Virtualized and Flexible ECC for Main Memory

Doe Hyun Yoon
Electrical and Computer Engineering Department
The University of Texas at Austin
doehyun.yoon@gmail.com

Mattan Erez
Electrical and Computer Engineering Department
The University of Texas at Austin
mattan.erez@mail.utexas.edu

Abstract

We present a general scheme for virtualizing main memory error-correction mechanisms, which map redundant information needed to correct errors into the memory namespace itself. We rely on this basic idea, which increases flexibility to increase error protection capabilities, improve power efficiency, and reduce system cost; with only small performance overheads. We augment the virtual memory system architecture to detach the physical mapping of data from the physical mapping of its associated ECC information. We then use this mechanism to develop two-tiered error protection techniques that separate the process of detecting errors from the rare need to also correct errors, and thus save energy. We describe how to provide strong chipkill and double-chip kill protection using existing DRAM and packaging technology. We show how to maintain access granularity and redundancy overheads, even when using $\times 8$ DRAM chips. We also evaluate error correction for systems that do not use ECC DIMMs. Overall, analysis of demanding SPEC CPU 2006 and PARSEC benchmarks indicates that performance overhead is only 1% with ECC DIMMs and less than 10% using standard Non-ECC DIMM configurations, that DRAM power savings can be as high as 27%, and that the system energy-delay product is improved by 12% on average.

Categories and Subject Descriptors B.3.4 [*Memory Structures*]: Reliability, Testing and Fault Tolerance—Error-Checking

General Terms Reliability, Design

Keywords fault tolerance, error correction, memory systems, reliability

1. Introduction

Nearly all applications in all segments are requiring larger and larger main memory capacities, increasing the cost of the memory system. 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 as little impact as possible on performance, efficiency, and availability. Today's solutions are to use DRAM modules that can store redundant information and apply *error-correcting codes* (ECCs) to detect and correct errors. These *ECC DIMMs* (dual in-line memory modules) require a larger number of DRAM chips and I/O pins, which increases both price and power consumption. In fact, in some systems, the power required to operate main memory is 50% of the system power [18, 30]. To make things worse, recent trends show an increased relative likelihood of entire DRAM chips malfunctioning [13, 34], requiring even more stringent protection than just tolerating single bit errors. In this paper, we explore cooperative operating-system and hardware techniques that maintain or improve current error protection levels, while reducing cost and energy.

Typically, the *ECC DIMMs* are used to provide *single-bit-error-correction and double-bit-error-detection* (SEC-DED) for each DRAM rank, and do so without impacting memory system performance. Tolerating failures of entire DRAM chips requires the use of *chipkill correct*, which "spreads" a DRAM access across multiple chips and uses a wide ECC to allow strong error tolerance [5, 12, 40]. The downside of chipkill is that it either increases memory access granularity, requiring more energy and restricting possible DRAM configurations, or increases the required level of redundancy, which again, increases cost [3]. The added 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 unless chipkill is used [20]. Providing chipkill protection in current DRAM packaging technology is expensive and requires the use of $\times 4$ DRAM configuration (will be discussed in Section 2.2). These narrow chips consume roughly 30% more energy for a given total DIMM capacity as more efficient $\times 8$ configurations [6]. This extra overhead is added to all the memory in these large capacity systems, which may be multiple terabytes if a memory extension appliance is used [24, 44]. We offer a fundamentally different approach to storing and manipulating redundant DRAM storage, which brings flexibility and generality to memory error protection.

Our architecture virtualizes the storage of redundant information, using mechanisms that are similar to virtual memory management [35]. We allow the system to dynamically choose the location of redundant data, instead of always storing redundant information in dedicated and aligned DRAM chips. 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 accessing 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 Virtualized ECC approach, we describe schemes ranging from low-cost protection, which uses Non-ECC DIMMs, up to double-chipkill techniques for high availability systems. We also show how virtualization enables us to maintain protection capability even when

This work is supported in part by donations from the Intel corporation.

The definitive version was published in the proceedings of *ASPLOS'10*, March 13–17, 2010, Pittsburgh, Pennsylvania, USA.

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 [37] and PARSEC suites [7] that have high memory access demands, as well as targeted micro-benchmarks.

If ECC DIMMs are available, we develop a range of techniques that are based on a two-tiered memory protection approach [33, 48, 49]. The main innovation of Virtualized ECC is dynamic adaptivity: Virtualized ECC can dynamically map redundant information and vary error protection level based on user or system needs. In two-tiered protection, the extra chips of the ECC DIMMs are used to store error-detection information only while the full redundant information required for correction is mapped as data. Error detection is performed on every DRAM access, but it only requires a subset of the total redundant information, which reduces energy consumption. The full redundancy is only needed on the very rare case that an error is detected, or for DRAM writes. Because the error correction information shares the DRAM namespace with the data, we can cache it to avoid some of the degradation in performance resulting from the added accesses. The flexibility of Virtualized ECC enables us to achieve very strong protection, even when relying on $\times 8$ DRAM chips, which is not possible with a one-tier scheme. We show that EDP for chipkill can be improved by 12% and that performance is hardly impacted, dropping by only $1-2\%$. We also describe a double-chipkill technique that can protect against two entire DRAM chips failing using standard DIMMs.

