluating the potential of the AGMS approach.

- CG+ECC: coarse-grain-only system as described in Table 6.2. ECC is stored in dedicated DRAMs.
- $CG_{\text{sub-ranked}}$+ECC: coarse-grain-only system, but it uses a sub-ranked memory system, similar to MC-DIMM [13]. We use the best configuration from [13], 4 sub-ranks per rank (see Figure 3.6(c)). For ECC support, we assume that each sub-rank has a dedicated ECC DRAM so that a 64B request is served by a burst of 32 transfers outt of three $\times 8$ DRAMs, of which two is for data and one is for ECC.
- FG+ECC: fine-grain-only system. The memory controller accesses only requested words (8B), but every fine-grained access is accompanied by 8B ECC; an 8B request is served by a burst of 8 transfers out of two $\times 8$ DRAMs, of which one is for data and the other is for ECC. As a result, the effective memory channel is only 32 bits wide. L1D and L2 caches and LLC are sector caches, where a 64B cache line is divided into eight 8B sectors to manage fine-grained data in the cache hierarchy.
- AG+ECC: AGMS with the AG_{split} scheme described in Section 6.2.4 and $2\times$ ABUS. As FG+ECC, AG+ECC also uses sector caches in L1D and L2 caches and LLC.

In addition to the above configurations with ECC, we also evaluate systems without ECC: CG, $CG_{\text{sub-ranked}}$, FG, and AG. These non-ECC configurations are identical to their ECC counterparts except that they do not support ECC. CG and $CG_{\text{sub-ranked}}$ do not have dedicated ECC DRAMs. FG and AG (for fine-grained accesses) do not transfer ECC, yielding twice the peak fine-grained data rates of FG+ECC and AG+ECC. Note that the AG schemes (AG and AG+ECC) can emulate the CG schemes (CG and CG+ECC) and the FG schemes (FG and FG+ECC) when we set all pages to coarse-grained or fine-grained, respectively.

Power models. We estimate DRAM power consumption using a power model developed by Micron Corporation [7]. For processor power analysis, we use the IPC-based mechanism presented in [13]: The maximum power per core is estimated as 16.8W based on a 32nm Xeon processor model using the McPAT 0.7 tool [72]; and half of the maximum power is assumed to be static (including leakage) with the other half being dynamic power that is proportional to IPC. In our experience, this rather simple measure of processor core power produces a power estimate that matches WATTCH [26] results well. Furthermore, our study focuses on main memory, and our mechanisms have minimal impact on the processor core's power behavior. We estimate LLC power using CACTI 6 [85]. To account for the cost of sector caches and the sub-ranked memory system with 2× ABUS and the register/demux, we add a 10% power penalty to the LLC and DRAM power consumption in AGMS, which we believe to be *very* conservative.

Table 6.3: PIN-based data page profiling.

Benchmark	Total data	Fine-grained data	Fraction of fine-grained data
SPEC			
`mcf`	1676MB	1676MB	100%
`omnetpp`	175MB	159MB	91%
`libquantum`	122MB	0.1MB	0.1%
`bzip2`	208MB	124MB	59%
`hmmer`	64MB	0.1MB	0.2%
`astar`	349MB	258MB	74%
`lbm`	409MB	6.3MB	1.5%
PARSEC			
`canneal`	158MB	157MB	99%
`streamcluster`	9MB	0.1MB	1.6%
SPLASH2			
`OCEAN`	56MB	0.1MB	0.1%
Olden			
`mst`	201MB	193MB	96%
`em3d`	89MB	28MB	31%
HPCS			
`SSCA2`	186MB	18MB	10%
Micro-benchmarks			
`Linked List`	178MB	178MB	100%
`GUPS`	192MB	64MB	33%
`STREAM`	47MB	0.1MB	0.2%

6.4 Results and Discussion

This section presents our analysis of AGMS: Section 6.4.1 provides data page profiling results and reports the storage overhead of the redundant information in fine-grained pages; Section 6.4.2 and Section 6.4.3 present cycle-based simulation results from 4 cores and 8 cores, respectively.

6.4.1 Page Profiling Results

We use PIN with the simple cache model described in Section 6.3 to profile fine-grained pages as well as total data pages. The fraction of fine-grained data pages is important because the storage overhead of ECC for fine-grained pages is twice that of coarse-grained ones. As shown in Table 6.3, the frac-

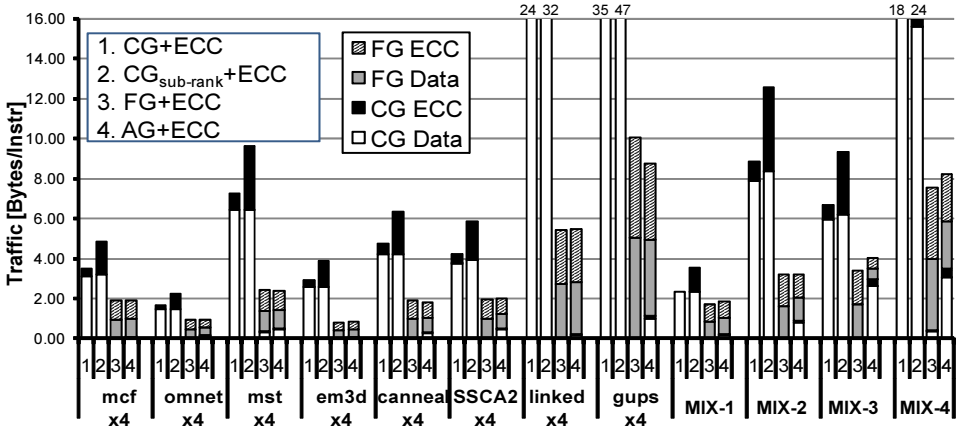
Table 6.4: Application mix for 4-core simulations.

MIX-1	mcf	omnetpp	mcf	omnetpp
MIX-2	SSCA2	lbm	astar	SSCA2
MIX-3	libquantum	hmmer	mst	mcf
MIX-4	SSCA2	Linked List	mst	hmmer

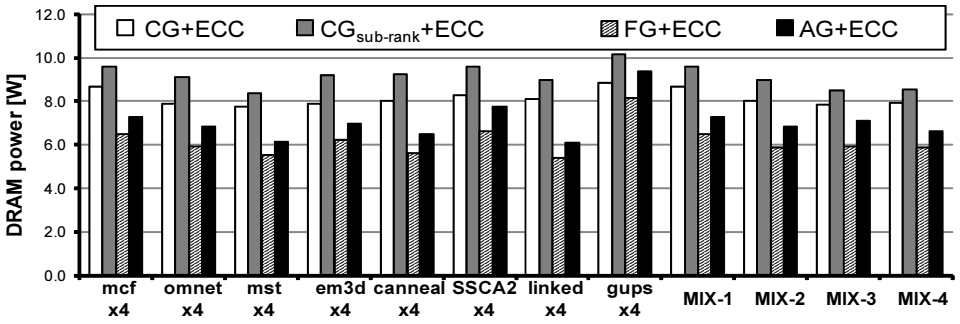
tion of fine-grained pages is low in hmmer, libquantum, lbm, streamcluster, OCEAN, em3d, SSCA2, GUPS, and STREAM, but is nearly 100% in many applications. Therefore, declaring a data page as fine grained must be done judiciously and used as part of overall system optimization (trading off performance and power-efficiency with a larger memory footprint). For some applications, the choice is clear. As shown in the performance analysis in Section 6.4.2, those applications that have a small fraction of fine-grained pages perform very well with AGMS. Others still gain from using AGMS, but require significantly more memory capacity. This particular tradeoff depends on many system-level parameters, and we leave this evaluation to future work.

