Frugal ECC: Efficient and Versatile Memory Error Protection through Fine-Grained Compression

Jungrae Kim   Michael Sullivan   Seong-Lyong Gong   Mattan Erez
{dale40, mbsullivan, sl.gong, mattan.erez}@utexas.edu
Department of Electrical and Computer Engineering
The University of Texas at Austin

ABSTRACT

Because main memory is vulnerable to errors and failures, large-scale systems and critical servers utilize error checking and correcting (ECC) mechanisms to meet their reliability requirements. We propose a novel mechanism, *Frugal ECC (FECC)*, that combines ECC with fine-grained compression to provide versatile protection that can be both stronger and lower overhead than current schemes, without sacrificing performance. FECC compresses main memory at cache-block granularity, using any left over space to store ECC information. Compressed data and its ECC information are then frequently read with a single access even without redundant memory chips; insufficiently compressed blocks require additional storage and accesses. As examples, we present chipkill-correct ECCs on a non-ECC DIMM with ×4 chips and the first true chipkill-correct ECC for ×8 devices using an ECC DIMM. FECC relies on a new Coverage-oriented-Compression that we developed specifically for the modest compression needs of ECC and for floating-point data.

Categories and Subject Descriptors

B.7.3 [**Reliability and Testing**]: Error-checking; B.3.2 [**Memory Structures**]: Primary Memory

Keywords

memory, DRAM, reliability, ECC, compression

1. INTRODUCTION

Large-scale and mission-critical compute systems typically rely on *error checking and correcting (ECC)* memory to achieve their reliability goals. Without strong ECC, DRAM faults lead to frequent system-level errors and failures, reducing availability and potentially corrupting computation [1, 2, 3, 4, 5]. ECC memory trades off redundant storage, bandwidth, and energy for increased reliability. ECC memories typically employ *ECC DIMM (Dual In-line Memory Modules)* that have 12.5% more DRAM chips than non-ECC DIMMs; hence ECC typically adds a 12.5% capacity, bandwidth, and energy overheads. In this paper we present a novel approach to managing ECC storage that offers new tradeoffs between reliability, performance, and efficiency. *Frugal ECC* combines a new compression approach with carefully selected ECC codes to provide benefits that exceed those of other recently published techniques that aim to reduce the overheads of ECC protection [6, 7, 8, 9, 10].

While 12.5% redundancy has become a de facto standard, the type of error and fault modes that are common to current DRAM parts pose significant challenges for efficient and effective ECC memory design. For example, several recent works demonstrate that to achieve required levels of reliability and availability, a strong form of ECC known as *chipkill-correct* or single-device data correction (SDDC),[1] is highly desirable [1, 2, 3, 4, 5, 15]. However, truly achieving this level of reliability necessitates compromises in memory system design, such as using wide 128-bit channels or using narrow x4 DRAM chips. With a large fraction of system power consumed by the memory system [16], these compromises can translate into significant performance and energy inefficiencies.

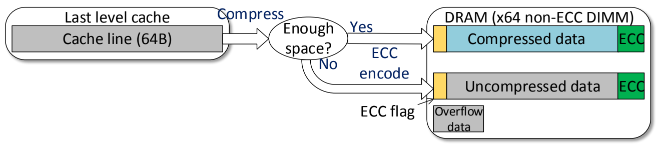
Recent ECC approaches attempt to improve reliability while staying within the bound of acceptable system parameters (64-bit memory data channels and near-12.5% redundant storage and bandwidth) by introducing new reliability and performance tradeoffs [6, 8, 7, 9, 10, 11]. One example is the use of ECC codes that do not precisely match the definition of chipkill-correct but still correct the vast majority of single chip errors [9, 10]; we refer to this class of codes as *chipkill-level* protection. Another example is the use of multi-tier codes in which a first ECC code is used for detection and a second code is used for correction [7, 9, 10]. We discuss the reliability and performance implications of these organizations and investigate other key tradeoffs in detail in Section 2.

We present *Frugal ECC* (FECC), an adaptive and strong ECC technique that relies on opportunistic compression to offer an entirely new set of tradeoffs between reliability and ECC overheads. FECC can even eliminate the need for specialized components and redundant DRAM devices without sacrificing reliability or performance efficiency. The insight behind FECC follows recent works which observe that compression at the cache-block granularity can free up enough space for other information [17, 18]; we use this free space for storing the ECC codes. Thus, when compression succeeds, FECC can match the performance of a conventional ECC organization that uses dedicated ECC memory devices with less, or even zero, dedicated redundancy. While the concept is simple, we introduce two innovations that are crucial to make FECC truly effective.

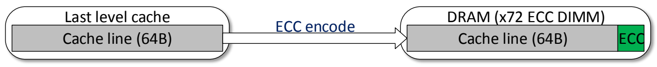
First, we develop a new compression scheme, *Coverage-oriented Compression (CoC)* that maximizes the fraction of blocks that compress

SC '15, November 15-20, 2015, Austin, TX, USA
© 2015 ACM. ISBN 978-1-4503-3723-6/15/11...$15.00
DOI: http://dx.doi.org/10.1145/2807591.2807659

[1]Chipkill-correct protection is referred to as Chipkill, Single Device Data Correction (SDDC), extended ECC, and ChipSpare protection by IBM, Intel, Sun (now Oracle), and HP, respectively [11, 12, 13, 14].



(a) Frugal ECC memory. A block is compressed and freed space is used to store ECC information. If a block compresses poorly, overflow data (from the overhead of compression and for ECC) is stored on a separate block.



(b) Existing ECC memory. ECC redundancy is stored along with data in a fixed amount of dedicated memory devices.

Figure 1: A comparison between FECC and existing ECC memory.

just enough for ECC rather than needlessly aiming for greater levels of compression. Specifically, CoC introduces new compression schemes for floating-point data and for memory blocks with heterogeneous data types (e.g., mixed floating-point and integer values), as well as mechanisms for more effectively representing a mix of compression schemes and tuning compression parameters. Prior memory compression techniques do particularly poorly with floating-point data—a significant problem for HPC workloads that we remedy. The second important innovation is how to protect poorly compressed blocks. Not all memory blocks are sufficiently compressible and some blocks fail to yield enough spare footprint for the redundant information. To address these *compression exceptions*, FECC applies different ECC layouts and protects its layout meta-data separately to *guarantee* up to true chipkill-correct protection (Figure 1).

We show that combining careful layout with CoC simultaneously achieves superior reliability and lower overheads when compared to state-of-the-art single and multi-tiered ECC mechanisms. We perform our evaluation across a large suite of benchmarks, which include floating-point heavy programs from the NAS Parallel Benchmarks and SPLASH2X and SPECFP 2006 suites, and evaluate reliability in the context of large-scale systems. To summarize our contributions and results:

- We describe and evaluate Coverage-oriented Compression for cache blocks and demonstrate that utilizing a new flexible-granularity compression scheme with a unique variable-length encoding results in only a small fraction of compression exceptions across a large set of both integer and floating-point-heavy benchmarks. CoC can sufficiently compress 84% of all accesses in SPEC Int, 93% in SPEC FP, 95% in SPLASH2X, and nearly 100% in the NPB suites.

- We present the mechanisms and layout for storing and protecting compression-exception blocks without degrading reliability; we then show that the new Frugal ECC organizations yield superior reliability to their traditional counterparts across a range of recently-described ECC codes, including Multi-ECC [10] and Bamboo-ECC [19]. In fact, by combining the FECC mechanism with a strong Bamboo-ECC code, we demonstrate the first *true chipkill-correct* memory system with $\times 8$ DRAM devices and a 72b channel.

- We are the first to directly compare the expected reliability of several recent ECC schemes. We also evaluate the performance and energy impact of different ECC schemes and show that FECC can significantly improve system efficiency while only marginally impacting performance and while fully maintaining the reliability of the underlying ECC code.

The paper proceeds as follows. We first review previous ECC and compression work in Section 2. Section 3 provides an overview of how FECC enables a rich set of reliability tradeoffs. Section 4 and Section 5 describe compelling ECC organizations and the novel coverage-oriented compression scheme used in FECC. Section 6 investigates the effectiveness of CoC compression and evaluates the reliability, performance, and energy impacts of FECC. Finally, Section 7 describes exciting future research avenues and Section 8 concludes the paper.

2. BACKGROUND

This section reviews the concepts and terminology that are fundamental to a full description and evaluation of FECC. Brief introductions to ECC and main memory compression are presented below.

2.1 Memory ECC

ECC schemes differ in their error detection and correction capabilities, the amount of redundant storage they require, their access granularity, and where they locate their redundant information. Chipkill-correct ECCs can tolerate a single failing chip, and there are many slightly weaker chipkill-level ECC implementations which can be categorized based on the width of the DRAM chips they can fully protect. The following subsections and Table 1 list some of the most important and efficient chipkill and chipkill-level ECC schemes.