In addition, our low-cost schemes enable error correction for systems that do not use specialized ECC DIMMs. We modify the OS to allocate storage for redundant information as necessary and associate it with the data through an architectural interface. The memory controller then performs two accesses for each DRAM read or write, one for the data and one for the ECC information. We show that this approach only degrades performance by $3-9\%$ on average and no more than $10-25\%$, while reducing power consumption because fewer DRAM chips are used. Overall, the system EDP can be improved by up to 12% when using $\times 8$ and $\times 16$ DRAMs.

We make four important contributions:

- We present a general OS/architecture mechanism for virtualizing DRAM ECC protection and decoupling the mapping of redundant information from the mapping of the data.
- We develop two-tiered protection for DRAM and show how to improve reliability guarantees and reduce the power consumption of the DRAM system, by using wider-access configurations.
- We provide a flexible error protection mechanism that can adapt error protection level to match application, user, and system needs.
- We describe how to provide ECC protection for systems with standard Non-ECC DIMMs without requiring changes to data mapping, and evaluate the performance and power impact using a variety of protection levels and DRAM configurations.

The rest of the paper is organized as follows: Section 2 provides background on memory systems and DRAM error protection; Section 3 details the architecture and OS requirements as well as the protection schemes; Section 4 presents our evaluation; Section 5 discusses related work; and Section 6 concludes the paper.

2. Background

In this section we very briefly describe modern memory systems and error protection techniques including chipkill correct. We focus on commodity DDRx DRAM based systems since a vast majority of computing platforms including high-end servers uses such devices.

2.1 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 an illusion of a flat and uniform address space to software. Physical memory is, however, composed of memory channels, each with multiple memory modules. In the rest of this subsection, we briefly review memory system organization, from DRAM chips to memory modules, since they are closely related to memory error protection as described in Section 2.2. A more complete description of memory systems is available in [21].

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 – 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, is a physical device that has, typically, 1 to 8 ranks; a standard 64-bit wide DIMM is also referred to a Non-ECC DIMM to differentiate it from an ECC DIMM explained in Section 2.2. 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. A DIMM with wider DRAMs ($\times 8$ or $\times 16$), if total capacity is the same, has fewer DRAM chips and consumes less power [6], hence $\times 8$ or $\times 16$ DIMMs are preferred. Systems that require high reliability/availability, however, favor $\times 4$ DIMMs; especially systems with chipkill correct as explained in the next subsection.

2.2 Memory Error Protection

Main memory is a major source of system malfunctions due to both soft errors and device failures. Traditional memory error protection applies ECC uniformly across the whole memory. We first discuss the commonly used single-bit-error-correcting and double-bit-error-detecting (SEC-DED) protection scheme that uses ECC DIMMs, then move on to stronger *chipkill correct* and its related module design.

The SEC-DED code [17, 19] typically uses 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 are used to store both the data and the redundant information. An ECC DIMM is constructed using 18 $\times 4$ chips ($\times 4$ ECC DIMM) or 9 $\times 8$ chips ($\times 8$ ECC DIMM), but there is no $\times 16$ 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 error protection mechanism to system designers.

High-end servers require an even stronger error protection mechanism that can tolerate a device failure, *chipkill correct* [12]. Chipkill correct trades off a larger access granularity to maintain 12.5% storage redundancy. As a result chipkill correct can currently only use $\times 4$ ECC DIMMs as explained below.

The error code for chipkill correct is a single-symbol-error-correcting and double-symbol-error-detecting (SSC-DSD) code. It uses *Galois Field* (GF) arithmetic [25] with b-bit symbols to tol-

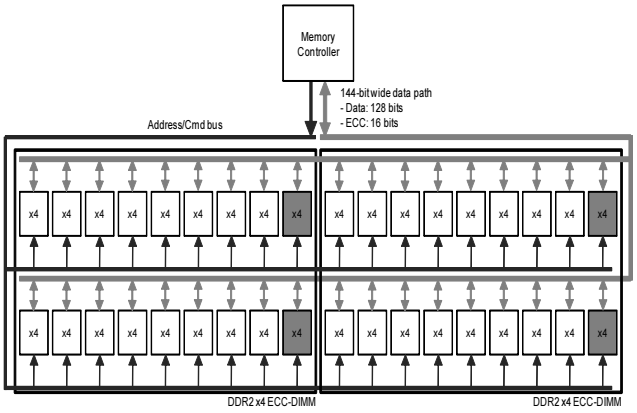
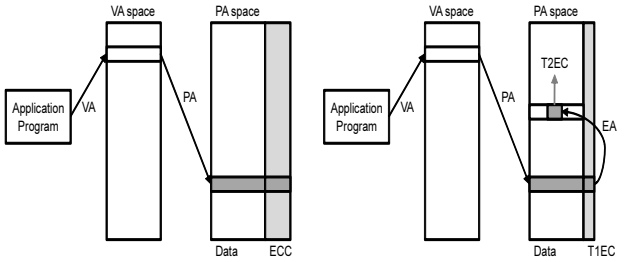


Figure 1: Baseline chipkill correct DRAM configuration (grey DRAMs are dedicated to ECC storage).



