ArchShield: Architectural Framework for Assisting DRAM Scaling by Tolerating High Error Rates

Prashant J. Nair Dae-Hyun Kim Moinuddin K. Qureshi
School of Electrical and Computer Engineering
Georgia Institute of Technology
{pnair6, dhkim, moin}@gatech.edu

ABSTRACT

DRAM scaling has been the prime driver for increasing the capacity of main memory system over the past three decades. Unfortunately, scaling DRAM to smaller technology nodes has become challenging due to the inherent difficulty in designing smaller geometries, coupled with the problems of device variation and leakage. Future DRAM devices are likely to experience significantly high error-rates. Techniques that can tolerate errors efficiently can enable DRAM to scale to smaller technology nodes. However, existing techniques such as row/column sparing and ECC become prohibitive at high error-rates.

To develop cost-effective solutions for tolerating high error-rates, this paper advocates a cross-layer approach. Rather than hiding the faulty cell information within the DRAM chips, we expose it to the architectural level. We propose *ArchShield*, an architectural framework that employs runtime testing to identify faulty DRAM cells. ArchShield tolerates these faults using two components, a *Fault Map* that keeps information about faulty words in a cache line, and *Selective Word-Level Replication (SWLR)* that replicates faulty words for error resilience. Both Fault Map and SWLR are integrated in reserved area in DRAM memory. Our evaluations with 8GB DRAM DIMM show that ArchShield can efficiently tolerate error-rates as higher as 10^{-4} (100x higher than ECC alone), causes less than 2% performance degradation, and still maintains 1-bit error tolerance against soft errors.

Categories and Subject Descriptors

B.3.4 [**Hardware**]: [Memory Structures - Reliability, Testing, Fault Tolerence]

Keywords

Dynamic Random Access Memory, Hard Faults, Error Correction

1. INTRODUCTION

Dynamic Random Access Memory (DRAM) has been the basic building block for main memory systems for the past three decades. Scaling of DRAM to smaller technology nodes allowed more bits in the same chip area, and this has been a prime driver for increasing the main memory capacity. Data is stored in a DRAM cell as charge on a capacitor. As we scale down the feature size, the amount of charge that must be stored on the capacitor must still remain constant in order to meet the retention time requirements of DRAM. DRAM technology has already reached sub 30nm regime, and it is becoming increasingly difficult to further scale the cells to smaller geometries. The challenge lies not only in inherent problems of fabricating small cylindrical cells for the capacitor but also from the increased variability and leakage across cells. Recently, DRAM scaling challenges have caused the community to look at alternatives technologies for main memory [1–3]. Until a viable DRAM replacement that is competitive in terms of cost and performance becomes commercially available, scaling DRAM to smaller feature sizes will continue to be critical for future systems.

The smaller geometry and increased variability for future technologies are likely to result in higher error-rates. To maintain system integrity, faulty DRAM cells must either be decommissioned or corrected. If the cost of tolerating faulty cells is significantly higher than the capacity gains from moving from a given technology node to a smaller technology node, future technology nodes may be deemed unviable, thus halting DRAM scaling. Therefore, techniques that can tolerate high error-rates at low cost can allow DRAM to scale to smaller technology nodes than possible with traditional techniques.

Figure 1 shows different schemes to mitigate errors in DRAM (without loss of generality, we consider 8GB Dual Inline Memory Module (DIMM) in our studies). If the bit error-rate (BER) of DRAM cells is less than 10^{-12} then the memory system may not need any error correction for faulty cells. Current DRAM systems rely on sparing of rows/columns to tolerate faulty cells. For example, with row sparing, the DRAM row containing the faulty DRAM cell is replaced by one of the spare rows. This method incurs an overhead of about 10K-100K bits (and several laser fuses) for tolerating one faulty bit. While seemingly expensive, this method works quite well at low bit error-rates that are typical in current DRAM chips. Unfortunately, the high cost makes this technique impractical for high error-rates.

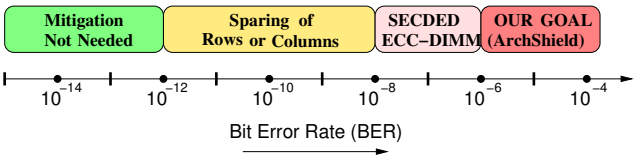

Figure 1: Fault mitigation technique depends on bit error-rate (BER). Row sparing works well at low error-rates and SECDED-based DIMMs can tolerate BER of approximately 10^{-6}. We target a BER that is about 100x higher.