2.1.1 $\times 4$ Chipkill-correct:

An old and well-known $\times 4$ chipkill-correct scheme applies a 4-bit Reed-Solomon (RS) code over a 144-bit data interface that is built from two 72-bit ECC DIMMs [20, 21]. This scheme utilizes 4 check symbols (and 16 bits of redundancy) to correct all single-chip errors and detect all double-chip errors. *Virtualized ECC* (VECC) [7] is a multi-tiered variant that can provide chipkill-correct on a wider variety of memory module organizations. VECC can provide $\times 4$ chipkill on a 128-bit data interface (two non-ECC DIMMS), by virtualizing and storing 3 check symbols elsewhere in the memory space as data. Two separate accesses are required for each access to memory—one for data and the other for redundancy—though caching the ECC information is often effective and helps to reduce the performance impact of the additional memory traffic. Alternatively, VECC can separate the codes used for error detection and correction and can use a 136-bit data interface (built from one 72-bit ECC DIMM and one 64-bit non-ECC DIMM) to provide chipkill-correct without needing two accesses in the common case. In this organization, VECC virtualizes a one-symbol error correcting code and stores it elsewhere in the memory space as data, but this secondary code needs only be accessed in the rare case when errors are detected (or upon a memory write, as it must be updated). Other approaches exist, similar to VECC, that embed ECC data elsewhere in the memory space to allow a more flexible use of non-ECC memory modules [6, 8]. This paper focuses on VECC as a representative for these approaches due to its clear and open description.

To prevent performance degradation from larger access granularities, the most aggressive and the state-of-the-art schemes can correct a single $\times 4$ chip error [11] or a single-chip-error/two-pin-errors [19] on a 72-bit ECC DIMM. These approaches rely on 8-bit Reed-Solomon codes, reducing the redundancy for a single $\times 4$ chip correction down to 8 bits. They have slightly degraded detection coverage for double chip errors, as they can only detect 98.9999996% (but not 100%) of such errors.

×4 Chipkill ECCs

ECC	Channel width	Protection	Overheads		
			Chip	Storage	Access
S4SC-D4SD	144-bit	1 chip correct - 2 chip detect	12.50%	12.50%	None
V-ECC	136-bit	1 chip correct - 2 chip detect	6.25%	9.38%	W
V-ECC	128-bit	1 chip correct - 2 chip detect	0.00%	9.38%	R/W
AMD	72-bit	1 chip correct	12.50%	12.50%	None
QPC	72-bit	1 chip correct	12.50%	12.50%	None
FECC+Multi[1]	64-bit	Almost 1 chip correct	0.00%	7.25%	R/W (Exception)
FECC+QPC	64-bit	1 chip correct	0.00%	13.50%	R/W (Exception)

×8 Chipkill ECCs

ECC	Channel width	Protection	Overheads		
			Chip	Storage	Access
OPC	144-bit	1 chip correct	12.50%	12.50%	None
V-ECC	144-bit	1 chip correct - 2 chip detect	12.50%	18.75%	W
V-ECC	128-bit	1 chip correct - 2 chip detect	0.00%	18.75%	R/W
LOT-ECC	72-bit	Most 1 chip correct	12.50%	26.50%	None
Multi-ECC	72-bit	Almost 1 chip correct	12.50%	12.90%	W
FECC+OPC	72-bit	1 chip correct	12.50%	26.00%	R/W (Exception)
FECC+Multi	64-bit	Almost 1 chip correct	0%	13.50%	R/W (Exception)

Table 1: A comparison between different chipkill-correct and chipkill-level ECC schemes for ×4 and ×8 DRAM chips. Schemes differ in their error detection and correction capabilities, their amount of redundant storage and chips, their access granularities, and the amount of additional accesses they require. 1: Multi-ECC [22] is modified for ×4 chips by building a 16-bit symbol over 4 beats.

2.1.2 ×8 Chipkill-correct:

Wider ×8 DRAM chips save energy by activating fewer chips per memory access for a given data channel width. However, as a chip failure compromises a larger number of bits, ×8 chipkill is generally more challenging and requires more redundancy than its ×4 counterpart. To correct a ×8 chip error, the minimum redundancy (given by the *Singleton bound*) is twice the correction size, giving a minimum of 25% redundancy for a 64-bit interface and 12.5% for a 128-bit interface. Octuple Pin Correcting [19] ECC reaches this lower bound and can correct single ×8 chip error, two concurrent ×4 chip errors, or four concurrent pin errors using 16 pins of redundancy on 128-bit interface.

There are additional recent academic approaches that target ×8 chipkill-level protection on a narrow channel. LOT-ECC [9] can correct most (127/128 or 99.2%) single chip errors on a 72-bit ECC DIMM by using a 4-tiered ECC scheme. Due to its use of a 7-bit error detecting checksum, however, 1/128 of chip errors remain undetected by LOT-ECC and lead to silent data corruption. Multi-ECC [22] uses an error localization and erasure code to enable chipkill-level protection on a 72-bit ECC DIMM with 12.9% redundancy. The error localization procedure of Multi-ECC is based on a 16-bit checksum, and it fails to identify the faulty symbol with 2^{-16} probability, making 2^{-16} of single chip errors detectable-yet-uncorrectable.

2.2 Memory Compression

Memory compression has been actively researched and deployed in caches and main memory to increase the available memory capacity and reduce off-chip traffic. We introduce some pertinent notation and then describe the most relevant prior work below.

In data compression, the *compression ratio* is defined as the ratio between the size of the uncompressed and compressed data (higher is better). High compression ratios are beneficial for stream or link-based compression schemes. Block-based compression schemes in main memory, however, have a fixed block size (e.g. a cache line) and over-compression results in unused memory. Instead, a better criterion for block-based main memory compression schemes is *compression coverage* [23]. Compression coverage is defined as the percentage of blocks that are compressed to a given threshold. A *compression exception* [23] occurs when a block is under-compressed such that its data cannot fit in the target footprint. Coverage is related to the rate and number of compression exceptions; compression exceptions should be minimized as they incur additional overheads.

2.2.1 Per-word compression:

Some memory compression techniques are able to compress data at a single-word (32 or 64-bit) granularity. Frequent value compression [24] targets a small set of 2–8 frequent values that collectively occupy over 50% of memory entries in some benchmarks. It replaces each frequent value by a small index to reduce space. Frequent pattern compression [25] targets frequent patterns rather than frequent values. Most of their patterns are based on the fact that large data types (e.g. 32 or 64-bit fixed-point numbers) often contain small values that do not fully utilize their allocated storage. Frequent pattern compression saves storage by opportunistically converting such values to use smaller data types.

2.2.2 Per-block compression:

As per-word compression applies a different compression technique to each word, it can spend a significant amount of storage on fine-grained compression meta-data. Base-Delta-Immediate (BDI) [26] compression exploits the small dynamic range of integer and pointer types at a larger memory-block granularity. If a cache line has homogeneous data types, values within the line are likely to have similar values. BDI stores the first word as a base and compresses all of the other words as differences from this base. For blocks with mixed integer and pointer types, BDI adds another base value (implicitly assumed to be 0) that is used for integers while the original base is employed for pointers. However, as the significand of floating-point data has poor value locality due to normalization, the performance of BDI suffers greatly on floating-point data.

2.2.3 Main Memory compression:

IBM MXT technology [27] is a word-granularity compressor that relies on a derivative of adaptive dictionary based coding [28] to more than double the effective main memory capacity. However, the drawbacks of MXT include a large cache line size (1KB) to amortize its significant dictionary cost, long compression/decompression latencies from sequential processing and additional accesses to locate the compressed data.

Linearly compressed pages (LCP) [23] compresses all cache lines within a page to the same size, making address calculation straightforward and avoiding memory fragmentation. Upon a compression exception, LCP allocates separate regions within the physical page for exception flags and exception data. If the number of exceptions overflows this allocated storage, LCP traps to operating system and requests a bigger physical page for the virtual page.

MemZip [18] compresses cache lines for memory traffic reduction, not for capacity savings. The compressed data are stored in the same footprint as uncompressed data but memory traffic and energy are reduced on a memory channel that supports fine-grained rank subsetting. Space savings can also be used to store meta-data, such as data bus inversion for memory interface energy saving or ECC for opportunistically stronger protection (though the potential for ECC is largely unexplored). Other work also employs memory compression to store optional prefetch hints [17].

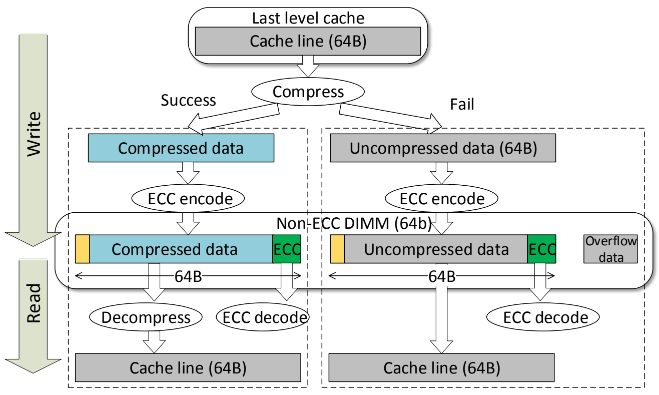


Figure 2: Memory reads and writes with Frugal ECC.