(a) Conventional architecture (b) Virtualized ECC architecture

Figure 2: High-level view of memory accesses in a conventional virtual memory with fixed ECC, and Virtualized ECC with a two-tiered flexible protection scheme.

erate up to an entire chip failing in a memory system. The 3-check-symbol error code [10] is the most efficient SSC-DSD code in terms of redundancy overhead. The code-word length of the 3-check-symbol code is, however, limited to $2^b + 2$ symbols, so it is a poor match for $\times 4$ configuration; using $\times 4$ DRAMs leads to a granularity mismatch that results in data words that are 60 bits long - a non power of two (3 4-bit check symbols can correct a symbol error in 15 data symbols). Instead, the 4-check-symbol error code [9] is used for $\times 4$ DRAMs, where the 4^{th} check symbol allows a longer code-word. Four 4-bit check symbols provide SSC-DSD protection for 32 4-bit data symbols, resulting in an access granularity of 128 bits of data with 16 bits of redundant information; this wide data-path is implemented using two $\times 4$ ECC DIMMs in parallel as shown in Figure 1. This organization is used by the Sun UltraSPARC-T1/T2 [40] and the AMD Opteron [5]. We use this $\times 4$ chipkill configuration as the baseline throughout this study. This chipkill memory system 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 with longer burst leading larger access granularity [3].

Wider DRAMs ($\times 8$ and $\times 16$) can use the 3-check-symbol code, since the maximum code-word length increases with 8- or 16-bit symbol. Supporting chipkill with $\times 8$ and $\times 16$ is, however, impractical. First of all, chipkill using $\times 16$ DRAMs is not possible unless we design a custom $\times 16$ ECC DIMM, which is expensive. Even for $\times 8$ configuration, 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% [3] (128-bit data and 24-bit ECC), and it also requires a custom $\times 8$ ECC DIMM having 16 $\times 8$ DRAMs for data and 3 $\times 8$ DRAMs for ECC, that then increases costs. Maintaining the same (or less) 12.5% ECC storage overhead, on the other hand, will double 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.

In summary, uniform error protection is transparent to software, but comes at a cost of additional storage and data pins for redundant information. Supporting chipkill exacerbates the inefficiencies by constraining memory module design to use a less energy efficient $\times 4$ configuration. In constrast to uniform error protection, Virtualized ECC, with two-tiered error protection and redundant storage virtualization, relaxes DIMM constraints even for chipkill, and enables flexible error protection.

3. Virtualized ECC Architecture

The main innovation of Virtualized 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. Virtualized ECC offers an opportunity to tailor memory protection levels and avoid the need to uniformly pay the overhead for worst-case scenarios [46]. 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 3.1), examples of possible protection techniques (Section 3.2), and the OS virtual memory interface (Section 3.3).

There are two basic mechanisms underlying Virtualized 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, where 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 [33, 48, 49]. Figure 2 compares traditional VM, with its fixed relation between data and ECC, and the decoupled two-tier approach of Virtualized ECC. Traditional VM (Figure 2(a)) translates a virtual address from the application namespace to a physical address in DRAM. A DRAM access then retrieves or writes both the data and the ECC information, which is stored aligned with the data in the dedicated ECC DRAM chips.

Figure 2(b) gives an example of a flexible mapping enabled by Virtualized 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 3.3). Thus, less total data is accessed on a read in Virtualized 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 Virtualized 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.

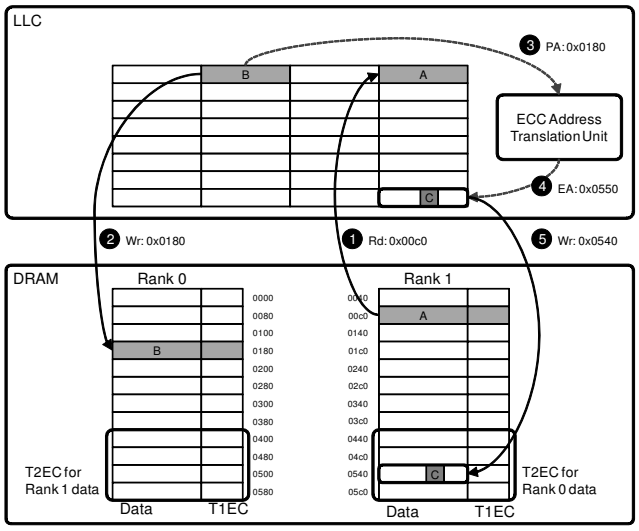


Figure 3: Operations of DRAM and LLC for accessing a two-tiered Virtualized ECC configuration.