6.4.2 4-core Cycle-Based Results

In this subsection, we present cycle-based simulation results of 4-core systems. Multi-programmed workloads are used for the 4-core system evaluation. We use 4 replicas of an application (suffix $\times 4$) as well as application mixes (Table 6.4). We utilize weighted speedup as the metric of system throughput. We use the fine-/coarse-grained decisions from our profiler. Both profiler and simulations used the same dataset, but the profiler was not heavily tuned.



(a) Traffic



(b) DRAM power

Figure 6.6: 4-core system off-chip traffic and power.

Off-Chip Traffic and Power. First, we compare the off-chip traffic and DRAM power consumption of CG+ECC, CG$_{\text{sub-ranked}}$+ECC, FG+ECC, and AG+ECC. Figure 6.6(a) shows the total memory traffic including ECC. AG+ECC reduces off-chip traffic by 66%, on average, compared to CG+ECC (56% excluding the micro-benchmarks: `GUPS`, `STREAM`, and `Linked List`).

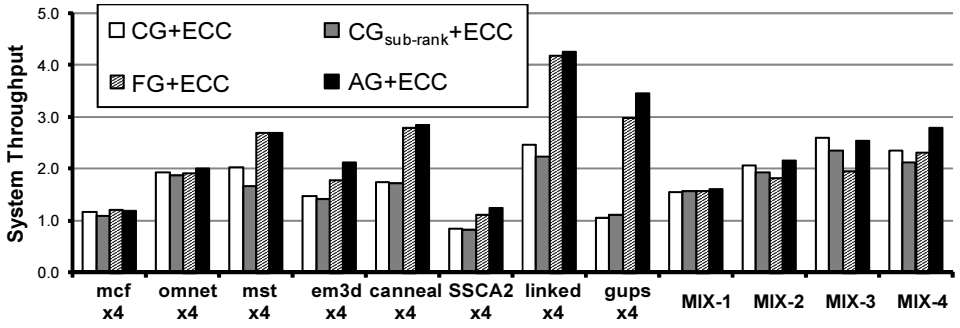
The reduced off-chip traffic leads to lower DRAM power consumption as shown in Figure 6.6(b). Remember that we added a conservative 10%

power penalty to the AG+ECC configurations. AG+ECC reduces DRAM power by 7% to 21% in most applications, and 14% on average. DRAM power actually increases for `GUPS`, but that is a result of the much higher performance obtained; efficiency is significantly improved as discussed below.

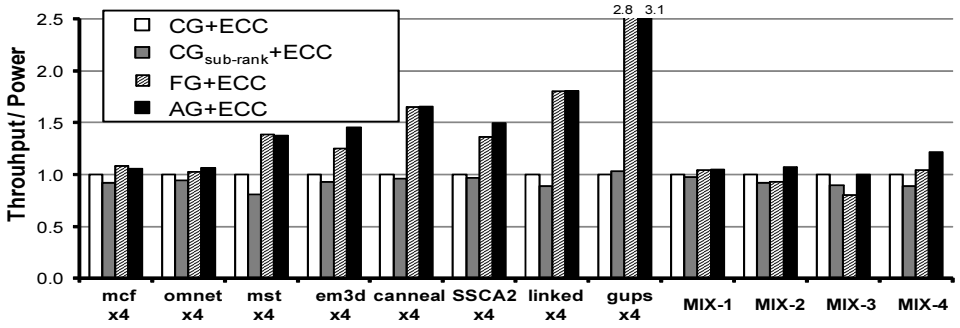
$CG_{\text{sub-ranked}}$+ECC, though the sub-ranked memory system was suggested for better energy efficiency, shows increased traffic and DRAM power consumption. This is mainly due to the cost of accessing redundant information; narrow access width in $CG_{\text{sub-ranked}}$+ECC necessitates high redundancy. FG+ECC, on the other hand, is effective in reducing traffic in most cases. FG+ECC, however, generates more traffic than AG+ECC. This is, again, due to ECC traffic; when spatial locality is high, coarse-grained accesses not only reduce miss rate but also minimize traffic including ECC. Though FG+ECC can minimize DRAM power consumption, since it touches only the necessary data, the reduced DRAM power consumption in FG+ECC does not necessarily lead to better performance or power efficiency as we show in the next paragraph.

Throughput and Power Efficiency. Figure 6.7(a) shows the system throughput of CG+ECC, $CG_{\text{sub-ranked}}$+ECC, FG+ECC, and AG+ECC. Overall, AG+ECC improves system throughput significantly: more than 130% in `GUPS`, 30% to 70% in `mst`, `em3d`, `SSCA2`, `canneal`, and `Linked List`, and 44% on average (22% on average excluding micro-benchmarks).

The results also show the advantage of adapting granularity compared to just using one of the mechanisms AGMS relies on fine-grained access (FG+ECC)



(a) System throughput



(b) Normalized system power efficiency

Figure 6.7: 4-core system throughput and power efficiency.

and memory sub-ranking (CG$_{\text{sub-ranked}}$+ECC). Even in these applications that can benefit from fine-grained access, AG+ECC consistently matches or outperforms FG+ECC.

We also evaluate applications with high spatial locality in Figure 6.8. As expected, when most pages are coarse-grained, AG+ECC does not degrade performance when compared to CG+ECC. The one exception is `bzip2`, which shows a very minor degradation, again as a result of inaccurate profiling. FG+ECC, on the other hand, degrades system throughput significantly: 17% in `libquantum`, 34% in `bzip2`, 36% in `OCEAN`, 50% in `streamcluster`, and

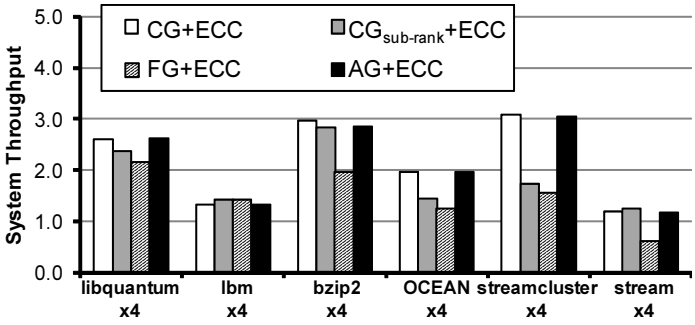


Figure 6.8: Applications with high spatial locality.

48% in `STREAM`. `lbm` is the only case that FG+ECC improves throughput (7%). Though `lbm` has very high DRAM page hit rate (82.6%), `lbm` references fewer than 4 words per cache line on average; hence, the FG scheme (FG+ECC), with lower total traffic, can slightly improve performance.

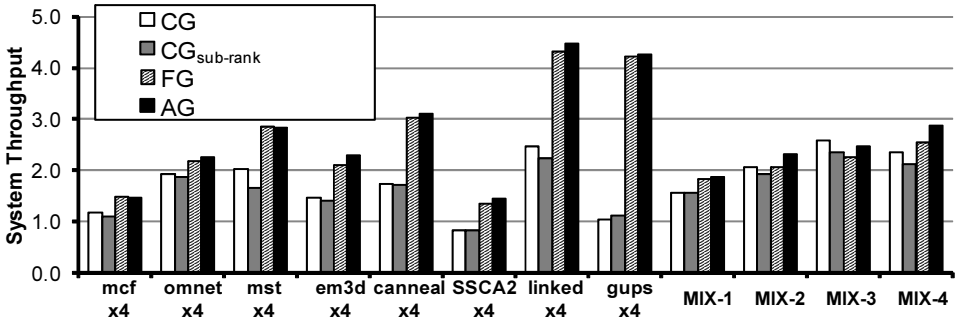
We also show that sub-ranking alone ($CG_{sub\text{-}ranked}$+ECC) significantly impacts performance because access latency increases as it takes longer to transfer a coarse-grained cache line into the cache over a narrower channel. This effect is pronounced in our system configuration that has limited latency hiding with only a single thread per core.