Free ECC [29] combines compression and ECC for the last level cache. Free ECC depends on customized tags to store compression meta-data, however, and it combines BDI and FPC compression. The lack of dedicated meta-data storage makes this approach inappropriate for DRAM; later results (Section 6.2) also show the ineffectiveness of BDI and FPC alone for a wide range of benchmark suites.

COP uses compression to enable ECC protection using non-ECC DIMMs [30]. The specifics of the COP approach make its level of error protection quite weak—significantly worse than SEC-DED—and preclude it from being a viable alternative to FECC for high-performance and high-availability systems. The COP ECC scheme cannot be extended in a straightforward manner to chipkill-correct levels of protection, even in its strongest organization, because the implicit manner in which it tracks the compressibility of a memory block degrades the coverage of its ECC code. The compression scheme used by COP also targets 6.25% redundancy and it provides poor compression coverage for the industry-standard 12.5% ECC footprint. There is also a patent that describes the use of compression with non-ECC DIMMs [31] without any evaluation or consideration of chipkill-level protection.

3. FRUGAL ECC OVERVIEW

Frugal ECC (FECC) is a flexible and efficient ECC solution that offsets ECC overheads through fine-grained compression. In conventional single-tier ECC organizations, redundant information is stored in dedicated memory devices that are accessed together with the devices that store data. In this way, ECC does not impact performance when compared to unprotected memory. FECC attempts to approach the performance of these conventional organizations while at the same time enabling more code design flexibility than just dedicating the full redundancy of a single-tier code.

Figure 2 provides an overview of how FECC works. On a memory write, FECC losslessly compresses a data block to remove data-inherent redundancy and free up space. If FECC successfully creates enough space for the redundant ECC information, the compressed data and ECC check bits are stored in the original memory footprint of the uncompressed data. As its starting address does not change, a memory block can still be randomly accessed and no additional address translation procedure or level of indirection is needed. Furthermore, as with a conventional ECC, both data and all redundant information are accessed in unison. On a memory read, ECC checking and decompression occur in parallel as ECC is computed on the compressed data. If no errors are detected, the decompressed data is forwarded to the last-level cache. Otherwise, ECC attempts to recover any lost data and decompression repeats using the corrected data.

Not all blocks can be compressed to the reduced footprint with lossless compression. Memory blocks that fail to meet the compression threshold for a given ECC scheme result in a *compression exception*. Upon a compression exception, FECC uses one full block and one partial block to store the uncompressed data and ECC check bits. Each compression exception degrades performance, because both the write and any future read require two (potentially cached) accesses to memory instead of the single access used in a conventional ECC design.

As the ECC layouts for compressed and exception data differ, some ECC meta-data is used to identify whether compression was successful and to determine the resulting ECC layout. FECC stores the *ECC flag* that indicates which layout is used along with the compressed data and applies an additional level of error protection to the flag—this higher level of protection is necessary because a compromised ECC flag indicates that a wrong set of bits should be treated as ECC information; using incorrect bits for ECC decoding degrades the protection level. We therefore use a 3 to 5-bit ECC flag in our design, as described further in Section 4.

The key to the success of FECC is a design that minimizes exceptions while maintaining enough redundancy to meet reliability goals. As such, FECC requires both state-of-the-art ECC and compression schemes. In the following sections, we present how FECC can utilize the most aggressive available ECC schemes and we present a novel compression scheme, *Coverage-oriented Compression*, to increase its compression coverage far beyond what is possible with prior compression techniques.

FECC statically reserves enough storage for its overflow data and accesses this overflow storage as-needed using a deterministic addressing scheme. After a read, if the ECC flag identifies an exception, FECC deterministically generates the address of its overflow data and checks the LLC. Upon a miss, FECC fetches the overflow data from DRAM (similar to VECC). The resulting worst-case latency is the LLC miss penalty plus the round-trip delay between the memory controller and LLC, but LLC caching reduces the latency in the common case. Optimizations of storage through dynamic overflow allocation are left for future work (Section 7).

4. ECC IN FRUGAL ECC

FECC can flexibly trade off performance for error protection using different ECC schemes and by changing its target compression threshold. We present several compelling configurations in the subsections below. Before providing the details of ECC, we first briefly discuss the encoding of the ECC flag. The ECC flag in FECC identifies which of several possible ECC layouts is used for a particular block; the specific layout depends on the code design and the level of compression achieved for a given block. In our current designs, we aim to free up either 64 bits or 32 bits of data for ECC; we use the terms *full and half-compression* to denote whether the full 64 bits needed for some ECC codes are freed or whether only half that amount are reclaimed. An error in the ECC flag can lead to undetected errors because data will be interpreted as ECC information or vice versa.

As we describe below, different FECC schemes require a different number of possible layouts. If there are only two layouts (uncompressed / compressed), we use Triple-Modular-Redundancy (TMR) to encode the 1-bit ECC flag; the flag is replicated twice and a majority vote between the 3 copies determines its value. If there are three layouts (uncompressed / half-compressed / fully-compressed), we need to encode 3 different values. To guarantee a low error rate on the flag, we require that the 3 codewords are different in at least 3 bits (with a Hamming distance of 3) and use codewords of 4 or 5 bits. In all cases, the bits used to store the ECC flag are distributed over different chips so that a chip error corrupts only a single bit of the encoded flag, and is therefore correctable. If two chip errors compromise a flag, a wrong data layout may be selected for decoding. However, even in this case, the codes used by FECC have very high detection coverage so that decoding with incorrect layout is very likely to result in a detected error.

4.1 x4 Chipkill FECC

FECC can employ ×4 chipkill ECCs, which use ×4 DRAM devices, to provide the same level of protection and a similar level of performance without the use of redundant memory devices. We use AMD chipkill [11] and QPC [19] as examples of ×4 ECC. Both codes require 12.5% redundancy (8 bits on a 64-bit channel) and FECC can provide this space by compressing a 512-bit block into 443 bits of data and 5 bits for the ECC flag. This 448b *compression target* determines the rate of compression exceptions, and thus the impact on performance. Therefore, to minimize performance degradation even further, we treat the ECC code as a two-tiered ECC with a 6.25%-redundancy first-tier error code (T1EC) that can detect any single-chip error and a second-tier 6.25%-redundancy code (T2EC) that can correct any such error.[2] If a block compresses to the 448-bit threshold, the T1EC and T2EC check bits are stored along with compressed data in the original uncompressed data footprint. If a block is half-compressed to a size in between 448 and 480 bits (including the 5b ECC flag), the T1EC is stored along with compressed data but the T2EC is stored in a separate block. If a block fails to compress to even a 480-bit footprint, part of the uncompressed data is stored along with the T1EC and T2EC in a separate memory block. These organizations are depicted in Figure 3. Note that we use a 4 or 5-bit encoding for the ECC flag to ensure true chipkill-correct reliability—even if a chip with ECC flag has errors, the ECC flag will be correctly decoded.

Multi-ECC was published as chipkill-level protection for ×8 DRAM devices through erasure coding. It uses one redundant symbol to detect any single ×8 chip error and employs a separate checksum (which is shared among multiple lines to amortize storage overheads) to locate the faulty chip. Once the error location is identified, it uses erasure coding with the previous one-symbol redundancy to correct the error. For ×4 chipkill, we modify Multi-ECC so that it builds a 16-bit RS symbol from 4 data transfer beats, reducing its main ECC redundancy down to 6.25%; we maintain the same checksum mechanism as in the original Multi-ECC design. FECC with ×4 Multi-ECC can access any block that is compressed to a 480-bit footprint without accessing secondary storage. In the case of a compression exception, the data and its ECC information are split over two blocks. This is depicted in Figure 4. We use a 3-bit encoding of the ECC flag for Multi-ECC because it has only two ECC layouts and triplicating the 1-bit flag can correct a single-bit error.

4.2 ×8 Chipkill FECC

The amount of redundancy required for the ×8 true chipkill-correct OPC [19] is 25% (16 bits on 64-bit channel). As the coverage-oriented compression of FECC primarily focuses on 12.5% redundancy (see Section 5.2), FECC uses ECC-DIMMs with 12.5% redundant devices to split the 25% ECC overhead between the redundant chips and the space made available by compression. FECC with OPC uses a two-tiered ECC so that if a block is compressed to less than 448 bits (443 bits of data and the 5-bit ECC flag), the compressed data and the 128 bits of redundant information (T1EC + T2EC) is stored within a single block of an ECC DIMM. If a 512-bit block is compressable to 512 bits (some blocks fail due to compression and ECC flag overheads), the ECC flag, compressed data, and 64-bit T1EC are stored on the ECC DIMM. In this common case, a write accesses an additional memory location for the T2EC but error-free reads require only a single memory access. If compression fails entirely, part of the uncompressed data, the T1EC, and T2EC are stored across the ECC chip and also in a separate memory block. The ×8 chipkill-level Multi-ECC requires 12.5% redundancy, so FECC can use a 64-bit non-ECC DIMM with a 448-bit compression target (445 bits of data and the 3-bit ECC flag); as with the ×4 Multi-ECC configuration, the checksum information is maintained separately as per the original Multi-ECC design.