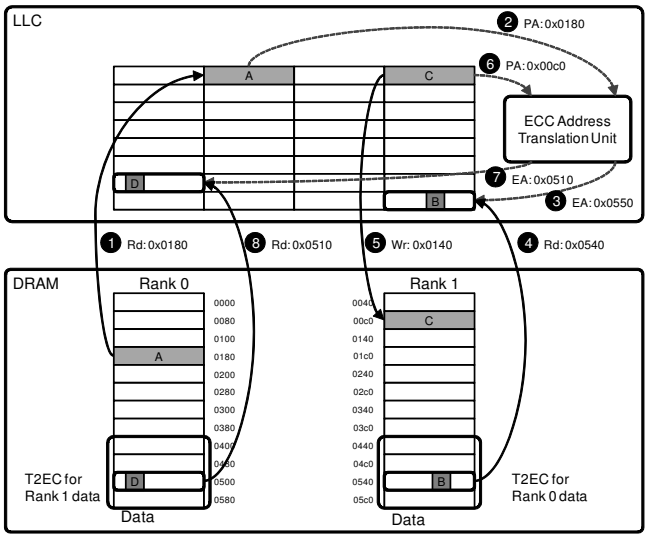


Figure 4: Operations of DRAM and LLC for accessing Virtualized ECC in a Non-ECC DIMM configuration.

3.1 Cache-DRAM Interface

The cache filters requests from the core to the DRAM system and can also help in improving the performance of Virtualized 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. Virtualized ECC does not require significant changes from the existing cache interface, with the additional hardware being the ECC address translation unit (described in Section 3.3) 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 DRAM 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.

3.1.1 Two-Tiered DRAM ECC

Figure 3 shows our two-tiered Virtualized 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 3. 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. 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. 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 repurpose 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. 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 [38, 39].

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. Very frequent errors indicate a hard-fault and can be mitigated by data migration, as suggested by Slayman [36].

3.1.2 Virtualized ECC Interface with Non-ECC DIMMs

Even if physical memory does not provide ECC storage, we can use Virtualized ECC to protect memory against errors. In the Non-ECC DIMM configuration, 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 4 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

Table 1: DRAM configurations for chipkill correct of the baseline system, MC-DIMMs that are non-standard DIMMs introduced in [3] to improve DRAM power consumption, and Virtualized ECC [1].

	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	$\times 4$	32	4	2 ECC DIMMs	N	N	N/A
MC-DIMM [3] with 128bit channel and 4 rank subsets							
MC-DIMM x4	$\times 4$	32	12	2 MC-DIMMs	N	N	N/A
MC-DIMM x8	$\times 8$	16	12	2 MC-DIMMs	N	N	N/A
MC-DIMM x16	$\times 16$	8	12	2 MC-DIMMs	N	N	N/A
Virtualized ECC							
ECC $\times 4$	$\times 4$	32	2	1 ECC DIMM and 1 Non-ECC DIMM	N	Y	2B
ECC $\times 8$	$\times 8$	16	2	2 ECC DIMMs	N	Y	4B
Non-ECC $\times 4$	$\times 4$	32	N/A	2 Non-ECC DIMMs	Y	Y	4B
Non-ECC $\times 8$	$\times 8$	16	N/A	2 Non-ECC DIMMs	Y	Y	8B
Non-ECC $\times 16$	$\times 16$	8	N/A	2 Non-ECC DIMMs	Y	Y	16B

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.

3.2 Virtualized ECC Protection Schemes

We now discuss possible DRAM configurations for Virtualized ECC, assuming a memory system that is representative of servers that require chipkill correct error 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 Section 2.2. We use DDR2 in this paper because of a readily available power model [45]. Our techniques work as well, or better, with DDR3 [39] because it uses longer bursts and limits use of traditional chipkill techniques [3].

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

3.2.1 Virtualized ECC with ECC DIMMs

While traditional chipkill uses a 4-check-symbol error code as explained in Section 2.2, Virtualized ECC, with two tiered protection and T2EC storage virtualization, enables a more efficient 3-check-symbol error code [10]. In two-tiered 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 error code. This effective 256-bit access does not actually change the DRAM access granularity, which is still 64 bytes as in the baseline DDR2-based chipkill system.

The other key point is to use Virtualized 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 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 [3].

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 2 chip failures). The third check symbol is the T2EC, which is stored in the data chips[3]. Thus, ECC $\times 4$ 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 $\times 8$ is an efficient scheme for chipkill protection using ECC DIMMs with $\times 8$ 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 $\times 8$ 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 pin requirements and redundancy to a fixed 18.5% [3], 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 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 [13]; and energy-efficient server systems that increasingly operate with narrower noise margins due to reduced cooling and voltage levels.

3.2.2 Virtualized ECC with Non-ECC DIMMs

Another advantage of Virtualized 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 [32], 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

[1] Non-ECC configurations of Virtualized ECC cannot detect 2 chip failures.

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

[3] We omit the full details of the error correction coding scheme, such as the parity check matrix and syndrome description, due to format and space restrictions in this paper.

Table 2: Virtualized ECC configurations for flexible error protection achieved by varying T2EC size[4].

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

scheme using $\times 4$, $\times 8$, and $\times 16$ DRAM chips are also summarized in Table 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 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 ECC 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 4.2.1).