MIX-3 and `bzip2` are the only experiments we ran in which performance degraded with the AG scheme, but the degradation is less than 4%. In `bzip2`, most memory accesses fall within coarse-grained regions so AG+ECC practically do not affect execution (degrade throughput by less than 4%). MIX-3 suffers a more apparent performance degradation. This is most likely because of unfair DRAM scheduling between the applications. The performance of `mcf` is degraded, but that of `mst` is improved. In the meanwhile, the performance

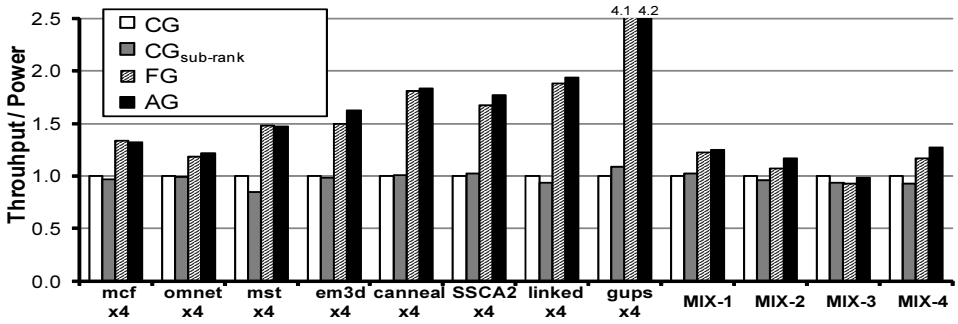
of `libquantum` and `bzip2` results almost remained unchanged. This is because `mst` has a relatively high DRAM page hit rate compared to `mcf` (40.5% vs. 19.1%). Since the FR-FCFS scheduling policy used in our memory controller tries to maximize memory throughput, `mst`'s requests are favored over those of `mcf`, leading to unfair resource allocation between `mcf` and `mst`. We believe that combining the adaptive granularity scheme with a better scheduling mechanism that provides fairness, such as parallelism-aware batch scheduling [86], can overcome this inefficiency in MIX-3. Note that AGMS does improve the performance of MIX-D in the 8-core simulation (see Section 6.4.3). MIX-D has two instances of each application of MIX-3, and AG+ECC improves performance when relative off-chip bandwidth is more scarce.

Recall that AGMS allows every page to have its own granularity. Hence, we can nullify all performance degradation; the OS can override the granularity hint if it suspects an increase in unfairness. The OS can set fine-grained regions in a more conservative way or even use only coarse-grained accesses.

We report the system power efficiency in terms of throughput per unit power (including cores, caches, and DRAM) in Figure 6.7(b). With reduced DRAM power consumption, AG+ECC improves the system power efficiency except in MIX-3. AG+ECC improves efficiency by 46% on average (24% excluding micro-benchmarks). The AG scheme degrades the throughput per power of MIX-3 by only less than 2%, which is due entirely to the 10% DRAM power penalty we conservatively added to account for the register/demux. In many current systems that are coarse-grained only, however, registered DIMMs



(a) System throughput



(b) Normalized system power efficiency

Figure 6.9: 4-core system throughput and power efficiency (non-ECC).

are already used for higher capacity. Compared to such systems, we do not expect any degradation in efficiency.

Note that we use power efficiency rather than a metric such as energy-delay product (EDP) because of our multi-programmed simulation methodology. All applications continue to run until the slowest-running application completes, and thus the amount of work varies from one configuration to another. Keep in mind that this is only for measuring power consumption, and the statistics for a specific core freeze when a core executes a fixed number of instructions so that the IPC comparison is based on the same number of in-

Table 6.5: Application mix for 8-core simulations.

MIX-A	mcf ×4	omnetpp ×4
MIX-B	SSCA2 ×2 omnetpp ×2	mcf ×2 mst ×2
MIX-C	SSCA2 mcf astar hmmer	omnetpp mst lbm bzip2
MIX-D	libquantum ×2 mst ×2	hmmer ×2 mcf ×2

structions across different configurations. We believe our approach is the most appropriate for our analysis. Alternative approaches, such as FIESTA [51], propose that in each experiment the same number of instructions is executed. Thus, the amount of work does not vary across experiments, allowing energy comparison. In FIESTA, however, the longest job runs alone at the end of a simulation, monopolizing shared resources such as shared caches and main memory. Because contention for shared memory resources is the crux of our research, applications running alone will skew the results.

We also present the throughput and the overall power efficiency of non-ECC configurations in Figure 6.9. While overall tendency is the same as the results of ECC configurations, AG shows even further gains: 61% in throughput and 67% in power efficiency on average (34% and 40% excluding micro-benchmarks). Note that we use the same profiler designed for AG+ECC, showing that AGMS is not sensitive to the profiler designs; using sub-optimal profiler in AG still provides significant gains in most applications. Furthermore, based on the improvements in FG and FG+ECC, simple per-thread decision (either all coarse-grained or all fine-grained) in the AGMS will lead to better performance and efficiency than coarse-grain-only baseline.

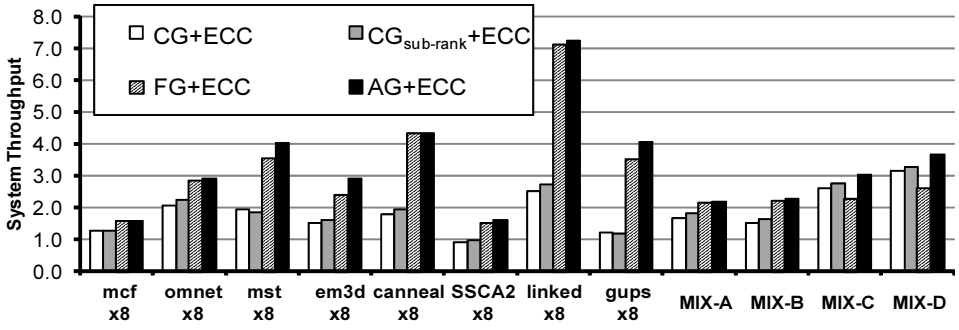


Figure 6.10: 8-core system throughput.

6.4.3 8-core Cycle-Based Results

This section presents results from 8-core system simulations. Systems with more cores are likely to have more memory channels, but we evaluate an 8-core system with a single channel. We chose this configuration because we expect systems in the future to generally have a lower memory bandwidth versus overall core arithmetic throughput. We use 8 replicas of an application (suffix $\times 8$) as well as application mixes (Table 6.5).

The results of 8-core simulations (Figure 6.10) show similar trends to those of 4-core systems. The performance and power gains, however, are higher in the more bandwidth-constrained 8-core configurations. Performance is improved by 85% (59% excluding micro-benchmarks) with AG+ECC. Though we do not show, AG (without ECC) shows further gains of 116% throughput improvement (87% excluding micro-benchmarks). Reductions in DRAM power consumption and traffic are similar to those in the 4-core experiments. In future systems, where off-chip bandwidth will be severely limited, it is likely that

virtually all applications will be bandwidth-bound; hence, we expect AGMS, which utilizes off-chip bandwidth more efficiently, to be even more beneficial.

6.5 Related Work

Caches. While orthogonal to our research on adaptive memory access granularity, work on cache architectures that support fine-grained data management is a necessary component of our design. These architectures provide mechanisms to maintain valid and dirty information at a granularity finer than just an entire cache line.