ISCA'13 Tel-Aviv, Israel

Another alternative to tolerate errors in DRAM is to use Error Correcting Code (ECC). Commodity DIMMs are also available with ECC, which can correct one bit out of the 8-byte word. While these DIMMs are aimed at tolerating soft errors, we can also use it to tolerate faulty DRAM cells. However, using such DIMMs to tolerate random bit errors, is still ineffective for high bit error-rates. Our analysis shows that ECC DIMMs can tolerate an error-rate of only in the regime of about 1 faulty cell per million. To tolerate higher error-rates, we would need higher levels of ECC. For example, for tolerating an error-rate of 10^{-4} we need 3-bit error correction per 64-bit word. Such high level of ECC is expensive in terms of both storage and latency. Furthermore, this approach sacrificed soft error resilience for tolerating faulty cells, and would need additional ECC to tolerate soft errors. Ideally, we want to use ECC DIMMs to tolerate both faulty cells due to manufacturing and soft errors due to alpha particles.

We advocate exposing the information about the faulty DRAM cells to the hardware, so that the amount of error tolerance can be tailored to the vulnerability level of each word. We propose such an architecture-level framework called *ArchShield*. ArchShield is built on top of commodity ECC DIMMs, and is geared towards tolerating 100x higher error-rates than can be handled by ECC DIMMs alone, while retaining the soft error tolerance. When a new DIMM is configured in the system, ArchShield performs a runtime testing of the DIMM to identify the faulty cells in the memory. In particular, it tracks if the given 64-bit word has zero error, one error, or more than one error.

ArchShield contains a *Fault Map* that stores information about faulty words on a per line basis. All faulty words (including the ones with one-bit error) are replicated in a spare region. Such *Selective Word Level Replication (SWLR)* allows decommissioning for words with multi-bit error, while providing soft error protection for words with one-bit error. On a memory access, the fault map entry is consulted. If the line is deemed to have a word with more than 1 error, the replication area is accessed to obtain the replicated words for the corresponding line. Whereas, if the line is deemed to have a word with 1-bit error, the replicated copy is accessed only when an uncorrectable fault is encountered at the original location, which allows fast access in common case. Thus, ArchShield can tolerate multi-bit errors, while retaining soft error protection of 1-bit error correction per word.

The Fault Map and word-level repair of ArchShield is inspired, in part, by similar approach to dealing with high error-rate in current Solid State Disk (SSD). Similar to SSD, we propose to embed the Fault Map and Replication Area in reserved portion of the DRAM memory. This reduces the effective main memory visible to the operating system. Fortunately, the visible address space provided by ArchShield is contiguous, so ArchShield can be employed without any software changes (except that the memory is deemed to have smaller capacity). Similarly, ArchShield does not require any changes to the existing ECC DIMMs, and only minor changes to the memory controller to do runtime testing, orchestrate Fault Map access, and update and access replicas.

We perform evaluations with 8GB DIMM. We show that for tolerating an error-rate as high as 10^{-4}, ArchShield requires 4% memory space, and causes a performance degradation of less than 2% due to the extra memory traffic of Fault Map and SWLR. ArchShield provides this while maintaining a soft error protection of 1-bit error per ECC word.

We also show how ArchShield can be used to reduce refresh operations in DRAM systems. With ArchShield, the system can reduce the refresh rate by almost 16x, and thus reduce refresh power and and performance penalties.

2. BACKGROUND AND MOTIVATION

The DRAM industry is on track to meet the ITRS projection of 28nm technology node for 2013 [4]. The ITRS road-map for the next decade projects DRAM technology node of 10nm in 2022, in essence a new technology node every three years. If DRAM technology could be kept on this scaling curve, we can expect a doubling of memory capacity of DRAM modules every three years. Unfortunately, scaling DRAM to smaller technology nodes has become quite challenging. In addition to the typical problems of scaling to smaller geometries, DRAM devices face several additional barriers.

2.1 Why DRAM Scaling is Challenging

The capacitive element used to store charge in DRAM is typically made as a vertical structure to save chip area (as shown in the inset in Figure 2). To meet the DRAM retention time, the capacitance stored on the DRAM device needs to be approximately 25fF. When DRAM technology is scaled to smaller node, the linear dimensions scale by approximately 0.71x, the surface area of the cell reduces to approximately 0.5x, which means the depth of the vertical structure must be doubled to obtain the same capacitance. Let *Aspect Ratio* be the ratio of the height of the cell to the diameter. As shown in Figure 2, the aspect ratio has been increasing exponentially and is expected to reach more than 100x at sub 20nm [5]. Such narrow cylindrical cells are inherently unstable due to mechanical reasons, hence difficult to fabricate reliably [6].