3.2.3 Flexible Protection Mechanisms

An exciting feature of Virtualized 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 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 the latter is to augment the OS virtual memory page table and hardware TLB to include protection information for each page. We do not explore the benefits of protection tuning in this paper, 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). Virtualized ECC offers opportunities for new tradeoffs, such as the double-chipkill technique described above or reduced power consumption by turning ECC off.

3.3 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.

3.3.1 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 [35], 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 3.3.2). 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 [2].

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 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 paper, however, we always maintain active T2EC data in DRAM.

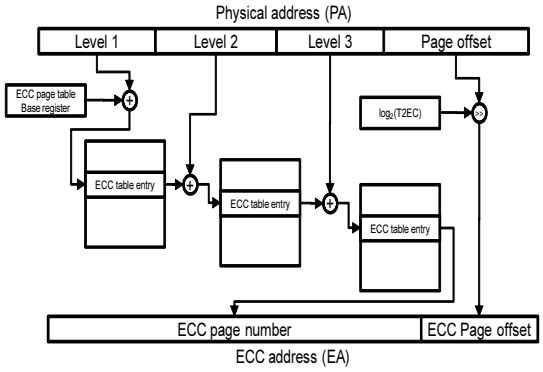


Figure 5: PA to EA translation using a three-level hierarchical ECC table mapping, similar to virtual memory address translation.

3.3.2 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 to the VA to PA page table. Figure 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 [47]. Note that the

[4] *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.

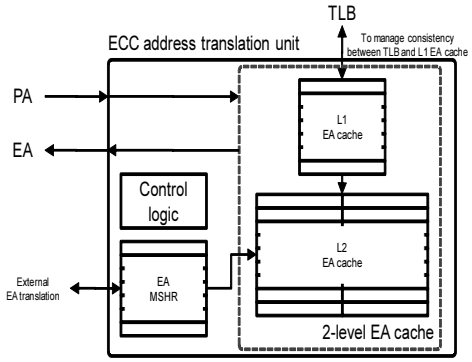


Figure 6: ECC address translation unit with a two-level EA translation cache.

Table 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 $\times 4$ chipkill correct open page, Read and Instruction Fetch First XOR based rank/bank interleaving [50] is used

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 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 4. 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 4.2.2. 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.

4. Evaluation

We evaluate Virtualized ECC using a combination of detailed cycle-based simulation to determine performance and power impact, and use the PIN binary instrumentation tool [26] 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 [27] with the GEMS [28] 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 integrated DRAMsim [45] with GEMS to accurately account for the impact of Virtualized ECC on memory bandwidth, performance, and power. Table 3 describes the baseline system configuration, which uses uniform chipkill ECC with $\times 4$ DRAMs.

We estimate DRAM power consumption using a power model developed by Micron Corporation [1] that is embedded within DRAMsim. For processor power analysis, we use WATTCH [8] to model out-of-order processor energy consumption. We updated the energy parameters to a 45nm technology based on technology scaling rules. We also estimate processor power consumption using IPC-based linear scaling of the maximum power consumption of a 45nm Intel Xeon [22], as suggested in [3]. The estimates of the two power models closely matched one another. Finally, we estimate cache power consumption using CACTI 5 [43].

4.1 Workloads

We use a mix of several SPEC CPU 2006 [37] and PARSEC [7] applications, as well as the STREAM [29] and GUPS [14] micro-benchmarks. We selected only applications that stress the memory system and that are potentially significantly impacted by Virtualized 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-accurate 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 used the reference input dataset and ran cycle accurate simulations for 200M representative instructions, as indicated by Simpoint [16], and ran the applications to completion using PIN. We used the simsmall dataset for the cycle-based simulations of PARSEC and simlarge with PIN. We ran all the PARSEC applications to completion using a single thread. Table 4 summarizes the IPC, cache miss ratio, and power consumption of the applications on the baseline system.

4.2 Results and Analysis

We present our analysis of Virtualized ECC in four parts: the T2EC information storage and translation management (Section 4.2.1–4.2.2), the overall performance and energy impact for single chipkill ECC (Section 4.2.3); the flexible reliability schemes (Section 4.2.4); and implications on multicore processors that have a lower effective memory bandwidth for each core (Section 4.2.5).

4.2.1 Virtualized ECC Storage Management

Maintaining T2EC information as data implies sharing storage resources between application data and its ECC redundancy, impacting data caching behavior. Figure 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).

Table 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

In general, T2EC miss rate and occupancy are directly proportional to the size of each T2EC (see Table 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 $\times 4$ and ECC $\times 8$) and those that do not (Non-ECC $\times 4$, Non-ECC $\times 8$, and Non-ECC $\times 16$). 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 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 $\times 16$ 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 8, but as we discuss in Section 4.2.3 and Section 4.2.5, 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.

4.2.2 Virtualized ECC Translation Management

In this subsection, we use PIN [26] 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 6 with a fully associative 16 entry L1 cache (same size as the UltraSPARC-III TLB [41]) 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, 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), 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 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 virtual memory [42]. 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.

Table 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

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.