[2]The T2EC code must be combined with the T1EC to correct a single-chip error; it is not independently capable of this level of correction.

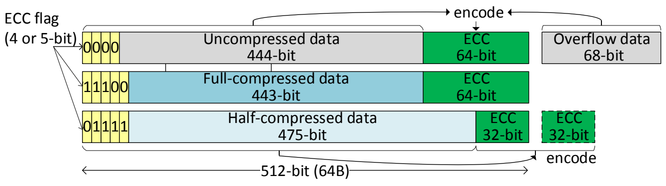


Figure 3: Frugal ECC layout for chipkill-correct (64-bit redundancy).

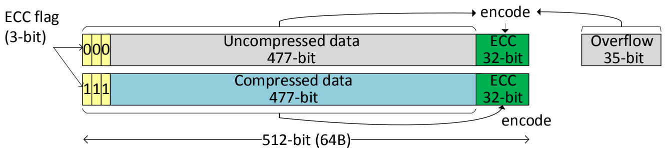


Figure 4: Frugal ECC layout for ×4 Multi-ECC (32-bit redundancy).

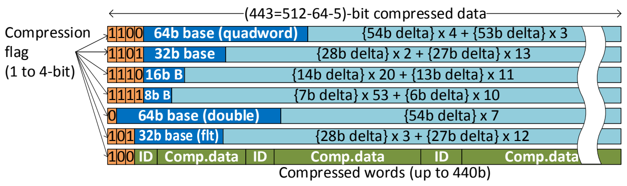

(443=512-64-5)-bit compressed data
Compression
flag
(1 to 4-bit)
1100 64b base (quadword) {54b delta} x 4 + {53b delta} x 3
1101 32b base {28b delta} x 2 + {27b delta} x 13
1110 16b B {14b delta} x 20 + {13b delta} x 11
1111 8b B {7b delta} x 53 + {6b delta} x 10
0 64b base (double) {54b delta} x 7
101 32b base (flt) {28b delta} x 3 + {27b delta} x 12
100 ID Comp.data ID Comp.data ID Comp.data
Compressed words (up to 440b)

Figure 5: Data layout for the 448-bit target (full compression).

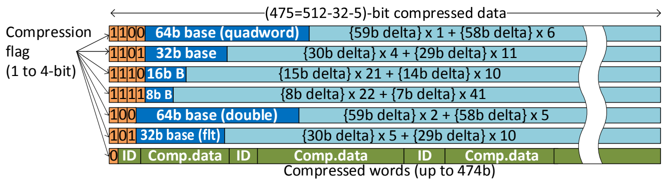

(475=512-32-5)-bit compressed data
Compression
flag
(1 to 4-bit)
1100 64b base (quadword) {59b delta} x 1 + {58b delta} x 6
1101 32b base {30b delta} x 4 + {29b delta} x 11
1110 16b B {15b delta} x 21 + {14b delta} x 10
1111 8b B {8b delta} x 22 + {7b delta} x 41
100 64b base (double) {59b delta} x 2 + {58b delta} x 5
101 32b base (flt) {30b delta} x 5 + {29b delta} x 10
0 ID Comp.data ID Comp.data ID Comp.data
Compressed words (up to 474b)

Figure 6: Data layout for the 480-bit target (half compression).

5. COMPRESSION IN FRUGAL ECC

While most existing compression schemes focus on improving the compression ratio, compressing data to as small a size as possible, leftover space from an over-compression is left unused in FECC. In the mean time, compression exceptions due to under-compression cost FECC additional memory accesses for overflow data, consuming additional latency, bandwidth, and energy. Therefore, we develop a novel compression scheme, *Coverage-oriented Compression (CoC)*, which focuses on maximizing coverage for the modest compression ratio goals of FECC. The fundamental observation behind CoC is that we can trade off compression ratio for compression coverage. A conventional compression-ratio-oriented compression assigns short codes to the most frequent values to reduce the average number of bits needed to store a value. A key drawback of this assignment is that the number of available "good" codes runs out quickly. Our coverage-oriented compression, on the other hand, tries to cover as many values as possible with "acceptable" codes to maximize coverage. For example, an ideal coverage-oriented compression would assign m-bit codes to the most frequent 2^m values, so that all of these values can be compressed into an m-bit footprint.

CoC has three main components that together maximize coverage with acceptable implementation overheads: 1) fitting base + delta compression for homogeneously-typed data, 2) exponent compression for floating-point data, 3) frequent word pattern compression for heterogeneously-typed data. These three components are described in detail below.

5.1 Fitting Base + Delta compression

Base + delta [26] exploits the small dynamic range typical of homogeneous integer data (e.g. arrays of integers) by storing the small differences between same-typed values. It also supports a mixture of integers and pointers by using a Base-Delta-Immediate organization, as described in Section 2.2.2. We increase delta sizes so that the overall compressed size fits within the modest thresholds needed for FECC. The top 4 organizations in Figure 5 and Figure 6 show the layouts we use in CoC to represent integer data when targeting 448-bit and 480-bit layouts, respectively, with an ECC flag of three values. Each organization includes the *compression flag* to indicate the type of compression used (unlike the ECC flag, the compression flag does not require redundant encoding because it is already protected by ECC), a base value, and a set of deltas. The 448-bit footprint is used to hold a 4-bit compression flag and a 64-bit base value. The remaining 375 bits are used to store deltas for the other 7 64-bit values in the block, which allows 54-bit deltas for 4 of the 7 64-bit values and 53-bit deltas for 3 of the 7 64-bit values. By fully utilizing every bit of the available footprint, *Fitting Base+Delta (FBD)* supports large delta sizes and compresses more cache lines without exceptions. For 64-bit data, the delta sizes are large enough to compress a mixture of 64-bit integers and 64-bit pointers without the need for a separate Base-Delta-Immediate organization for pointers. We follow similar reasoning for maximizing delta sizes to fit within the compression footprint for 32-bit, 16-bit, and 8-bit integers, as shown in Figures 5 and 6.

5.2 Floating-point compression

Low-latency floating-point (FP) compression is difficult, as the normalized floating-point significand can cause small value changes to manifest as a very different binary representation. As the target compression ratio in FECC is low (8:7 compression for 12.5% redundancy), FP compression in CoC only targets the exponent and sign of floating-point data. These fields often have higher value locality and comprise a larger number of bits than our compression goals (18.8% and 28.1% of double and single-precision numbers, respectively). Similar to the use of difference coding for integer values, we can compress the sign and exponent with simple subtraction assuming that their values exhibit locality. The bottom organizations in Figure 5 and Figure 6 show how we compress homogeneously-typed floating-point data using FBD compression.

The 448-bit footprint is used to hold a 1-bit compression flag and a 64-bit base value with the remaining 378 bits evenly split to represent the 7 remaining FP values. Each 54-bit value represents the 1-bit original sign, 52-bit original mantissa, and a 1-bit exponent delta from the exponent of the base. While a 1-bit delta can cover only +0 and -1 changes in the exponent, our experiments show good coverage with this scheme across a range of applications (see Section 6.2). For single-precision floating-point numbers, we increase the delta size to 3 or 4 bits, because our experiments showed a need for a larger range of exponent differences. For the 480b compression target (32-bit redundancy for ECC), the compressed size for double-precision FP increases to 58 or 59 bits and we use the additional bits to represent 6 or 7-bit exponent deltas.

Description	Size (bits)	ID
All zero 64-bit	0	001
Same as left 64-bit	0	1000
Same as left-left 64-bit	0	111110
Same as left-left-left-left 64-bit	0	11110
32-bit data, 32-bit zero	32	11010
2 x {16-bit sign-extension to 32-bit}	32	11011
32-bit sign-extension	32	1010
48-bit sign-extension	48	11100
44-bit delta from left 64-bit	44	11101
52-bit delta from left 64-bit	52	1001
12-bit delta from left-left 64-bit	12	1011
sign, exponent([63:52]) as 4-bit delta from bias (1023)	56	010
sign, exponent([63:52]) as 8-bit delta from bias (1023)	60	011
sign, exponent([63:52]) as 4-bit delta from left 64-bit	56	1100
Exponent([62:52]) as 3-bit delta from bias (1023)	56	111111
Incompressible 64-bit	64	000

Table 2: Per-word compression patterns, selected and variable-length encoded based on SPEC CPU2006 benchmark profiling results with the "test" inputs. The compression coverage of these selected patterns appears to be robust across several benchmark suites and input sets (see Section 6.2).