We already discussed the sector cache [75] in Section 3.3.2. The decoupled sectored cache [104] and a pool-of-sectors cache [97] refine the sector design and enable more elastic mapping between sectors and tags to reduce the miss rate of a sector cache. The dual cache [45] uses two L1 caches, one with large line size and the other with small line size. Similarly, the adaptive line size cache [120] gradually changes cache line size. Other techniques, spatial footprint prediction [67] and spatial pattern prediction [32], utilize a hardware predictor to reduce fetch traffic between L1 and L2 or to save leakage power by applying power-gating to the subblocks that are predicted to be non-referenced; and the line-distillation cache [93] splits the LLC into a line-oriented cache and a word-oriented cache.

These techniques assume that off-chip memory can be accessed with fine granularity, which is unfortunately no longer true because modern DRAM systems evolved to provide high bandwidth with coarse access granularity.

Our adaptive granularity memory system re-enables these fine-grained cache-management techniques. AGMS can be combined with any fine-grained cache-management technique, and in this dissertation we use a "sector cache" as a simple example.

DRAM Systems. As reviewed in Section 3.3.1, there are several proposals that can enable fine-grained accesses in modern DRAM systems. The many-narrow-channels approach such as Cray Black Widow [11] provides high throughput for fine-grained accesses, but is inherently expensive due to ABUS pin and redundancy overheads. The sub-ranked DRAM systems are also proposed and studied: Rambus's micro-threading [124] and threaded memory module [8, 125]; HP's MC-DIMM [13, 14]; Mini-rank memory system [140]; and Convey's S/G DIMM [25]. Most of these proposals focus on energy efficiency of coarse-grained accesses by mitigating the "overfetch" problem. Though slightly discussed in [13, 25, 124], to the best of our knowledge, we provide the first quantitative evaluation of sub-ranked memory systems for fine-grained access and ECC support.

Other Related Work. Our adaptive granularity memory system is closely related to the Impulse memory controller [136] that uses a shadow address space to provide the illusion of contiguous data for non-unit stride or indexed gather/scatter accesses. The Impulse memory controller translates a shadow address to potentially multiple physical addresses, and then collects multiple fine-grained data blocks to form a dense coarse-grained data block, reducing traffic on the bus between the cache controller and the off-chip memory

controller. Unfortunately, on chip memory controllers in recent architectures as well as ever-increasing memory access granularity neutralize Impulse's advantages. Moreover, it is unclear how to support both fine-grained memory accesses and ECC with Impulse.

Similar to the tradeoff between storage efficiency and fine-grained throughput presented in this dissertation, RAID-Z implemented in the ZFS file system [5] uses different reliability schemes for stores with varying granularities. RAID-Z, however, has completely different management mechanisms and characteristics because it deals with the much larger data blocks of hard disk drives.

6.6 Summary

We present a novel architecture that enables a tradeoff between storage efficiency and fine-grained throughput and power efficiency. The adaptive granularity memory system (AGMS) utilizes finite off-chip bandwidth more efficiently: Fine-grained memory access minimizes off-chip traffic and reduces DRAM power consumption by not transferring unnecessary data, while also increasing concurrency in memory accesses and improving performance; coarse-grained memory access minimizes control and redundancy overheads and can potentially reduce miss rate. In 4-core systems, AGMS, even with higher ECC overhead, improves system throughput and power efficiency by 44% and 46% respectively. It also reduces memory power consumption and traffic by 14% and 66%. AGMS, though we design it for ECC-enabled memory systems, is also beneficial for non-ECC memory systems and provides further gains:

Throughput improvement is 61%; and power efficiency gain is 67%. When off-chip bandwidth is more constrained, as in our 8-core configurations, AGMS is even more beneficial and important: System throughput increases are 85% with ECC and 116% without ECC. Thus, we conclude that adapting access granularity will be more effective in future systems, where off-chip bandwidth is relatively scarce. Note that these promising results were obtained with a very rough profiler for determining the preferred granularity. We expect to improve on this work by developing more sophisticated heuristics, utilizing more programmer knowledge, and studying dynamic adaptivity.

Chapter 7

Conclusions and Future Research Directions

This dissertation presents and evaluates a set of memory resiliency mechanisms. We demonstrate that the proposed mechanisms reduce on-chip cache area and power consumption as well as improve system-wide energy efficiency with minimal impact on performance. Also, the proposed mechanisms are flexible so that the error tolerance level and access granularity can be tuned to dynamic user, system, and environmental demands.

The key to the efficient and flexible resiliency mechanisms are (i) two-tiered protection that minimizes the common case resiliency penalty, and (ii) virtualizing redundant information that decouples data and its associated ECC, allowing dynamic adaptation of error tolerance levels and access granularities.

Two-tiered LLC protection. Two-tiered protection, introduced in Section 3.1, is a generalization scheme of previously proposed decoupled error detection/correction [64, 71, 99]. Two-tiered protection splits the common case error detection/light-weight error correction and the uncommon case error correction. The uniform T1EC provides simple and efficient error detection (or light-weight error correction), while the T2EC provides strong error correc-

tion capability. Furthermore, we store the T2EC within memory namespace to avoid dedicating resources to the T2EC.

In Chapter 4, we apply two-tiered protection with virtualized T2EC to LLC protection and develop Memory Mapped ECC (MME) and ECC FIFO. Both mechanisms saves on-chip area and power by $15-25\%$ and $9-18\%$, respectively, while performance penalty is only 0.7% on average. Two-tiered LLC protection is flexible in choosing ECC codes, and even very strong ECC codes can be used at low area and power overheads. We describe a set of example two-tiered ECC codes and show the error protection tradeoffs also.

Virtualized Redundant Information. Virtualizing redundant information is introduced in Section 3.2 and allows single hardware to use different ECC schemes based on user, system, and environmental demands.

In Chapter 5, we develop Virtualized ECC (V-ECC) for chipkill-correct level main memory protection. We augment the virtual memory interface to store all or part of redundant information within memory namespace itself. The simpler T1EC, with the virtualized T2EC, relaxes DRAM module design constraints so that more energy efficient, wider, DRAM configurations can be used for chipkill-correct. We show that V-ECC can improve system energy efficiency by 12%, while performance impact is only $1-2\%$. We also develop V-ECC with Non-ECC DIMMs, enabling memory protection in low-cost or performance-oriented platforms.

Another key advantage with V-ECC is the flexible and adaptive error tolerance level. With the virtualized T2EC, it is possible to use different ECC for different memory pages. As a result, we avoid unnecessary cost for reliability by adapting the error tolerance level.

Adaptive Granularity. One of the limitation with uniform ECC is that the ECC code determines memory access granularity, and typically we use coarse-grained accesses to amortize ECC overhead over a large data block. We present that this coarse-grain-only system squanders two important resources: Off-chip bandwidth is wasted, and unnecessary power is consumed for transferring unused data bits.

In Chapter 6, we develop the adaptive granularity memory system (AGMS) that can adapt memory access granularity according to application memory access patterns. AGMS relies on virtualizing redundant information to enable ECC support for fine-grained accesses and employ sub-ranked DRAM and the sector cache described in Section 3.3. AGMS achieves 44% higher throughput and 46% better power efficiency than conventional coarse-grain-only systems in a 4-core CMP. In future CMP systems, where off-chip bandwidth is more scarce, AGMS provides even better system efficiency, while providing memory resiliency.

7.1 Future Research Directions

The key mechanisms, we develop in this dissertation, can be applied to other architectures, even for different purposes, and we present the short-term

future work in Section 7.1.1. Also, dynamically tunable resiliency is important and necessary in future power-limited systems, and we describe the long-term future work on system-level dynamic tunable resiliency in Section 7.1.2.