4.2.3 Chipkill Performance and Energy

Figure 9 presents the execution time and system energy-delay product (EDP) of the Virtualized ECC configurations described in Table 1, normalized to those of the baseline $\times 4$ chipkill ECC. For system EDP, we measured system power consumption as the sum of processor core power, LLC power, and DRAM power. Virtualized 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 $\times 4$ and ECC $\times 8$ 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

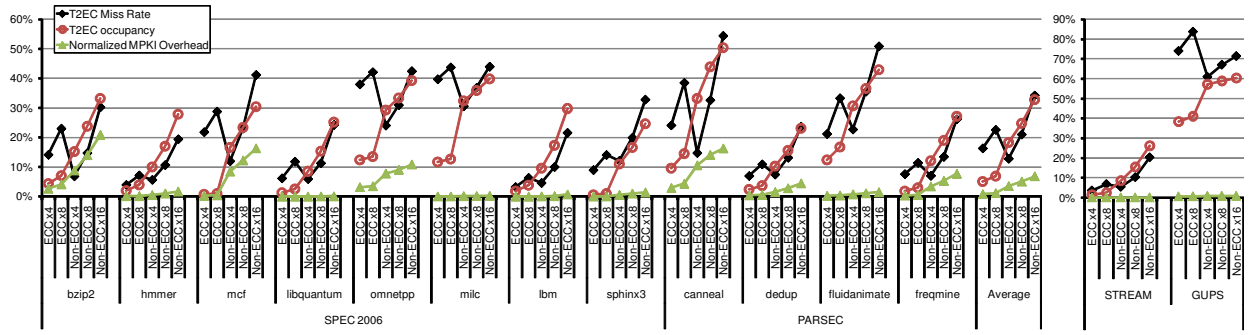


Figure 7: Impact on the LLC: T2EC miss rate and occupancy in the LLC as well as impact on application data miss rate.

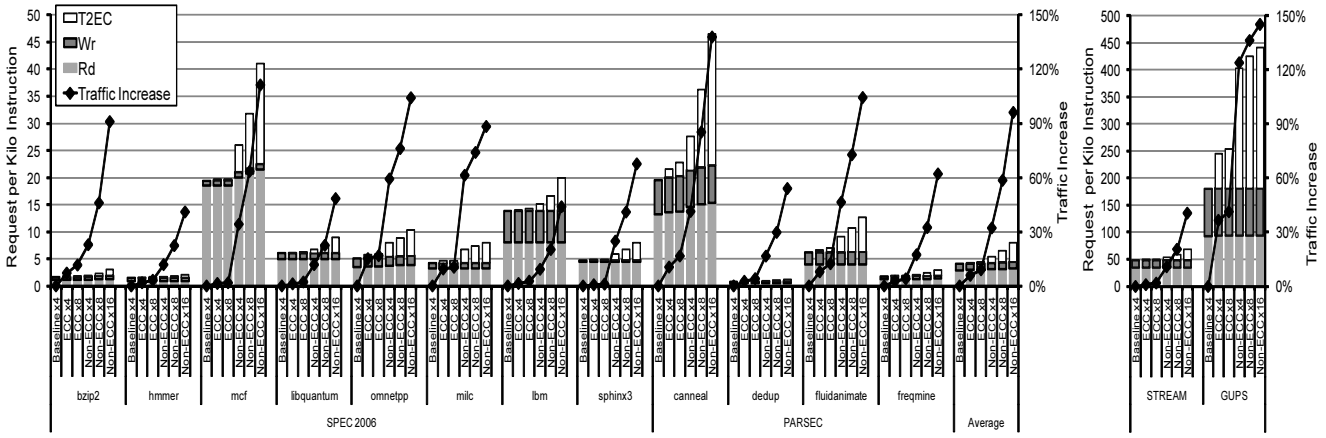


Figure 8: Impact of Virtualized ECC on DRAM traffic.