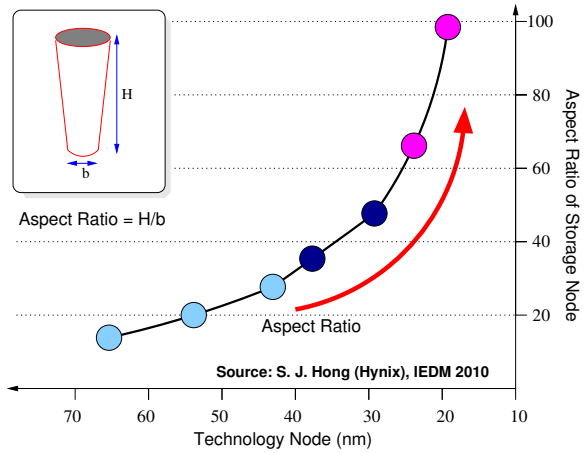

Figure 2: Exponential increase in aspect ratio of DRAM cells with scaling to smaller technology nodes (redrawn from [5])

The second problem is reduction in the thickness of the dielectric material of the DRAM cell. This makes it challenging to ensure the same capacitance value, given the unreliability of the ultra-thin dielectric material.

The third problem is the increase in gate induced drain leakage and increased variability, which means that to obtain the same retention time we may be forced to increase the capacitance of the DRAM cell, exacerbating the problem of cell geometry and reliability of the dielectric material.

Due to the challenges from shrinking dimensions and variability, future DRAM cells will be expected to have much higher rate of faulty cells than current designs. To assist DRAM scaling, cost effective solutions must be developed to tolerate such high rate of faulty cells, otherwise it may become prohibitive to scale DRAM to smaller nodes.

Unfortunately, the exact data about error-rates in DRAM memories tend to be proprietary information and is guarded closely by DRAM manufactures. So, in our studies we assume that error-rates exceed significantly than what are handled by traditional techniques. We will also assume that these errors are persistent, and that they are distributed randomly across the chips. In this paper, we target a bit error-rate in the regime of 100 parts per million (ppm), or equivalently 10^{-4}.

2.2 Existing DRAM Repair Schemes

Current DRAM chips tolerate faulty cells by employing row sparing and column sparing. These mechanisms tend to mask the faulty cell at a large granularity. For example, with row sparing, the entire DRAM row containing the faulty cell gets decommissioned and replaced by a spare row. Given that DRAM rows contain in the regime of 10K-100K bits, masking each faulty cell incurs a significant overhead. Further-more disabling the faulty row and enabling the spare row must be done at design time, hence it must rely on non-volatile memory. Typically laser fuses are used to disable the row with faulty cell, and enable the spare row for the given row address, as shown in Figure 3 (derived from [7]). To handle a memory array containing few thousand rows, each spare row requires fuse memory of few tens of bits. Unfortunately, each bit of laser fuse incurs an area equivalent few tens of thousands of DRAM cells [8]. Thus, sparing incurs an overhead of approximately several hundred thousand DRAM cells to fix one faulty cell. While this overhead may be acceptable at very small error-rate, it is prohibitive to tolerate error-rates in the regime of several parts per million.

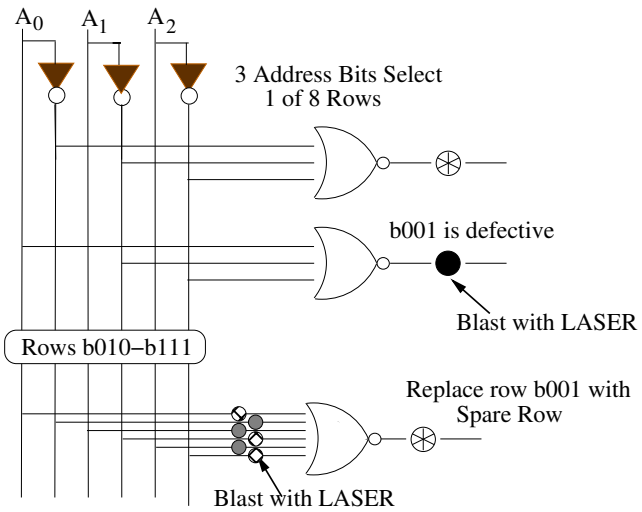

Figure 3: Typical row sparing design relies on laser fuses and sacrifices an entire row for masking a faulty cell.