5.3 Frequent word compression

While FBD provides high compression coverage for homogeneously-type data, it suffers from poor coverage with heterogeneously-typed data (e.g., structs and classes). In particular, FBD works poorly with heterogeneously-typed data with mixed fixed and floating-point numbers, as no single base is appropriate.

To augment this weakness, CoC employs a secondary compression scheme that compresses at the 64-bit (QWORD) granularity. Each QWORD is compressed as either a frequent pattern [25] or as a difference from a previous QWORD within its block. The per-word compressor suite in CoC is selected from a large pool of compressors with different bases, bit positions, and delta sizes. As per-word compression is designed to complement FBD, we run the compressors on blocks incompressible with 443b FBD from the SPECCPU 2006 benchmark suite using "test" inputs. We pick the top 16 compressors based on their estimated coverage and assign their IDs using Huffman coding to reduce the necessary amount of meta-data [32]. In Section 6.2 we show that our choice of compressors and encoding is robust across other benchmark suites and inputs. Table 2 shows the complete list of the selected compressors.

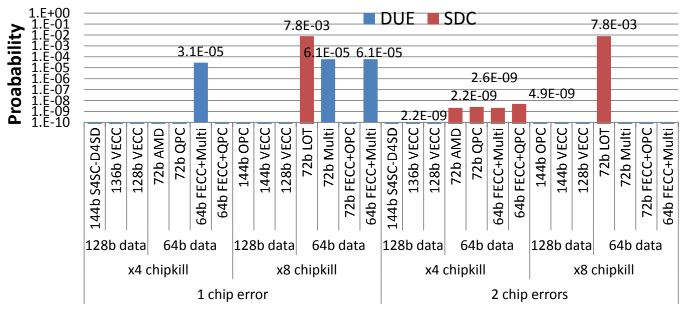
6. EVALUATION

To understand the benefits of FECC and the tradeoffs involved, we evaluate its reliability (in terms of both undetected and detected uncorrectable error rates), performance impact, and energy efficiency in comparison to competing ECC schemes (including current commercial approaches as well as academic designs). We then analyze the effectiveness of CoC and demonstrate that it is both an excellent match for the goals of FECC and superior to previously published compression mechanisms in this context. We also analyze an implementation of key components of the CoC hardware. Throughout this evaluation, we rely on the SPEC CPU 2006 [33, 34], PARSEC [35, 36], SPLASH2X[3] [37, 38], and NAS Parallel Benchmarks suites.[4]

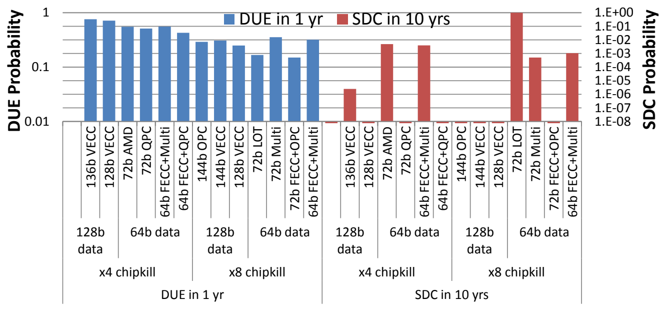
For the reliability, performance, and energy consumption evaluation, we compare 14 ECC configurations. The two commercial

[3]SPLASH2X is an update of the SPLASH2 benchmark suite [37] with larger input sets [38]. It is distributed as part of PARSEC v3.0 [35].

[4]While the majority of benchmarks are evaluated, some programs are omitted due to compilation issues or runtime errors.



(a) Probability of a detected uncorrectable error (DUE) and undetected error (SDC) of a single rank with 1 or 2-chip errors; ECC flag protection is evaluated assuming blocks are fully-compressed/half-compressed/uncompressed with 70/20/10% probabilities, respectively.



(b) Probability of at least one detected uncorrectable error (DUE) and at least one undetected error (SDC) for a system with 100K DIMMs, assuming only 1 or 2-chip errors. DUE rates decrease with FECC, as each rank has fewer redundant chips. We were unable to find the decoding details of S4SC-D4SD and omit it from these results.

Figure 7: Single-rank and full-system reliability with 1 and 2-chip errors across a range of ECC schemes.

configurations are the S4SC-D4SD strong chipkill-correct ECC on a 144b channel (similar to [39]) and the AMD chipkill-correct ECC on a 72b channel, both with ×4 DRAM devices. Bamboo-OPC is a chipkill-correct code for ×8 devices on a 144b channel and Bamboo-QPC is a chipkill-correct ECC similar to, though slightly stronger than, AMD chipkill. Virtualized chipkill is evaluated with 136b and 128b channels for ×4 devices and 144b and 128b channels for ×8 devices and follows the configuration described in Section 2. The LOT-ECC and Multi-ECC chipkill-level schemes are also evaluated. Finally, we evaluate four FECC variants: chipkill-correct FECC+QPC and chipkill-level FECC+Multi-ECC on 64b channels with ×4 devices, chipkill-correct FECC+OPC on a 72b channel with ×8 devices, and chipkill-level FECC+Multi-ECC on a 64b channel with ×8 devices. Note that the FECC configurations have the smallest number of redundant DRAM devices, with the exception of the 128b VECC configurations, though VECC is only defined for wide channels. Also note that the 72b FECC+OPC is, to the best of our knowledge, the only true chipkill-correct ECC that can use both a 72b channel and ×8 DRAM devices.

6.1 Reliability

To evaluate the reliability of chipkill-correct and chipkill-level ECCs, we combine the FaultSim [40] and ErrorSim [19] Monte Carlo fault and error injectors. FaultSim randomly injects faults into a simulated memory channel (using Poisson-distributed faults with empirically observed DRAM fault rates [3]) and ErrorSim generates random errors on a cache-line-size block of memory based on the assumption that all bits within the faulty region flip with a 50% probability. Erroneous blocks are tested using the appropriate Reed-Solomon or one's complement checksum decoders to judge whether each ECC code can correct the error, detect the error but not correct it (DUE), or not detect the error, potentially leading to silent data corruption (SDC). If there are multiple codewords within a cache line (e.g., with AMD chipkill), we assume that an error is detected if any of the codewords reports a detectable error.

Figure 7 shows the reliability evaluation results. LOT-ECC and Multi-ECC are not true chipkill-correct techniques and exhibit errors even with only single-chip errors (see Section 2.1). LOT-ECC uses a 7-bit checksum at a chip granularity and thus exhibits a high SDC probability, which is particularly problematic for large systems (Figure 7b). Multi-ECC with ×8 devices has a 2^{-16} probability of uncorrectable errors per codeword; with 2 codewords per 512b block, the probability of a DUE is 2^{-15}. Despite this problematic protection for single-device errors, the overall reliability of Multi-ECC is very close to that of the other good chipkill-level ECCs, such as AMD chipkill and QPC because the fault model does lead to some errors that span two or more chips. Importantly, FECC shows almost identical protection as the underlying ECC code it utilizes (i.e., QPC, OPC, or Multi×8), despite using fewer devices. In fact, the best overall protection in terms of both the DUE and SDC rates is provided by FECC+OPC, as it relies on fewer devices and therefore exhibits fewer faults, assuming only the faults reported by Sridharan and Liberty [3] can occur.

In contrast to the HPC-oriented high protection level of FECC, COP falls far short. The implicit tracking of compressibility used by COP converts chip, pin, and rank errors in incompressible blocks into severe silent data corruption, because such errors will cause an erroneous decompression.[5] If placed on Figure 7, the COP approach would be orders of magnitude worse than the other schemes and both DUE and SDC probabilities would be 100% in teh evaluated scale.

6.2 Compression Coverage

To test the compressibility of benchmark memory traffic and the effectiveness of coverage-oriented compression, we use a Pin-based [41] cache model that implements Frugal compression at the main memory interface. Serial program versions are used, with a weighted average over all program invocations taken in the case of fork-based multi-programmed benchmarks. Every main memory read and write-back is compressed[6] and its size tallied.

Figure 8 shows the overall compression coverage results. Because of the data dependence of CoC, we conduct compression coverage experiments over a wide variety of benchmarks and input sets. The harmonic mean behavior for each benchmark suite is given by a corresponding HM column. It can be seen that CoC compresses most benchmarks well, with many benchmarks resulting in >99% compression coverage. In general, integer benchmarks compress sufficiently down to the 448b compression target, while many floating-point benchmarks must resort to the 480b level of compression. This is not unexpected, since the compressibility of floating-point numbers is limited (see Section 5.2 for more details).

[5]The COP evaluation does not consider anything except single-bit errors, so it does not capture this effect; however, it is reflected in our more-complete reliability evaluation.