7.1.1 Extending the Proposed Resiliency Mechanisms

We plan to apply the proposed resiliency mechanisms to other architectures. This includes two-tiered protection for low-V_{CC} caches and GPU on-chip memory, V-ECC for GPU main memory and emerging non-volatile memory (NVRAM), applying V-ECC mechanisms for managing generic meta-data, and AGMS for vector/GPU architectures.

Two-Tiered Protection for Low-V_{CC} Caches. Recent proposal on low V_{CC} caches reduce error margins to save energy consumption. This, however, makes many memory cells, which operate correctly under nominal V_{CC}, faulty in low-V_{CC} mode, appearing as random bit errors.

We can design two-tiered protection for low-V_{CC} caches; the T1EC detects many random-bit errors and correct small number of errors for the common case accesses, while the T2EC, off-loaded to main memory namespace, can handle the uncommon case of T1EC DUE. With the T2EC off-loaded to main memory, the two-tiered approach can mitigate the overhead of prior solutions in low-V_{CC} caches.

Two-Tiered Protection for GPU On-Chip Memory. Graphics processing units (GPUs) are already being used as compute accelerators, and memory protection is essential for integrating GPUs in larger high-end systems. For

instance, NVidia's Fermi GPU [89] implement uniform ECC for on-chip memory. As we discussed in Section 2, this uniform ECC dedicates storage and bandwidth to redundant information, yielding lower system efficiency. For instance, uniform SEC (single bit-error correcting) code at a 32-bit granularity requires 22% static storage overhead.

With two-tiered protection, we can accommodate only the T1EC uniformly (1-bit parity per 32-bit data word) and store the SEC T2EC in main memory namespace. Although GPU on-chip memory does not maintain dirty words explicitly, the compiler can help identify dirty words in the explicitly managed GPU on-chip memory so that we can write-back only dirty word T2EC to main memory.

V-ECC for GPU Memory. Main memory for GPU is high-bandwidth memory products (such as GDDR5), where the dedicated storage for ECC is not available. V-ECC for Non-ECC DIMMs can be straightforwardly applied and enable memory protection in GPU systems.

Already, NVidia's Fermi [89] supports SEC-DED protection with GDDR5 memory, and we believe that the mechanism is quite similar to the technique we present in Section 5, but without adaptivity. We argue that SEC-DED (and static in-memory ECC) is not enough for GPU systems. Unlike general purpose systems, GPU systems do not utilize memory modules so the entire card needs to be replaced if only a single memory device fails (or manifests any intolerable hard failures). Hence, more stringent protection mechanisms (e.g., chipkill-correct) are required in GPU systems eventually to maintain

low cost and high availability. Supporting chipkill-correct level redundancy in GPUs, however, will significantly degrade system efficiency. The flexible memory protection enabled with V-ECC can allow GPU systems to support just the required error tolerance level. In essence, the cost of chipkill-correct is only be incurred after a memory device fails, enabling graceful performance degradation at low cost.

V-ECC for Emerging NVRAM. V-ECC can also protect emerging NVRAM such as phase-change memory (PCRAM) or memristors. This new memory technology has finite write-endurance so tolerating hard failures is a challenge. V-ECC can dynamically adapt error tolerance levels to NVRAM wear-out status; only minimal protection is applied to brand new devices, maximizing performance, while the error tolerance levels are increased as necessary to NVRAM wear-out status.

V-ECC Mechanism for Generic Meta-Data Storage. Recently, there have been proposals leveraging meta-data [19, 40, 47, 114, 121, 128] in memory systems for security, concurrency checking, directory-based cache coherency, and safety checking. Like uniform ECC, we can accommodate meta-data along with data in dedicated DRAM chips. This, however, not only is inefficient but also requires custom memory modules for storing uniform meta-data.

The mechanisms presented in Section 5 is general enough to manage any meta-data in memory systems. We can virtualize meta-data within memory namespace, similar to V-ECC; this allows any form of meta-data and selectively enable/disable different meta-data for user demands.

AGMS for Vector/GPU Architectures. In applications running on vector processors or GPUs, the off-chip bandwidth is often the bottleneck. Also, these applications leverage gather/scatter memory operations to handle irregular patterns in a regular SIMD datapath. Thus, AGMS can naturally fit to these high-throughput computing platforms, where both better off-chip bus utilization and reliability are necessary.

7.1.2 Dynamically Tunable Resiliency

In this dissertation, we focus on hardware frameworks that enable flexible and tunable resiliency mechanisms and evaluate the potential of this approach. The unique combination of two-tiered protection and virtualizing redundant information enables new tradeoffs between cost, error tolerance, and performance. Importantly, these new tradeoffs can be tuned at runtime to meet dynamically changing reliability needs. In future research, we will develop runtime support for the adaptive/tunable protection and study how applications exploit dynamically tunable resiliency.

One research direction is to design a runtime module that monitors system wear-out status or soft error rates; this runtime module sets the default error tolerance level and takes the reliability demands from applications so that the system operates at optimal energy efficiency, while guaranteeing the required reliability.

Another research direction is to augment programming languages to embrace error tolerance and access granularity. This can be achieved by adding

annotations to exiting programming languages or overload runtime support functions. Then, the programmer can request the required error tolerance level and the preferred access granularity for each memory object or region.

Finally, we can exploit the adaptive error tolerance levels for mitigating the reliability cost in large-scale computing platforms. Currently, such systems implement periodic co-ordinated checkpointing and roll-back to the previous checkpoint upon an error. This overhead of taking periodic checkpointing is increasing and predicted to be unacceptably high in future systems. With adaptive resiliency, we can selectively increase error tolerance levels of nodes that suffer frequent errors. Even with increased checkpointing periods, the nodes with stronger error tolerance can make progress, by tolerating errors rather than rolling-back to a checkpoint.

Bibliography

[1] Cross-layer reliability. http://www.xlayer.org.

[2] HPC challenge. http://icl.cs.utk.edu/hpcc/hpcc_results.cgi.

[3] HPCS scalable synthetic compact application (SSCA). http://www.highproductivity.org/SSCABmks.htm.

[4] Linked list traversal micro-benchmark. http://www-sal.cs.uiuc.edu/~zilles/llubenchmark.html.

[5] ZFS the last word in file systems. http://www.opensolaris.org/os/community/zfs/docs/zfs_last.pdf.

[6] Calculating memory system power for DDR2. Technical Report TN-47-04, Micron Technology, 2005.

[7] Calculating memory system power for DDR3. Technical Report TN-41-01, Micron Technology, 2007.

[8] Rambus and Kingston co-develop threaded module prototype for multi-core computing, 2009.

[9] M. Abbott, D. Har, L. Herger, M. Kauffmann, K. Mak, J. Murdock, C. Schulz, T. B. Smith, D. Tremaine, D. Yeh, and L. Wong. Durable

memory RS/6000^{TM} system design. In *Proc. the 24th Int'l Symp. Fault-Tolerant Computing (FTCS)*, Jun. 1994.

[10] J. Abella, J. Carretero, P. Chaparro, X. Vera, and A. Gonzalez. Low V_{ccmin} fault-tolerant cache with highly predictable performance. In *Proc. the 42nd IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Dec. 2009.

[11] D. Abts, A. Bataineh, S. Scott, G. Faanes, J. Schwarzmeier, E. Lundberg, M. Byte, and G. Schwoerer. The Cray Black Widow: A highly scalable vector multiprocessor. In *Proc. the Int'l Conf. High Performance Computing, Networking, Storage, and Analysis (SC)*, Nov. 2007.

[12] N. Aggarwal, J. E. Smith, K. K. Saluja, N. P. Jouppi, and P. Ranganathan. Implementing high availability memory with a duplication cache. In *Proc. the 41st IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Nov. 2008.

[13] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber. Future scaling of processor-memmory interfaces. In *Proc. the Int'l Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2009.