2.3 Tolerating Faulty Cells with ECC DIMM

Instead of masking faulty cells, one can correct them using ECC. Commodity memory modules are typically also available in ECC enabled versions, in a (72,64) configuration. Such modules contain an extra ECC chip in addition to the eight data chips, and can correct up-to one error (and detect up-to two errors) in the 64-bit word. While the typical applications for ECC DIMM tend to be to tolerate soft errors, we can potentially use it to tolerate faulty DRAM cells as well. However, even with an ECC DIMM the error-rates that can be tolerated is low.

In our studies we consider an 8GB DIMM, containing one billion 8-byte words. The expected number of random errors that would result in a word with two errors can be computed using the Birthday Paradox analysis [9]. For example, if balls are randomly thrown into N buckets, on an average after $1.2\times\sqrt{N}$ throws, we can expect at-least one bucket to have more than one ball. Similarly, on average, a memory with 1 billion words would tolerate approximately 40K errors before getting a word with two errors. Thus, the error rate tolerated with ECC DIMM is 40K divided by the number of bits in memory (77 billion), or equivalently 0.5 ppm, approximately 200x lower than the error-rate we want to handle. Furthermore, such usage of ECC DIMM to tolerate faulty cells increases the vulnerability of the system to soft errors. Ideally, we want to tolerate faulty cells while retaining soft error protection of ECC DIMMs.

2.4 Need for Handling Multiple Faults/Word

A higher rate of faulty cells can be tolerated with the ECC approach if we correct multiple errors per word. To estimate the amount of multi-bit error protection required, we compute the expected number of words for a given number of faults. Let p be the probability of bit failure. Let there are b bits in the word. The expected number of faulty bits per word is $p \cdot b$. If $p \cdot b << 1$, then the probability (P_k) that the word has k errors ($k \geq 1$) can be approximated by Equation 1.

$$P_k = \frac{(p \cdot b)^k}{k!} \quad (1)$$

In our studies, we consider a traditional (72,64) ECC DIMM. So, the number of bits in the ECC word is 72. Table 1 shows the expected number of words in an 8GB memory that have 0, 1, 2, 3, and 4 or more errors for a probability of bit failure of 100 ppm. The episodes of 4 or more errors are rare, but we need to tolerate three faulty cells per word.

Table 1: Percentage of words with multiple faulty cells (and expected number of words in 8GB memory, i.e. 2^{30} words).

Num Faulty bits	0	1	2	3	4+
Probability	0.993	0.007	$26 \cdot 10^{-6}$	$62 \cdot 10^{-9}$	10^{-10}
Num words	0.99 Bln	7.7 Mln	28K	67	0.1

2.5 Low Cost Fault Handling by Exposing Faults

To handle 3-bits per word, the ECC overhead would be approximately 24 bits per word, or approximately 37%. Thus, the storage overhead of uniform fault tolerance is prohibitive at high error-rates. The problem with both row sparing and ECC schemes is that they try to hide the faulty cell information from the architecture, hence they incur significant storage overhead. To develop a cost-effective solution, we take inspiration from the fault tolerant architecture typically used in Solid State Drives (SSD) [10]. SSD are made of Flash technology, that tends to have high error-rates. The management layer in SSD keeps track of bad blocks and redirects access to good location. A similar approach can also allow DRAM systems to tolerate high error-rates.

From Table 1 we see that only a small fraction of words have more than 1 faulty cell. If we can expose the information about faulty cells to the architecture layer, then we can tolerate faulty words by decommissioning and redirecting at a word granularity and thus significantly reduce the storage overhead of tolerating faulty cells. Note that we cannot arbitrarily disable words in memory, as

the operating system relies on having a contiguous address space. We propose the *ArchShield* framework that can efficiently tolerate high rate of faulty cells, provides contiguous address space to the Operating System (OS), does not require changes to the existing ECC DIMMs, while still retaining soft error tolerance.

3. ARCHSHIELD FRAMEWORK

ArchShield leverages existing ECC DIMMs and enables them to tolerate high-rate of faulty DRAM cells. Figure 4 shows an overview of ArchShield. ArchShield divides the memory into two regions: one that is visible to the OS, and the other reserved for handling faulty cells. Thus, the OS is provided with a contiguous address space, even though this space may have faulty cells. ArchShield contains two data structures: Fault Map (FM) and Replication Area (RA). The Fault Map contains information about the number of faulty cells in the word. ArchShield employs *Selective Word Level Replication (SWLR)*, whereby only faulty words are replicated in the Replication Area. On a memory access, ArchShield obtains the Fault Map information associated with the line. If the line contains word with faulty cells, it is repaired with the replicas from the Replication Area.