[6]Stratified sampling [42] is used to reduce the experimental runtime of SPEC with the "ref" input set and PARSEC/SPLASH2X with the "native" input sets. Fast-forward, detailed warming, and execution periods of 16M, 2M, and 1M instructions are used, respectively.

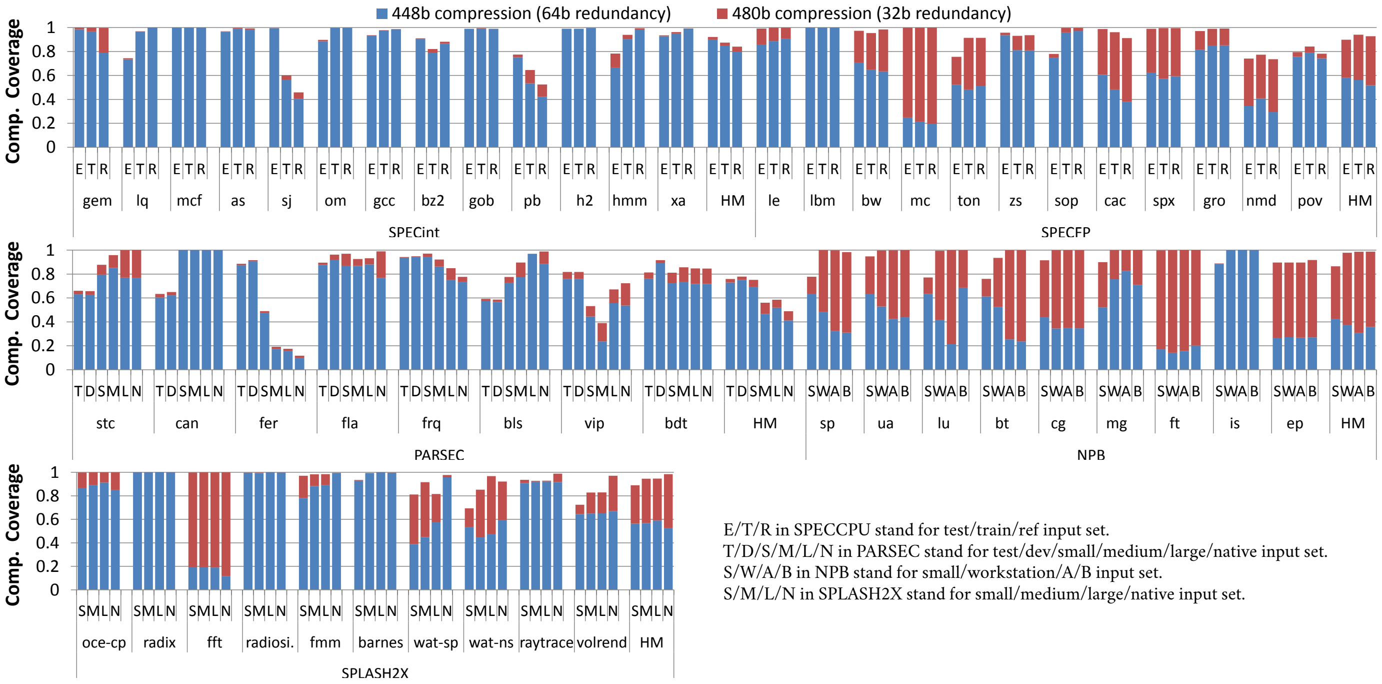


Figure 8: Compression coverage for 64-bit and 32-bit redundancy for the SPECCPU2006, PARSEC, NPB, and SPLASH2X benchmark suites. Benchmarks are sorted within each suite by descending memory traffic.

The proper compression target varies according to the target ECC scheme (see Section 4 for more details). On a 64b rank with ×4 DRAM chips, FECC+Multi uses a 480b target coverage for both reads and writes and FECC+QPC uses a 448b target fully-compressed blocks and 480b for half-compressed blocks. On a 72b rank with ×8 DRAM chips, FECC+Multi uses a 448b target and FECC+OPC uses 448b and 512b targets full-compressed and half-compressed blocks, respectively. (Note that Figure 8 gives results only on a 64b rank; compression coverage trends over a 72b rank are similar but they are not shown in the interest of brevity.)

While most benchmarks and input sets compress satisfactorily with CoC, a few benchmarks show more lackluster compressibility. Ferret performs a content-based similarity search of images using JPEG images as query inputs. The inherent compressibility of these images is poor, perhaps leading to the low CoC coverage. Similarly, VIPS is an image processing system. Within PARSEC, the input images to VIPS are formatted in a native file format [43] and they happen to encode pixels in a non-IEEE754 floating-point format (using the "labq" encoding). It is expected that this mismatch of floating-point formats leads to the lackluster CoC coverage, but more investigation is needed. Sjeng is an artificial intelligence program for chess. It makes heavy use of heterogeneous mixed-size data structures and it is expected that the data structures are not aligned or coalesced properly for CoC to fully capture the value locality within them. Word-level compression does adapt somewhat to the heterogeneous nature of Sjeng—without it the compression coverage is dismal instead of lackluster, and enabling word-level compression increases the coverage by 5.54× and 6.25× at 448b and 480b, respectively. Perlbench is heavily multiprogrammed through repeated forks; some of these program invocations are highly compressible and others compress poorly. More investigation is needed to find out the source of the poorly compressible benchmark segments.

The average compressibility of programs in Figure 8 is high; this high compression coverage is a direct result of our coverage-oriented compression scheme and it would not have been possible using prior fine-grained compression approaches. Figure 9 demonstrates this through compression coverage experiments for PARSEC and NPB using BDI and FPC compression (see Section 2.2 for more details) as well as fitting base + delta compression alone, which represents CoC without word-granularity compression. The results show that CoC always performs as well or better than the best of the other approaches, in some cases (like LU) outperforming all others. It is also readily apparent how insufficient BDI and FPC compression are for floating-point data—the floating-point compressors used for Frugal ECC are necessary for HPC benchmark suites such as NPB. Correspondingly, the compression approach used by Free ECC (combining BDI and FPC) [29] would not suffice for a large range of benchmark programs.

Processor	2GHz OoO core
L1 cache	16kB Inst/64kB Data, 2-way, 2-cycle hit latency, 2 MSHR entries
L2 cache	2MB, 64 or 128B line, 8-way (7-way for Multi-ECC), 20-cycle hit latency, 20 MSHR entries (40 for V-ECC, Multi-ECC, and FECC), stride prefetcher
DRAM	DDR3-1600 / 2 rank / 2 channel for 64b data and 1 channel for 128b data
Simulation	400M cycles after 800M cycles of fast-forwarding
Energy	Micron 2Gb DDR3-1600 parameters + Micron model [44]
Overheads	+1 RD / +1 WRQ latency (S4SC-D4SD, VECC, AMD, and Multi-ECC) +4 RD / +4 WRQ latency (QPC, OPC, and LOT-ECC) +2 RD / +3 WRQ latency (FECC+Multi) +4 RD / +5 WRQ latency (FECC+QPC and FECC+OPC)

Table 3: Simulation parameters for the performance and energy evaluation.

6.3 Performance and DRAM Energy

We measure the performance impact of different ECC schemes using the Gem5 simulator [45] and estimate DRAM energy consumption using the Micron DDR3 power model [44]. Because of the long

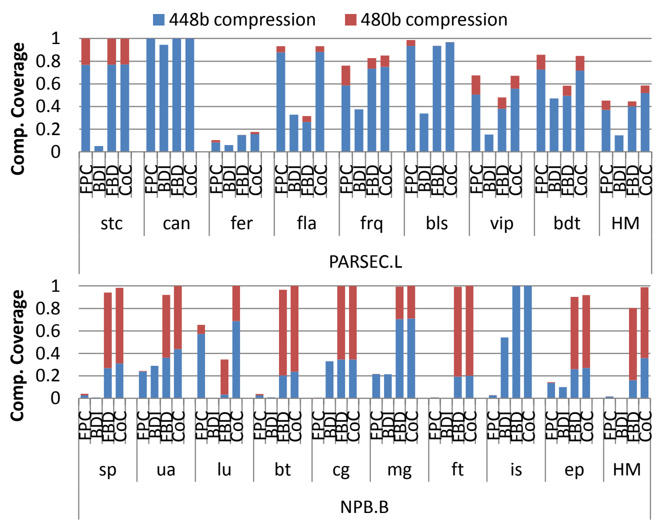


Figure 9: A compression coverage comparison between CoC and prior memory compression schemes.