[14] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi. Multicore DIMM: An energy efficient memory module with independently controlled DRAMs. *IEEE Computer Architecture Letters*, 8(1):5–8, Jan. - Jun. 2009.

[15] AMD. BIOS and kernel developer's guide for AMD NPT family 0Fh processors, Jul. 2007.

[16] H. Ando, K. Seki, S. Sakashita, M. Aihara, R. Kan, K. Imada, M. Itoh, M. Nagai, Y. Tosaka, K. Takahisa, and K. Hatanaka. Accelerated testing of a 90nm SPARC64 V microprocessor for neutron SER. In *Proc. the IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE)*, Apr. 2007.

[17] S. Ankireddi and T. Chen. Challenges in thermal management of memory modules. `http://electronics-cooling.com/html/2008_feb_a3.php`.

[18] J. Baggio, D. Lambert, V. Ferlet-Cavrois, C. D'hose, K. Hirose, H. Saito, J. M. Palau, F. Saigne, B. Sagnes, N. Buard, and T. Carriere. Neutron-induced SEU in bulk and SOI SRAMs in terrestrial environments. In *Proc. the IEEE 42nd Ann. Int'l Reliability Physics Symp. (IRPS)*, 2004.

[19] L. Baugh, N. Neelakantam, and C. Zilles. Using hardware memory protection to build a high-performance, strongly-atomic hybrid transactional memory. In *Proc. the 35th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2008.

[20] R. C. Baumann and E. B. Smith. Neutron-induced Boron fission as a major source of soft errors in deep submicron SRAM devices. In *Proc. the IEEE 41st Ann. Int'l Reliability Physics Symp. (IRPS)*, 2000.

[21] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. Technical Report TR-811-08, Princeton Univ., Jan. 2008.

[22] R. E. Blahut. *Algebraic Codes for Data Transmission.* Cambridge University Press, 2003.

[23] R. C. Bose and D. K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3:68–79, 1960.

[24] D. C. Bossen. B-adjacent error correction. *IBM J. Res. and Dev.*, 14(4):402–408, 1970.

[25] T. M. Brewer. Instruction set innovations for the Convey HC-1 computer. *IEEE Micro*, 30(2):70–79, 2010.

[26] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. the 27th Ann. Int'l Symp. Computer Architecure (ISCA)*, Jun. 2000.

[27] B. H. Calhoun and A. P. Chandrakasan. A 256kb sub-threshold SRAM in 65nm CMOS. In *Proc. the Int'l Solid State Circuits Conf. (ISSCC)*, Feb. 2006.

[28] P. Calingaert. Two-dimensional parity checking. *J. ACM*, 8(2):186–200, Apr. 1961.

[29] E. H. Cannon, D. D. Reinhardt, M. S. Gordon, and P. S. Makowenskyj. SRAM SER in 90, 130 and 180 nm bulk and SOI technologies. In *Proc. the IEEE 42nd Ann. Int'l Reliability Physics Symp. (IRPS)*, 2004.

[30] M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. Technical Report TR-483-95, Princeton University, 1995.

[31] L. Chang, D. M. Fried, J. Hergenrother, J. W. Sleight, R. H. Dennard, R. K. Montoye, L. Sekaric, S. J. McNab, A. W. Topol, C. D. Adams, K. W. Guarini, and W. Haensch. Stable SRAM cell design for the 32nm node and beyond. In *Digest of Technical Papers of Symp. VLSI Technology*, Jun. 2005.

[32] C. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos. Accurate and complexity-effective spatial pattern prediction. In *Proc. the 10th Int'l Symp. High-Performance Computer Architecture (HPCA)*, Feb. 2004.

[33] C. L. Chen. Symbol error correcting codes for memory applications. In *Proc. the 26th Ann. Int'l Symp. Fault-Tolerant Computing (FTCS)*, Jun. 1996.

[34] C. L. Chen and M. Y. Hsiao. Error-correcting codes for semiconductor memory applications: A state-of-the-art review. *IBM J. Res. and Dev.*, 28(2):124–134, Mar. 1984.

[35] Z. Chisti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu. Improving cache lifetime reliability at ultra-low voltages. In *Proc. the 42nd IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Dec. 2009.

[36] R. Danilak. Transparent error correction code memory system and method. US Patent, US 7,117,421, Oct. 2006.

[37] T. J. Dell. A white paper on the benefits of chipkill-correct ECC for PC server main memory. IBM Microelectronics Division, Nov. 1997.

[38] T. J. Dell. System RAS implications of DRAM soft errors. *IBM J. Res. and Dev.*, 52(3):307– 314, 2008.

[39] N. Derhacobian, V. A. Vardanian, and Y. Zorian. Embedded memory reliability: The SER challenge. In *Proc. the Records of the 2004 Int'l Workshop on Memory Technology, Design, and Testing*, Aug. 2004.

[40] J. Devietti, C. Blundell, M. M. K. Martin, and S. Zdancewic. HardBound: Architectural support for spatial safety of the C programming language. In *Proc. the 13th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2008.

[41] Digital Equipment Corporation. *Alpha 21264 Microprocessor Hardware Reference Manual*, Jul. 1999.

[42] Earl Joseph II. GUPS (giga-updates per second) benchmark. `http://www.dgate.org/~brg/files/dis/gups/`.

[43] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 28(3):42–53, 2008.

[44] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proc. the 34th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2007.

[45] A. Gonzalez, C. Aliagas, and M. Valero. A data cache with multiple caching strategies tuned to different types of locality. In *Proc. the Int'l Conf. Supercomputing (ICS)*, Jul. 1995.

[46] C. S. Guenzer, E. A. Wolicki, and R. G. Allas. Single event upset of dynamic RAMs by neutrons and protons. *IEEE Trans. Nuclear Science*, 26(6):5048–5052, Dec. 1979.

[47] L. Gwennap. Alpha 21364 to easse memory bottleneck. *Micro-processor Report*, Oct. 1998.

[48] M. J. Haertel, R. S. Polzin, A. Kocev, and M. B. Steinman. ECC implementation in non-ECC components. US Patent Pending, Serial No. 725,922, Sep. 2008.

[49] G. Hamerly, E. Perelman, J. Lau, and B. Calder. SimPoint 3.0: Faster and more flexible program analysis. In *Proc. the Workshop on Modeling, Benchmarking and Simulation (MoBS)*, Jun. 2005.

[50] R. W. Hamming. Error correcting and error detecting codes. *Bell System Technical J.*, 29:147–160, Apr. 1950.

[51] A. Hilton, N. Eswaran, and A. Roth. FIESTA: A sample-balanced muilti-program workload methodology. In *Proc. the Workshop on Modeling, Benchmarking and Simulation (MoBS)*, Jun. 2009.

[52] A. Hocquenghem. Codes correcteurs d'erreurs. *Chiffres (Paris)*, 2:147–156, 1959.

[53] M. Y. Hsiao. A class of optimal minimum odd-weight-column SEC-DED codes. *IBM J. Res. and Dev.*, 14:395–301, 1970.

[54] M. Y. Hsiao, D. C. Bossen, and R. T. Chien. Orthogonal latic square codes. *IBM J. Res. and Dev.*, 14(4):390–394, Jul. 1970.

[55] J. Huynh. *White Paper: The AMD Athlon MP Processor with 512KB L2 Cache*, May 2003.

[56] IBM. Enhancing IBM Netfinity server reliability. `ftp://ftp.software.ibm.com/systems/support/system_x/chipkif1.pdf`, 1999.

[57] Intel Corp. *Intel(R) IA-64 and IA-32 Architecture Software Developer's Manual*, Mar. 2010.