For implementing ArchShield several challenges must be addressed. For example, having Fault Map entry for every word incurs high overhead. Similarly, accessing Fault Map from memory on every access incurs high latency. Also, the replication area must be architected to reduce the storage and latency overhead associated with obtaining replicas. Ideally, we want almost all of the memory address available for demand usage (visible to OS), and we want to keep the performance penalties associated with Fault Map access and Replication Area to be small, while retaining soft error protection.

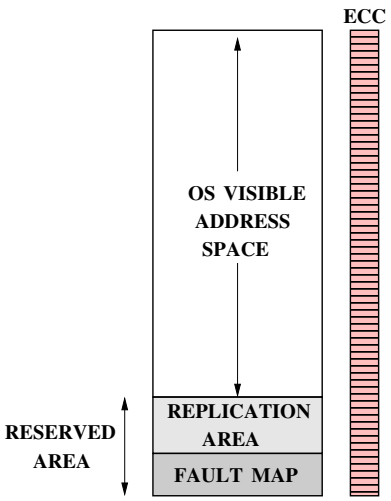

Figure 4: Overview of ArchShield (Figure not to scale)

3.1 Testing for Identifying Faulty Cells

ArchShield relies on having the location of faulty cells available. If the error-rate was small, then this information can be supplied by the manufacturer using some non-volatile memory on the DRAM module. Unfortunately, this method does not scale well to high error rates, as it incurs high storage overhead and cost (especially if the non volatile memory is employed with laser fuses as done with row sparing). So, for tolerating high error-rates, we advocate runtime testing. We assume there is a Built-In Self Test (BIST) controller present in the system that performs testing on the memory module when the module is first configured in the system. Testing can be done by writing a small number of patterns (such as "all ones" and "all zeros") as done in [11, 12] or by using well-known testing algorithms such as MARCH-B, MARCH-SS, GALPAT, and pseudo random algorithms for testing Active Neighborhood Pattern Sensitive Faults (ANPSFs) [13, 14].

As ECC protection exists at the word granularity, testing is also performed at word granularity. During the testing phase, the words are classified into three categories: Words with no faulty cells (NFC), Words with single faulty cell (SFC), words with multiple faulty cells (MFC). We assume that testing is able to identify all faulty cells,[1] and the Fault Map and Reserved Area are populated with the results of testing.

3.2 Architecting Efficient Fault-Map

ArchShield makes a separation between words with single faulty cell (SFC) and multiple faulty cells (MFC) as words with SFC can be handled with ECC in the absence of soft error. Thus, the Fault Map entry for each word must provide a tertiary value: NFC, SFC, or MFC. If we keep 2-bits per 64-bit word, this would result in a storage overhead of 1/32 of the entire memory. Furthermore, there may be faulty cells in the Fault Map as well, so additional redundancy would make the storage overhead of Fault Map prohibitive.

3.2.1 Line Level Fault Map

We reduce the storage overhead of Fault Map by exploiting the observation that memory is typically accessed at a cache line granularity (64 bytes). So, we can keep the information about faulty words at the cache line granularity as well. To ensure correctness, the fault level of all the words in the line is determined by the word with the most number of errors. If the line contains no faulty cell, it will be classified to be an NFC line. If the line contains at-least one SFC word, but no MFC word, the line is classified as an SFC line. Whereas, if the line has a MFC word, the line is classified as an MFC line.

As the line contains eight words, the probability of SFC line is approximately 8x higher than SFC word, increasing from 0.7% of words to 5.6% of the lines. Similarly, the probability that the line is classified as MFC line is increased by approximately 8x as well, increasing from 26ppm to 200ppm. The increase in SFC line does not impact performance significantly, as the replicated information is not accessed on a read (unless there is soft error). The dual read because of increase in MFC line is negligible to have any meaningful impact system performance, as it affects one out of 5000 accesses.

3.2.2 Fault Tolerance and Overhead of Fault Map

ArchShield assumes that the entire memory can contain faulty cells, including the area used to store the Fault Map. Therefore, we use redundancy in storing the Fault Map entry. Each Fault Map entry consists of 4-bits. If it is 0000, the line is deemed to have no faulty cells. If it is 1111, the line is deemed to have at-least one (or more) word with at-most one faulty cell. For any other combination, the line is conservatively deemed to be a MFC line. We store an MFC line as 1100 in the Fault Map.