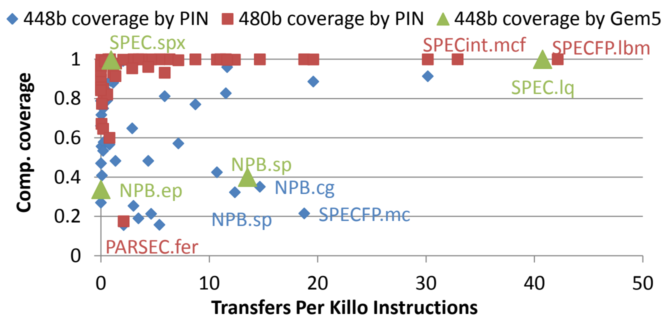


Figure 10: The memory traffic and compression coverage results of all benchmarks measured with Pin and the four representative detailed simulations selected.

runtime of the simulator, we do not evaluate the performance of all the benchmarks for which we obtained compression results. Instead, we focus on four benchmarks that represent different points in the space of parameter values that most-strongly impact the performance and energy behavior of FECC; each of the chosen benchmarks is fast-forwarded 400M instructions and is then simulated for 200M instructions.

The important parameters affecting the performance and energy of FECC are the rate at which off-chip data requests are made and the frequency of compression exceptions. Figure 10 depicts how the different benchmarks whose coverage we evaluate with Pin map across these two dimensions; we use last-level cache misses per thousand instructions (MPKI) to represent the memory-intensity of each application and its compression coverage to represent the expected compression exception rate. We choose Sphinx from SPEC 2006 (denoted SPEC.spx) as representative of benchmarks with low traffic and high compression coverage, which are ideal for FECC. Libquantum from SPEC 2006 (SPECFP.lq) and EP from NPB (NPB.ep) are representative of benchmarks with high coverage / high traffic and low coverage / low traffic, respectively; both of which we expect to also perform well with FECC. As an example for an application that stresses FECC, we evaluate SP from NPB (NPB.sp), which has poor compression coverage and also significant traffic. Note that these cases do not precisely match the extreme points observed with the Pin-based evaluation, but they are as close as we were able to find in detailed simulation.

We perform detailed single-core simulations with the system parameters summarized in Table 3. We assume a 2 memory-cycle latency for CoC compression/decompression and 1 cycle latency for ECC encoding and decoding after the necessary data are fully available (see Section 6.4). While multi-core simulations are more realistic and put more pressure on memory bandwidth, we choose to present single-core simulations because FECC degrades performance mostly due to the extra latency it incurs and because single-core results allow for a clearer presentation of behavior and insights. The baseline IPC (instructions per cycle on the single core) and memory-only energy for our four benchmarks, measured for the AMD chipkill configuration are: 0.82 IPC / 0.23J for Libquantum, 1.33 IPC / 0.12J for Sphinx3, 1.24 IPC / 0.13J for NPB.ep, and 1.71 IPC / 0.10J for NPB.sp.

Figure 11 shows the estimated execution time and DRAM energy consumption over the chosen simulation interval of the selected benchmarks. Execution times are shown in blue bars (left y-axis) normalized to the AMD chipkill configuration and the relative energy is shown as a red line (right y-axis). As expected, the FECC configurations exhibit essentially no impact on performance with the ideal SPEC.spx and also with the low-traffic NPB.ep.

The performance of SPECFP.lq is impacted by type of code used and channel width, but as expected, the addition of FECC has a negligible impact (<1%) because of the high compression coverage. The only configuration for which FECC has a measurable impact is for NPB.sp, which has meaningful traffic and poor compression coverage; the 64b FECC+Multi configuration exhibits a 3.7% performance degradation because it requires a 480b level of compression for all reads and writes. FECC+OPC does not show a similar degradation because it relies on a 72b channel and does not add any latency for read operations, regardless of the compression achieved. Energy results are similar, and the FECC variants slightly improve energy efficiency over their non-frugal counterparts for chipkill-level codes. FECC+OPC is the only chipkill-correct code that that can use both a 72b channel and $\times8$ devices, and because of this it requires only about half the energy consumption of ECCs with competing levels of protection.

6.4 CoC Hardware

Frugal compression requires some hardware to compress and decompress each memory transfer. We design a simple implementation of the main parts of the compressor in Verilog and conclude that FECC requires a very small amount of chip area and is readily implementable in the 2-cycle latency assumed by the performance and energy evaluation. Figure 12 shows an overview of the compression process, broken into two stages. The first stage implements the block and word-granularity compressors through subtraction and equality testing. The output of this first-stage logic is the compressed data for each constituent compressor and the validity of each compression (whether it captures all of the data in its allotted space). This compressor and validity data is fed into a second stage that chooses the most appropriate compressor and outputs the final result.

The first-stage compressor logic is expected to consume the majority of the area for the Frugal compressor. A straightforward, behavioral implementation of this first-stage logic easily achieves a 1ns latency. The block and word compressor consume about 4,383 and 5,866 NAND2 gates' worth of chip area, respectively.[7] Combined, the block and word compressor consume about 25% more chip area than

[7]These delay and area estimates are found through standard-cell synthesis using the Synopsys toolchain and the 40nm TSMC standard cell library [46, 47] but are presented in a technology-independent manner.

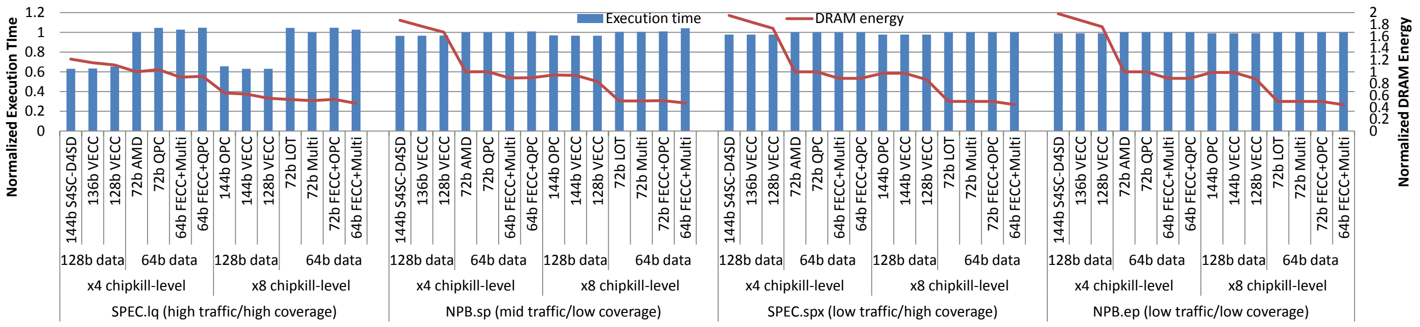


Figure 11: Execution time and DRAM energy consumption normalized to AMD chipkill.

a single 32-bit fixed-point multiplier—an insignificant area compared to an entire processor.

The second compressor stage consists of a priority encoder with a small amount of ID generation logic, and is expected to easily and cheaply fit within the remaining 1ns of compression latency. The Frugal decompressor is expected to consume less area than the compressor, because it does not need to speculatively perform all constituent compression schemes in parallel and can effectively use less internal arithmetic. Also, because decompression uses addition instead of subtraction for all delta operations, logic sharing between the different-width compression schemes should be a straightforward and potentially lucrative optimization.

7. DISCUSSION

Two promising future optimizations for FECC organizations are described below.

Optimizing Performance: Compression Prediction.

FECC performs two sequential memory accesses when reading a compression-exception block whose auxiliary block is not already cached. While such problematic memory reads are not so common as to override the performance benefits of FECC (as evidenced by Section 6.3), it would further improve the performance of FECC to access both the main memory location and the auxiliary overflow data in parallel. As such, a future performance optimization includes a dynamic compression-exception predictor whose purpose is to prefetch the overflow data for incompressible lines before they return from memory.

Optimizing Storage: Dynamic Overflow Mapping.

While FECC focuses on improving performance and energy efficiency by fetching data and ECC information with a single access, it

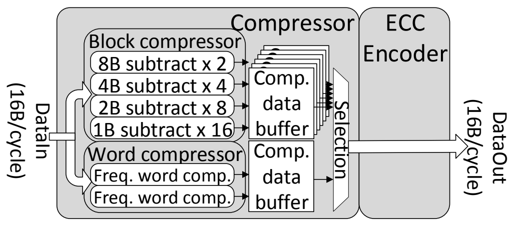


Figure 12: A block diagram of the Frugal compressor. Two combinational logic structures implement the block and word-granularity compressors, respectively. The validity information from each constituent compressor is used to select the most appropriate scheme and to format the output data. As a 64-bit DDR memory provides 128 bits (16B) of data per cycle, the two 8B words are compressed in parallel.

can be extended to also save storage by dynamically allocating auxiliary blocks only when needed. Our current FECC organizations reserve space for overflow data conservatively—redundant space is statically allocated even for memory blocks that compress well. However, as compression exceptions are rare and overflow storage is rarely used, storage overheads can be significantly reduced. One potential problem with such dynamic overflow allocation is how to best reclaim unneeded redundant storage after it is no longer needed (once its corresponding block is rewritten without a compression exception). We expect periodic scrubbing to be a good approach, but have not considered it in this paper. Future work may include the complete evaluation of such dynamic overflow mapping for use in systems where a balance between performance and storage is important. An orthogonal approach for reducing second-tier ECC storage, ECC Parity [48], could be applied in lieu of, or in addition to this optimization and it should be included in a future evaluation. COP ECC describes an organization with pointer-based dynamic exception overflow tracking to reduce ECC storage overheads [30]. The COP scheme may not be applicable to Frugal ECC, however, as this pointer is very costly to protect with chipkill or chipkill-level protection.