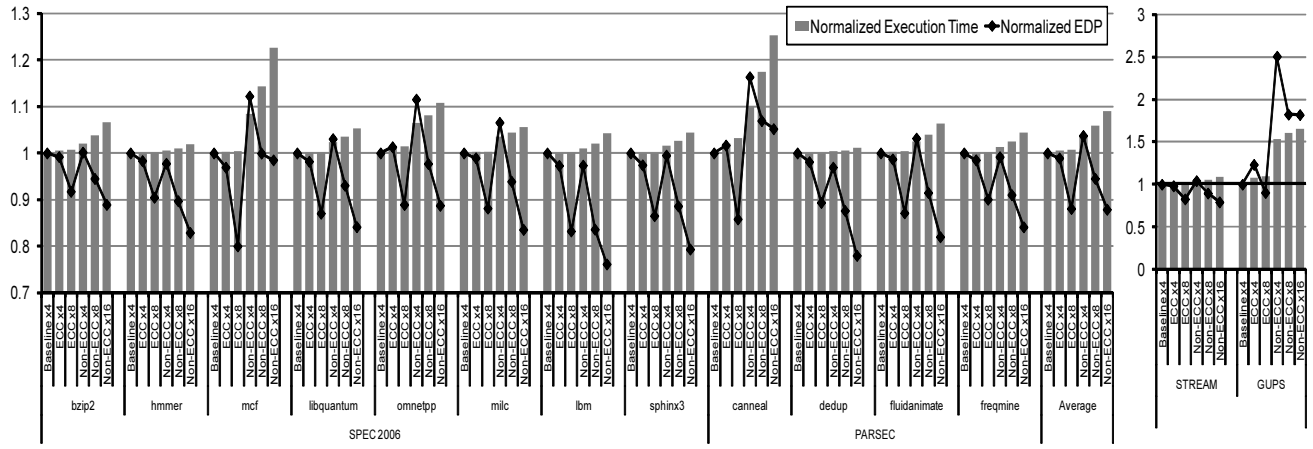
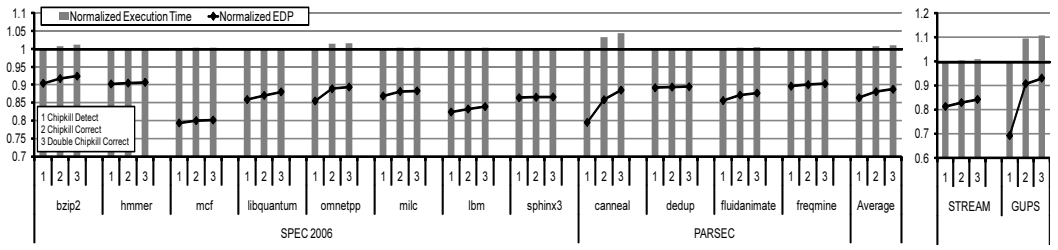
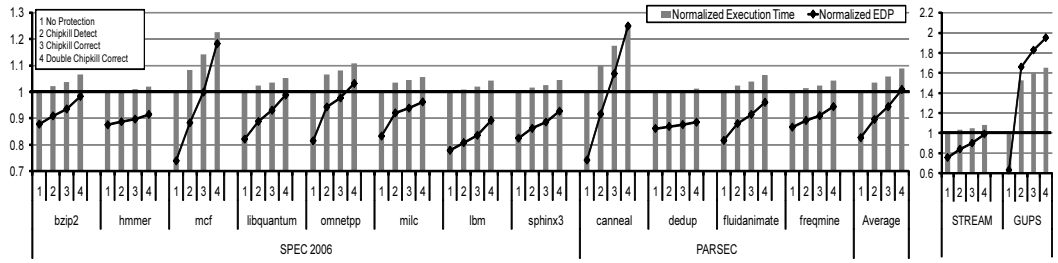


Figure 9: Performance and system EDP for baseline and Virtualized ECC chipkill correct.

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 4), and thus benefit more from increased memory power efficiency. `GUPS` demands even higher memory performance. While the EDP



(a) $\times 8$ ECC DIMM



(b) $\times 8$ Non-ECC DIMM

Figure 10: Performance and system EDP while varying error protection level (execution time and system EDP are normalized to those of baseline $\times 4$ chipkill).

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.

Virtualized 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` and `GUPS` have a relatively worse EDP with Non-ECC $\times 8$ and Non-ECC $\times 16$. 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.

4.2.4 Flexible Protection

As described in Section 3.2.3 the Virtualized ECC architecture enables flexibility in choosing the error protection level based on dynamic application, user, and system needs. To assess a new tradeoff that Virtualized ECC enables, we evaluate the effect of different T2EC sizes, which are summarized in Table 2. Figure 10 presents the execution time and system EDP, normalized to those of the baseline system, when using $\times 8$ DRAM chips and 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 10(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 10(b) shows the normalized execution time and EDP of Non-ECC $\times 8$. The overall system EDP is better than the baseline in all configurations, with the exception of `mcf`, `omnetpp`, `canneal`, and `GUPS`; Non-ECC $\times 16$ also has similar behavior. 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 6 summarizes the average performance penalty and DRAM power/system EDP gains of ECC $\times 8$, Non-ECC $\times 8$, and Non-ECC $\times 16$; $\times 4$ configurations are omitted since they do not show much gain.

4.2.5 Implications on Multicore Processors

So far our design and evaluation have focused on a single-core system. As described in Section 3.1, T2EC information is exclusively accessed by a DRAM controller and not shared among multiple processor cores. Therefore, it is easy to integrate Virtualized ECC within a multiprocessor or multicore system. In multicore systems, however, the increased traffic due to T2EC accesses and 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 $\times 8$ configurations of Virtualized ECC on a multicore by simulating a single-core system with half the memory bandwidth (6.4GB/s). We believe this is indicative of the system properties in a multiprogrammed environment and will serve as a basis to drive future research.

The results of this reduced-bandwidth experiment are shown in Figure 11. The relative performance penalty of ECC $\times 8$ and Non-ECC $\times 8$ 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 $\times 8$ 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 [23].