[1]Given that ArchShield provides a protection of 1-bit soft error per word, it can tolerate a small probability of faults escaping the testing procedure. In particular, the system can tolerate one untested fault per word. A persistent soft error in the word can be notified to the Fault Map.

An error in Fault Map results in reading the replicated version of the word. The Fault Map area is also protected by ECC, so on any detected (or corrected) fault, the design conservatively tries to read from the replicated region. With 4-bits per 64-byte line, the storage overhead of Fault Map would be 1/128 of the entire memory, or equivalently 64MB for a 8GB DIMM. The address of the Fault Map entry can be obtained by simply adding the line address to the Fault Map Start Address (which is kept in a register of ArchShield).

3.2.3 Caching Fault Map Entries for Low Latency

The Fault Map must be consulted on each memory access. A naive implementation of probing Fault Map in main memory on every memory access would result in high performance overhead. So, we recommend caching the Fault Map entries in the on-chip cache, on a demand basis. Each Fault Map access can bring in a cache line worth of Fault Map information and cache it in the Last Level Cache (LLC). Given each Fault Map entry is only 4-bits, each cache line of Fault Map contains Fault Map information for 128 lines, resulting in high spatial and temporal locality. Our analysis shows that the Fault Map hit rate in the on-chip LLC to be in the regime of 95% on average, thus significantly reducing the memory accesses for Fault Map and associated performance penalties.

3.3 Architecting Replication Area

The Replication Area stores a replica for all the words with a faulty cell. The Fault Map only identifies if the line has a word with faulty cell, it does not identify the location of the replicated copy of this word. Therefore, the Replication Area must also contain a tag entry associated with each word. The tag size depends on the ratio of Replication Area to Memory size. To tolerate a BER of 10^{-4}, the Replication Area needs to store 7.74 million faulty words for an 8GB DIMM. If we could configure the Replication Area as a fully associative structure, we would need only 7.74 million entries, incurring about 1% of memory capacity. Unfortunately, this configuration would incur unacceptably high latency overheads. Replication Area is provisioned to be $\frac{1}{64}$th of main memory for BER of 10^{-4}. So we have 6 bits for line address, 3 bits for word in line, 1 valid bit and 2 overflow bits (replicated) for every entry, hence we get 1.5 bytes for tag. Thus, each entry in the replication region would be 9.5 bytes (1.5 bytes for tag and 8 bytes for data). This section identifies the appropriate structure for Replication Area to reduce latency while keeping the storage overhead manageable.

3.3.1 A Set Associative Structure

We want the interaction between the memory and the memory controller to be at a cache line granularity. Therefore, even the memory of the Replication Area can be accessed at a cache line granularity. Given that the cache line is 64 bytes, and each Replication Area entry is 9.5 bytes (1.5 bytes tag + 8 bytes data), we can store six entries in each line of 64 bytes, and have seven bytes of unused storage, as shown in Figure 5.

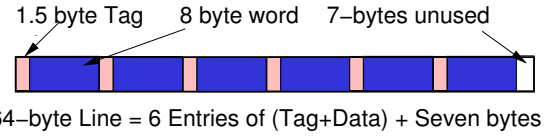

Figure 5: A 64-byte line configured as one set in the replication region. It can hold six entries and have seven bytes unused.

Given that we can hold six entries in each line, we can configure the Replication Area as a 6-way set associative structure. If the access across sets was uniform we would need only 1.3 million sets (7.74 Million divided by six). Unfortunately, as errors are spread randomly throughout the memory space, the allocation of this structure is non-uniform. We want to avoid the overflow of any of the set, as it would mean that we are unable to accommodate all faulty cells, and that module may be deemed unusable.

We can reduce the probability of overflow by increasing the number of sets. However, even with 2 million sets, approximately 10% sets overflow. Our analysis shows that to avoid the overflow of any set, the total number of sets must be increased by 12x compared to the minimal configuration. This incurs a storage overhead of approximately 15% of memory, making such design unappealing.

3.3.2 Efficiently Handling Overflow of Sets

Given that the overflow of the set associative structure are infrequent, we can tolerate these with a flexible organization that handles overflows in the set associative structure. We provide the set associative structure with a victim-cache like structure. Each group of 16-sets is provisioned with a 16 additional overflow sets. The 7-bytes unused in each set is used to link to one of the entries in the overflow region. The location of the overflow set can be identified with 4-bits and coupled with a valid bit, the pointer to overflow sets would take 5-bits. We use triple modulo redundancy on the pointer for fault tolerance. We call such a structure of 16 sets + 16 overflow sets as a *Replication Area group*, or simply RAgroup. Figure 6 shows the overview of RAgroup.