[58] S.-M. Jung, H. Lim, W. Cho, H. Cho, H. Hong, J. Jeong, S. Jung, H. Park, B. Son, Y. Jang, and K. Kim. Soft error immune 0.46 μm^2 SRAM cell with MIM node capacitor by 65nm CMOS technology for ultra high speed SRAM. In *Technical Digest of IEEE Int'l Electron Devices Meeting (IEDM)*, Dec. 2003.

[59] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd. Power7: IBM's next-generation server processor. *IEEE Micro*, 30(2):7–15, 2010.

[60] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proc. the 28th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun.-Jul. 2001.

[61] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway. The AMD Opteron processor for multiprocessor servers. *IEEE Micro*, 23(2):66–76, Mar.-Apr. 2003.

[62] G. H. Kemmetmueller. RAM error correction using two dimensional parity checking. US Patent, US 4,183,463, Jan. 1980.

[63] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. C. Hoe. Multi-bit error tolerant caches using two-dimensional error coding. In *Proc. the 40th IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Dec. 2007.

[64] S. Kim. Area-efficient error protection for caches. In *Proc. the Conf. Design Automation and Test in Europe (DATE)*, Mar. 2006.

[65] S. Kim and A. K. Somani. Area efficient architectures for information integrity in cache memories. In *Proc. the 26th Ann. Int'l Symp. Computer Architecture (ISCA)*, May 1999.

[66] J. P. Kulkarni, K. Kim, and K. roy. A 160mV robust schmitt trigger based subthreshold SRAM. *IEEE J. Solid-State Circuits*, 42(10):2303–2313, Oct. 2007.

[67] S. Kumar and C. Wilkerson. Exploiting spatial locality in data caches using spatial footprints. In *Proc. the 25th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 1998.

[68] R. Kuppuswamy, S. R. Sawant, S. Balasubramanian, P. Kaushik, N. Natarajan, and J. D. Gilbert. Over one million TPCC with a 45nm 6-core Xeon CPU. In *Proc. the Int'l Solid State Circuits Conf. (ISSCC)*, Feb. 2009.

[69] H.-H. S. Lee, G. S. Tyson, and M. K. Farrens. Eager writeback - a technique for improving bandwidth utilization. In *Proc. the 33rd IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Nov.-Dec. 2000.

[70] K. Lee, A. Shrivastava, I. Issenin, N. Dutt, and N. Venkatasubramanian. Mitigating soft error failures for multimedia applications by selective data protection. In *Proc. the Int'l Conf. Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Oct. 2006.

[71] L. Li, V. S. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Soft error and energy consumption interactions: A data cache perspective. In *Proc. the Int'l Symp. Low Power Electronics and Design (ISLPED)*, Aug. 2004.

[72] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proc. the 42nd Ann. IEEE/ACM Int'l Symp Microarchitecture (MICRO)*, Dec. 2009.

[73] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *Proc. the 35th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2008.

[74] S. Lin and D. J. Costello Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.

[75] J. S. Liptay. Structural aspects of the system/360 model 85, part II: The cache. *IBM Sys. J.*, 7:15–21, 1968.

[76] G. H. Loh, S. Subramaniam, and Y. Xie. Zesto: A cycle-level simulator for highly detailed microarchitecture exploration. In *Proc. the Int'l Symp. Performance Analysis of Software and Systems (ISPASS)*, Apr. 2009.

[77] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. PIN: Building customized program analysis tools with dynamic instrumentation. In *Proc. the ACM Conf. Programming Language Design and Implementation (PLDI)*, Jun. 2005.

[78] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. SIMICS: A full system simulation platform. *IEEE Computer*, 35:50–58, Feb. 2002.

[79] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong. Characterization of multi-bit soft error events in advanced SRAMs. In *Technical Digest of the IEEE Int'l Electron Devices Meeting (IEDM)*, Dec. 2003.

[80] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Computer Architecture News (CAN)*, 33:92–99, Nov. 2005.

[81] T. C. May and M. H. Woods. A new physical mechanism for soft errors in dynamic memories. In *Proc. the 16th Ann. IEEE Int'l Reliability Physics Symp. (IRPS)*, Apr. 1978.

[82] J. D. McCalpin. STREAM: Sustainable memory bandwidth in high performance computers. `http://www.cs.virginia.edu/stream/`.

[83] M. McTague and H. David. Fully buffered DIMM (FB-DIMM) design considerations. Intel Developer Forum (IDF), Feb. 2004.

[84] Micron Corp. *Micron 1 Gb* ×4, ×8, ×16*, DDR3 SDRAM: MT41J256M4, MT41J128M8, and MT41J64M16*, 2006.

[85] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0. Technical report, HP Labs., Apr. 2009.

[86] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In

Proc. the 35h Ann. Int'l Symp. Computer Architecture (ISCA), Jun. 2008.

[87] J. A. Nerl, K. Pomaranski, G. Gostin, A. Walton, and D. Soper. System and method for controlling application of an error correction code. US Patent, US 7,437,651, Oct. 2004.

[88] J. A. Nerl, K. Pomaranski, G. Gostin, A. Walton, and D. Soper. System and method for applying error correction code (ECC) erasure mode and clearing recorded information from a page deallocation page. US Patent, US 7,313,749, Dec. 2007.

[89] NVIDIA. Fermi architecture. `http://www.nvidia.com/object/fermi_architecture.html`.

[90] K. Osada, K. Yamaguchi, and Y. Saitoh. SRAM immunity to cosmic-ray-induced multierrors based on analysis of an induced parasitic bipolar effect. *IEEE J. Solid-State Circuits*, 39:827–833, May 2004.

[91] A. M. Patel and M. Y. Hsiao. An adaptive error correction scheme for computer memory system. In *Proc. the Fall Joint Computer Conf., part I*, Dec. 1972.

[92] N. Quach. High availability and reliability in the Itanium processor. *IEEE Micro*, 20(5):61–69, Sept.-Oct. 2000.

[93] M. K. Qureshi, M. A. Suleman, and Y. N. Patt. Line distillation: Increasing cache capacity by filtering unused words in cache lines. In

Proc. the 13th Int'l Symp. High Performance Computer Architecture (HPCA), Feb. 2007.

[94] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *J. Soc. for Industrial and Applied Math.*, 8:300–304, Jun. 1960.

[95] S. Rixner, W. J. Dally, U. J. Kapasi, P. R. Mattson, and J. D. Owens. Memory access scheduling. In *Proc. the 27th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2000.

[96] P. Roche, F. Jacquet, C. Callat, and J.-P. Schoellkopf. An alpha immune and ultra low neutron SER high density SRAM. In *Proc. the IEEE 42nd Ann. Int'l Reliability Physics Symp. (IRPS)*, Apr. 2004.

[97] J. B. Rothman and A. J. Smith. The pool of subsectors cache design. In *Proc. the 13th Int'l Conf. Supercomputing (ICS)*, Jun. 1999.

[98] J. B. Rothman and A. J. Smith. Sectored cache design and performance. Technical Report UCB/CSD-99-1034, University of California, Berkeley, Jan. 1999.

[99] N. N. Sadler and D. J. Sorin. Choosing an error protection scheme for a microprocessor's L1 data cache. In *Proc. the Int'l Conf. Computer Design (ICCD)*, Oct. 2006.

[100] G. Schindlbeck and C. Slayman. Neutron-induced logic soft errors in DRAM technology and their impact on reliable server memory. In

Proc. the IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE), Apr. 2007.

[101] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM errors in the wild: A large-scale field study. In *Proc. the 11th Int'l Joint Conf. Measurement and Modeling of Computer Systems (SIGMETRICS)*, Jun. 2009.