Table 6: Summary of flexible error protection with Virtualized ECC

	ECC $\times 8$			Non-ECC $\times 8$				Non-ECC $\times 16$			
	chipkill detect	chipkill correct	double chipkill correct	no protection	chipkill detect	chipkill correct	double chipkill correct	no protection	chipkill detect	chipkill correct	double chipkill correct
Performance penalty	0%	0.7%	1%	0%	3.4%	5.8%	8.9%	0%	5.8%	8.9%	12.8%
DRAM power reduction	29%	27.8%	27.2%	37.1%	32.6%	30.1%	26.7%	59.5%	53.4%	50.1%	46.2%
System EDP gain	14.6%	12%	11.2%	17.3%	10.4%	5.6%	−0.9%	27.8%	17.8%	12.1%	4.9%

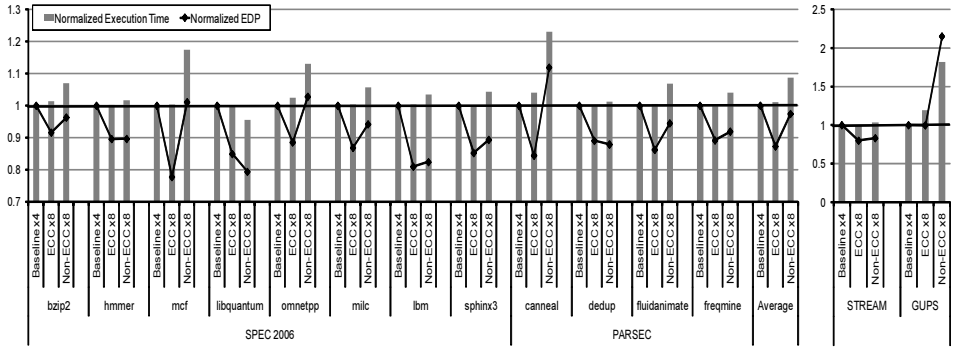


Figure 11: Performance and system EDP for $\times 8$ Virtualized ECC in a half-bandwidth (6.4GB/s) multicore-like scenario.

5. Related Work

The proposed memory error protection mechanisms build on a large body of prior work, including general work on error coding [9, 10] and chipkill correct [5, 12, 40]. 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 Virtualized ECC, that uses the data chips to also store redundant information is mentioned in [51]. 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 [11, 15] 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 [51], MC-DIMMs [4] 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 [3]. These techniques, like the ones in Section 3.2, 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 [2] 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 [46] 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.

The basic idea of two-tiered memory protection has been suggested in the past for on-chip caches. The Punctured ECC Recovery Cache (PERC) [33] proposed decoupled error codes, where cache reads only perform error detection to save energy while writes update both detection and correction information. Memory-Mapped ECC (MME) [49] generalized this idea; MME uses two-tiered ECC and stores second-tier information as data within the cached memory hierarchy. We extend the work on MME to address the needs of main memory reliability and study flexible approaches and benefits to protection and energy consumption.

6. Conclusions and Future Work

In this paper we presented general mechanisms to virtualize ECC-based memory error tolerance mechanisms. We showed how Virtualized 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 paper has been on developing the fundamental mechanisms and evaluating potential ECC schemes that take advantage of Virtualized ECC. We showed promising early 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 described in this paper are just the starting point for opportunities to take advantage of dynamically tunable memory protection.

In our future work we will explore tighter interaction and collaboration between the application software and the reliability layer to improve overall performance and efficiency. We believe one key to

increasing efficiency is to exploit flexible mechanisms and not commit redundancy and overhead based on the intersection of all worst-case scenarios. We will develop the infrastructure and evaluate the potential of varying the error tolerance methods used on different applications, phases, and memory regions using the techniques we described. We also plan to apply Virtualized ECC to other architectures, such as GPUs, where error protection is an emerging requirement [31], and memory extension appliances [24, 44]. GPU design is perhaps more cost-sensitive than traditional processors because the main market is not for general-purpose computing and does not require strict reliability and availability guarantees.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] AMD. BIOS and kernel developer's guide for AMD NPT family 0Fh processors, Jul. 2007. URL `http://support.amd.com/us/Processor_TechDocs/32559.pdf`.

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

[7] 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.

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

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

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

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

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

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

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

[15] 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.

[16] 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*, Jun. 2005.

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

[18] HP. Server power calculators. URL `http://h30099.www3.hp.comconfigurator/powercalcs.asp`.

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

[20] IBM. Enhancing IBM Netfinity server reliability, 1999.

[21] B. Jacob, S. Ng, and D. Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2007.

[22] 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. Int'l Solid State Circuits Conf. (ISSCC)*, Feb. 2009.

[23] 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.

[24] 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.

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

[26] 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 SIGPLAN Conf. Programming Language Design and Implementation (PLDI)*, Jun. 2005.

[27] 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.

[28] 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.

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

[30] U. Nawathe, M.Hassan, L. Warriner, K. Yen, B. Upputuri, D.Greenhill, A.Kumar, and H. Park. An 8-core, 64-thread, 64-bit, power efficient SPARC SoC. In *Proc. the Int'l Solid State Circuits Conf. (ISSCC)*, Feb. 2007.

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

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

[33] 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.

[34] 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.

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

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

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

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

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

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

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

[42] 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.

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

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

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.