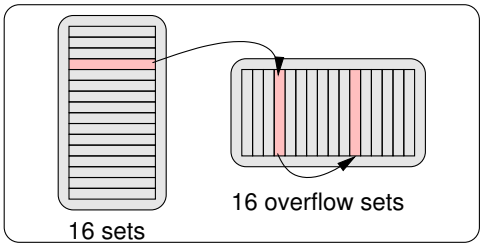

Figure 6: An RAgroup with 16-sets and 16 overflow sets. An overflow set can overflow into another set of same RAgroup.

Note that even though there is linkage between the normal sets and overflow sets, this does not impact the deterministic latency of existing memory interfaces. We first access the normal sets in the group. If no words for the given line is present, and there is a link to the overflow sets, then we send another memory request for obtaining the overflow set. Thus, our proposed structure can be easily incorporated in existing (deterministic) memory controllers.

Given that the normal sets occupy a storage of 1KB and the overflow sets also occupy the a storage of 1KB, the entire RAgroup can be in the same row buffer, as long as the row buffer size is 2KB or more. Thus, the access to overflow set is guaranteed to get a row buffer hit, reducing the access latency. To handle 7.75 million faulty words, we use 128K RAgroups (each with 16-set + 16 overflow sets). As each RAgroup incurs a storage overhead of 2KB, our proposed structure for the Replication Area incurs a storage overhead of 256MB.

Figure 7 shows the probability that this structure will not be able to handle a given number of random errors, for different value of overflow sets in the group. We perform this analysis using Monte-Carlo simulation, and repeating it 100K times. Even in 100K simulations, the structure with 16 overflow sets was unable to handle 8 Million errors only once. Thus, the structure has low variance which means the probability of deeming the DIMM unusable is negligible (10ppm).

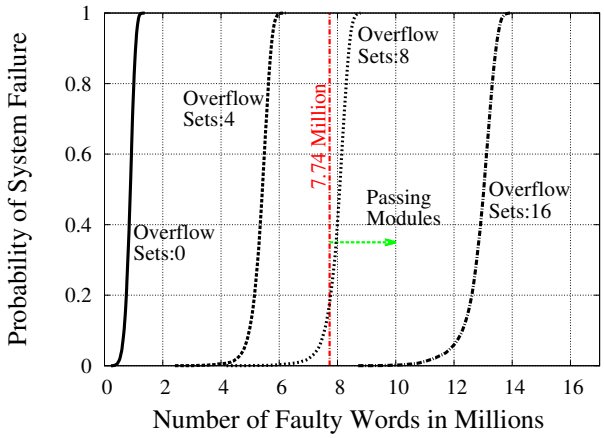

Figure 7: Probability that a structure is unable to handle given number of errors (in million). We recommend the structure with 16 overflow sets to tolerate 7.74 million errors in DIMM.

3.4 ArchShield Operation: Reads and Writes

ArchShield extends the memory controller to do read and write operations appropriately. On a read request that misses in the LLC, the request is sent to memory. In parallel, the address for the Fault Map entry is computed and the LLC is probed with the Fault Map address. In case there is a LLC hit for the Fault Map address (common case), the Fault Map entry is retrieved. Otherwise, another request is sent to memory to obtain the line containing the Fault Map (an uncommon case) and is installed in the LLC. If the Fault Map entry shows that the line does not have any faulty cell, we can use the data supplied from the main memory. If the line is deemed to have single faulty cell words, and ECC operation on the line does not result in uncorrectable error, we do not read the replicated copy. However, if there is one bit soft error and the ECC operation results in uncorrectable error, the replicated copy is read, thus providing soft error protection. If the line is deemed to have a word with multiple faulty cells, then the replicated copy is read and the matching words are incorporated in the line. Thus, accessing a line with multiple faults causes extra latency, however this is a rare event. For an error-rate of 10^{-4}, extra read is performed for less than one in few thousand read operations.

We add a bit called *Replication bit (R-bit)* to the tag-store entry in each line of the LLC to mark if the line requires replication on writeback. If, on the demand read, the line was determined to have a single faulty cell or multiple faulty cells the R-bit is set. A write to two locations (a good location and the replicated location) in case of word with single fault ensures that soft errors can be corrected by reading the copy from the Replication Area.