[102] J. R. Schwank, V. Ferlet-Cavrois, M. R. Shaneyfelt, P. Paillet, and P. E. Dodd. Radiation effets in SOI technologies. *IEEE Trans. Nuclear Science*, 50(3), Jun. 2003.

[103] 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 *Proc. the IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE)*, Apr. 2007.

[104] A. Seznec. Decoupled sectored caches: Conciliating low tag implementation cost. In *Proc. the 21st Ann. Int'l Symp. Computer Architecture (ISCA)*, Apr. 1994.

[105] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. Wiley, Dec. 2004.

[106] C. Slayman. Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations. *IEEE Trans. Device and Materials Reliability*, 5:397– 404, Sep. 2005.

[107] C. Slayman. Impact of error correction code and dynamic memory reconfiguration on high-reliability/low-cost server memory. In *Proc. the IEEE Int'l Integrated Reliability Workshop (IIRW)*, Oct. 2006.

[108] C. Slayman. Impact and mitigation of DRAM and SRAM soft errors. IEEE SCV Reliability Seminar `http://www.ewh.ieee.org/r6/scv/rl/articles/Soft%20Error%20mitigation.pdf`, May 2010.

[109] Standard Performance Evaluation Corporation. SPEC CPU 2006. `http://www.spec.org/cpu2006/`, 2006.

[110] J. Standards. JESD89 measurement and reporting of alpha particles and terrestrial cosmic ray-induced soft errors in semiconductor devices, JESD89-1 system soft error rate (SSER) method and JESD89-2 test method for alpha source accelerated soft error rate, 2001.

[111] J. Standards. JESD 79-2e DDR2 SDRAM specification, 2008.

[112] J. Standards. JESD 79-3b DDR3 SDRAM specification, 2008.

[113] D. Strukov. The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories. In *Proc. Asilomar Conf. Signals Systems and Computers*, October 2006.

[114] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas. Secure program execution via dynamic information flow tracking. In *Proc. the 11th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2004.

[115] Sun Microsystems Inc. *UltraSPARC®III Cu*, Jan. 2004.

[116] Sun Microsystems Inc. *OpenSPARC T2 System-On-Chip (SOC) Microarchitecture Specification*, May 2008.

[117] M. Talluri and M. D. Hill. Surpassing the TLB performance of superpages with less operating system support. In *Proc. the 6th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 1994.

[118] J. M. Tendler, J. S. Dodson, J. S. F. Jr., H. Le, and B. Sinharoy. POWER4 system microarchitecture. *IBM J. Res. and Dev.*, 46(1):5–25, Jan. 2002.

[119] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. CACTI 5.1. Technical report, HP Labs., Apr. 2008.

[120] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji. Adapting cache line size to application behavior. In *Proc. the Int'l Conf. Supercomputing (ICS)*, Jun. 1999.

[121] G. Venkataramani, I. Doudalis, Y. Solihin, and M. Prvulovic. Flexitaint: A programmable accelerator for dynamic taint propagation. In *Proc. the 14th Int'l Symp. High Performance Computer Architecture (HPCA)*, Feb. 2008.

[122] Violin Memory Inc. Scalable memory applicance. `http://violin-memory.com/DRAM`.

[123] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. DRAMsim: A memory-system simulator. *SIGARCH Computer Architecture News (CAN)*, 33:100–107, Sep. 2005.

[124] F. A. Ware and C. Hampel. Micro-threaded row and column operations in a DRAM core. In *Proc. the first Workshop on Unique Chips and Systems (UCAS)*, Mar. 2005.

[125] F. A. Ware and C. Hampel. Improving power and data efficiency with threaded memory modules. In *Proc. the Int'l Conf. Computer Design (ICCD)*, 2006.

[126] P. M. Wells, K. Chakraborty, and G. S. Sohi. Mixed-mode multicore reliability. In *Proc. the 14th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2009.

[127] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. Trading off cache capacity for reliability to enable low voltage operation. In *Proc. the 35th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2008.

[128] E. Witchel, J. Cates, and K. Asanovic. Mondrian memory protection. In *Proc. the 10th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2002.

[129] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considera-

tions. In *Proc. the 22nd Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 1995.

[130] J. Wuu, D. Weiss, C. Morganti, and M. Dreesen. The asynchronous 24MB on-chip level-3 cache for a dual-core Itanium®-family processor. In *Proc. the Int'l Solid-State Circuits Conf. (ISSCC)*, Feb. 2005.

[131] D. H. Yoon and M. Erez. Flexible cache error protection using an ECC FIFO. In *Proc. the Int'l Conf. High Performance Computing, Networking, Storage, and Analysis (SC)*, Nov. 2009.

[132] D. H. Yoon and M. Erez. Memory mapped ECC: Low-cost error protection for last level caches. In *Proc. the 36th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2009.

[133] D. H. Yoon and M. Erez. Virtualized and flexible ECC for main memory. In *Proc. the 15th Int'l. Conf. Architectural Support for Programming Language and Operating Systems (ASPLOS)*, Mar. 2010.

[134] D. H. Yoon and M. Erez. Virtualized ECC: Flexible reliability in main memory. *IEEE Micro, Special Issue: Micro's Top Picks from 2010 Computer Architecture Conferences (MICRO TOP PICKS)*, 31(1):11–19, Jan./Feb. 2011.

[135] D. H. Yoon, M. K. Jeong, and M. Erez. Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput. In *Proc. the Int'l Symp. Computer Architecture (ISCA)*, June 2011.

[136] L. Zhang, Z. Fang, M. Parker, B. Mathew, L. Schaelicke, J. Carter, W. Hsieh, and S. McKee. The Impulse memory controller. *IEEE Transactions on Computers, Special Issue on Advances in High Performance Memory Systems*, 50(11):1117–1132, Nov. 2001.

[137] W. Zhang. Replication cache: A small fully associative cache to improve data cache reliability. *IEEE Trans. Computer*, 54(12):1547 –1555, Dec. 2005.

[138] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam. ICR: In-cache replication for enhancing data cache reliability. In *Proc. the Int'l Conf. Dependable Systems and Networks (DSN)*, Jun. 2003.

[139] Z. Zhang, Z. Zhu, and X. Zhang. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *Proc. the 33rd IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Dec. 2000.

[140] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In *Proc. the 41st IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Nov. 2008.

[141] H. Zheng, J. Lin, Z. Zhang, and Z. Zhu. Decoupled DIMM: Building high-bandwidth memory systems using low-speed DRAM devices. In *Proc. the 36th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2009.

[142] J. F. Ziegler and W. A. Lanford. Effects of cosmic rays on computer memories. *Science*, 206:776–788, Nov. 1979.

Vita

Doe Hyun Yoon was born in Seoul, Korea on 21 June 1975. Doe Hyun studied at Yonsei University, Seoul, Korea, where he received a B.S. degree in Electrical Engineering in 1998 and an M.S. degree in Electrical and Computer Engineering in 2000. Doe Hyun begun his work at LG Electronics, Seoul, Korea in 2000. He has worked on audio/video codec development, real-time OS programming, and scalable video codec (SVC) standardization at LG for five years. Doe Hyun continued his study at Stanford University, California, where he received an M.S. degree in Electrical Engineering in 2007.

Doe Hyun started his Ph.D. study on computer architecture at the University of Texas at Austin in 2007. His research is focused on reliability issues in memory systems, including caches, DRAM, and emerging non-volatile memory. His research has been published in major computer architecture conferences (ISCA, SC, ASPLOS, and HPCA). One paper is selected as Top Picks by the *IEEE Micro* magazine. Doe Hyun worked as a research intern at MIPS technology in 2006 and HP Labs in 2010.

Permanent address: Dong-Bu Centreville Apt. 101-801, Songpa-gu,
Garak-dong, Seoul, Korea 138-160

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.