8. CONCLUSION

In this paper, we make three main important contributions. First, we describe and evaluate new ECC schemes that provide superior reliability with lower overheads than prior work and with minimal impact on performance. By combining novel compression and metadata encoding and management techniques, our Frugal ECC designs are able to match the memory access characteristics of conventional ECC designs while requiring fewer, or even no, redundant DRAM devices. Second, we show the importance of co-designing a compression scheme with its intended use and introduce a new coverage-oriented compression technique that far exceeds the capabilities of previously-published fine-grained compression mechanisms, especially for floating-point intensive programs. Third, we provide a thorough and fair comparison of the overheads and reliability characteristics of several recently-published chipkill-level ECC schemes and conclude that while their error correction capabilities match those of true chipkill-correct, these chipkill-level designs have a much higher risk of silent-data corruption when evaluated with published fault models [3, 4, 5].

9. ACKNOWLEDGMENTS

The authors acknowledge the Texas Advanced Computing Center for providing HPC resources and the support of the Department of Energy under Award #B609478 and the National Science Foundation under Grant #0954107, which partially funded this research.

10. REFERENCES

[1] B. Schroeder, E. Pinheiro, and W.-D. Weber, “DRAM Errors in the Wild: a Large-Scale Field Study,” in *Proceedings of the International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2009.

[2] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, “Cosmic Rays Don’t Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design,” in *Proceedings of the International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.

[3] V. Sridharan and D. Liberty, “A Study of DRAM Failures in the Field,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2012.

[4] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, “Feng Shui of Supercomputer Memory: Positional Effects in DRAM and SRAM Faults,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, November 2013.

[5] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, “Memory Errors in Modern Systems: The Good, The Bad, and The Ugly,” in *Proceedings of the International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. New York, NY, USA: ACM, 2015, pp. 297–310.

[6] R. Danilak, “Transparent error correction code memory system and method,” U.S. Patent US7 117 421 B1, October, 2006. [Online]. Available: http://www.google.com/patents/US7117421

[7] D. H. Yoon and M. Erez, “Virtualized and Flexible ECC for Main Memory,” in *Proceedings of the International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2010.

[8] M. J. Haertel, R. S. Polzin, A. Kocev, and M. B. Steinman, “ECC implementation in non-ECC components,” U.S. Patent US8 135 935 B2, March, 2012. [Online]. Available: http://www.google.com/patents/US8135935

[9] A. N. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis, and N. P. Jouppi, “LOT-ECC: Localized and Tiered Reliability Mechanisms for Commodity Memory Systems,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2012.

[10] X. Jian, H. Duwe, J. Sartori, V. Sridharan, and R. Kumar, “Low-power, low-storage-overhead chipkill correct via multi-line error correction.” in *SC*. ACM, 2013, p. 24.

[11] Advanced Micro Devices (AMD), Inc., “BIOS and Kernel Developer’s Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors,” Jan 2013.

[12] Intel corp., “Intel Xeon Processor E7 Family: Reliability, Availability, and Serviceability,” 2011.

[13] Oracle, Inc., “Oracle SPARC Server RAS Comparison.” [Online]. Available: http://www.oracle.com/us/products/servers-storage/servers/sparc-enterprise/sparc-ras-comparison-190946.pdf

[14] Hewlett-Packard, “How memory RAS technologies can enhance the uptime of HP ProLiant servers,” 2013.

[15] C. Di Martino, Z. Kalbarczyk, R. Iyer, F. Baccanico, J. Fullop, and W. Kramer, “Lessons learned from the analysis of system failures at petascale: The case of Blue Waters,” in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2014, pp. 610–621.

[16] P. Nikolaou, Y. Sazeides, M. Kleanthous, and L. Ndreu, “The Implications of Different DRAM Protection Techniques on Datacenter TCO,” in *Proceedings of the Workshop on Silicon Errors in Logic–System Effects (SELSE)*, 2015.

[17] G. Liu, J.-K. Peir, and V. Lee, “Miss-correlation folding: Encoding per-block miss correlations in compressed dram for data prefetching,” in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, 2012, pp. 691–702.

[18] A. Shafiee, M. Taassori, R. Balasubramonian, and A. Davis, “MemZip: Exploring unconventional benefits from memory compression,” in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[19] J. Kim, M. Sullivan, and M. Erez, “Bamboo ECC: Strong, Safe, and Flexible Codes for Reliable Computer Memory,” in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[20] Sun Microsystems, Inc., “T2 core microarchitecture specification.” [Online]. Available: http://www.oracle.com/technetwork/systems/opensparc/t2-06-opensparct2-core-microarch-1537749.html

[21] Advanced Micro Devices (AMD), Inc., “Kernel developer’s guide for AMD NPT family 0Fh processors,” 2007. [Online]. Available: http://developer.amd.com/wordpress/media/2012/10/325591.pdf

[22] X. Jian, H. Duwe, J. Sartori, V. Sridharan, and R. Kumar, “Low-power, low-storage-overhead chipkill correct via multi-line error correction,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*. ACM, 2013, p. 24.

[23] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, “Linearly Compressed Pages: A Main Memory Compression Framework with Low Complexity and Low Latency,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2013.

[24] J. Yang, Y. Zhang, and R. Gupta, “Frequent Value Compression in Data Caches,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2000, pp. 258–265.

[25] A. R. Alameldeen and D. A. Wood, “Frequent Pattern Compression: A Significance-Based Compression Scheme for L2 Caches,” Technical Report 1500, Computer Sciences Department, University of Wisconsin-Madison, Tech. Rep., 2004.

[26] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, “Base-delta-immediate Compression: Practical Data Compression for On-chip Caches,” in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, September 2012.

[27] B. Abali, H. Franke, D. E. Poff, R. A. Saccone, C. O. Schulz, L. M. Herger, and T. B. Smith, “Memory Expansion Technology (MXT): Software Support and Performance,” *IBM Journal of Research and Development*, vol. 45, no. 2, pp. 287–301, March 2001.

[28] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE TRANSACTIONS ON INFORMATION THEORY*, vol. 23, no. 3, pp. 337–343, 1977.

[29] L. Chen, Y. Cao, and Z. Zhang, “Free ECC: An efficient error protection for compressed last-level caches,” in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, 2013.

[30] D. J. Palframan, N. S. Kim, and M. H. Lipasti, “COP: To compress and protect main memory,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2015, pp. 682–693.

[31] H. Stracovsky, M. Espig, V. W. Lee, and D. Kim, “Reliability support in memory systems without error correcting code support,” U.S. Patent US 8 495 464 B2, 2013.

[32] D. Huffman, “A Method for the Construction of Minimum-Redundancy Codes,” *Proceedings of the IRE*, 1952.

[33] Standard Performance Evaluation Corporation, “SPEC CPU 2006.” [Online]. Available: http://www.spec.org/cpu2006/

[34] J. L. Henning, “SPEC CPU2006 Benchmark Descriptions,” *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[35] “The Princeton Application Repository for Shared-Memory Computers (PARSEC).” [Online]. Available: http://parsec.cs.princeton.edu/

[36] C. Bienia, “Benchmarking modern multiprocessors,” Ph.D. dissertation, Princeton University, January 2011.

[37] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 1995.

[38] PARSEC Group, “A Memo on Exploration of SPLASH-2 Input Sets,” 2011.

[39] T. J. Dell, “A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory,” IBM Microelectronics Division, November 1997.

[40] D. Roberts and P. Nair, “FAULTSIM: A fast, configurable memory-resilience simulator,” in *The Memory Forum: In conjunction with ISCA*, vol. 41.

[41] 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 *Programming Languages Design and Implementation (PLDI)*, 2005.

[42] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe, “SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2003, pp. 84–95.

[43] “The VIPS file format.” [Online]. Available: {http://www.vips.ecs.soton.ac.uk/supported/current/doc/html/libvips/file-format.html}

[44] Micron Technology Co., “Calculating Memory System Power for DDR3,” http://www.micron.com/-/media/Documents/Products/Technical%20Note/DRAM/TN41_01DDR3_Power.pdf, 2007.

[45] “The gem5 Simulator System: A Modular Platform for Computer System Architecture Research,” http://www.gem5.org/.

[46] Synopsys Inc., “Design Compiler I-2013.12-SP5-2,” September 2014.

[47] Taiwan Semiconductor Manufacturing Company, “40nm CMOS Standard Cell Library v120b,” 2009.

[48] X. Jian and R. Kumar, “ECC Parity: A Technique for Efficient Memory Error Resilience for Multi-Channel Memory Systems,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2014, pp. 1035–1046.