When a dirty line is evicted from the cache, and the R-bit is not set, writeback is done in normal manner. However, if the R-bit is set, we also need to update the replicated region. After the normal write is performed, the memory controller probes the replicated area for obtaining the set containing the replicated words for the given line. It then updates the data value for the corresponding words of the line, and updates the replicated region. Thus, while the Fault Map is cached in LLC, the replicated region is updated by the memory controller on a demand basis, and is not cached. Also note that the latency for doing the multiple writes is not in the critical path, however the extra operations can cause contention and thus impact performance indirectly. For an error-rate of 10^{-4}, 5.6% of the memory lines will require extra write operations.

Figure 8 shows the flowchart for servicing memory requests with ArchShield. The performance impact of ArchShield is determined by the hit-rate for the Fault-Map entries. As the hit rate is high (approximately 95%), for a read request ArchShield performs only one memory access in the common case.

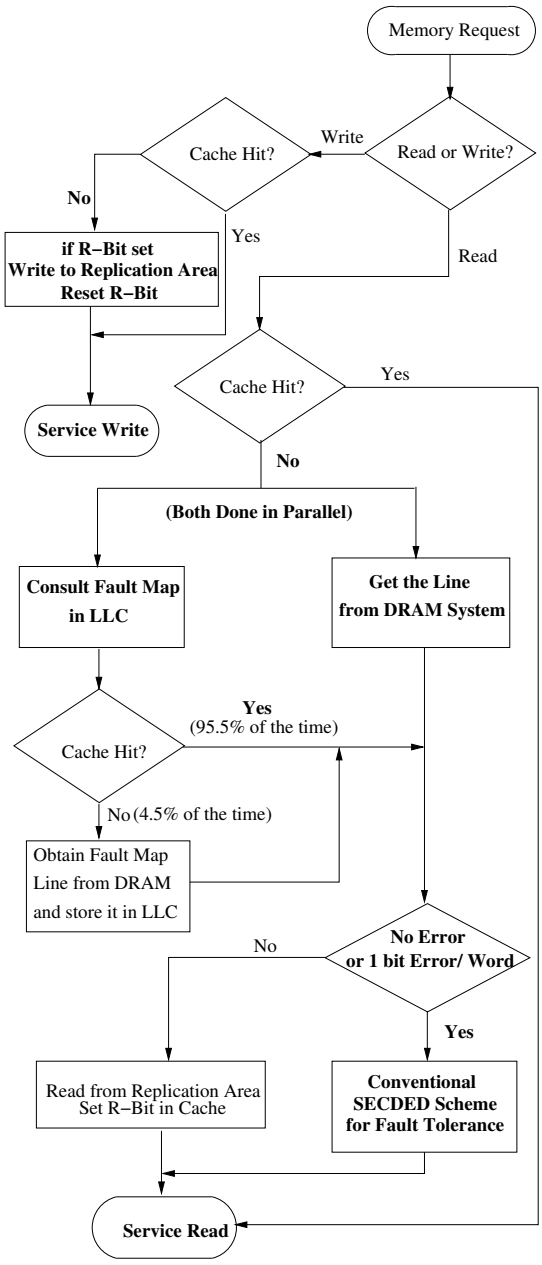

Figure 8: A Flowchart of the read and write operations in ArchShield. The decisions in 'Bold' words indicate the most frequent path for requests in case of a LLC miss

Our proposed implementation assumes an R-bit for each cache line. If the cache does not support this, we can still implement ArchShield by making dirty evictions from the cache probe the Fault Map in memory in order to determine if dual writes must be performed. Similarly, Fault Map requires 4-bit per line (64MB for 8GB chip). This structure is designed to handle high BER. When the BER is low, an alternative implementation (such as Bloom filters and lookup-tables) can be used to reduce the storage overhead.

3.5 ArchShield: Tying it All Together

Figure 9 shows a memory system with ArchShield. The main memory consists of traditional ECC DIMMs and does not require any changes. The memory space is divided into addressable space, Replicated Area and Fault Map. The memory controller is extended to compute the address of the Fault Map entry, check that entry in the LLC, and in cases of an LLC miss for the Fault Map, read the required line with Fault Map information and cache it in the LLC. On an LLC read miss, the memory controller obtains the Fault Map entry, and determines if a second read from the replicated region is required. If so, it reads the replicated region and repairs the line with replicated words. In case of an LLC writeback, the memory controller determines if the replicated region must be updated. If so, the extra write operations are performed. This check for replicated writeback is assisted by the R-bit in the LLC. Thus, ArchShield requires changes to the memory controller and minor changes to the cache structure (to add the R-bit to the tag store entry).

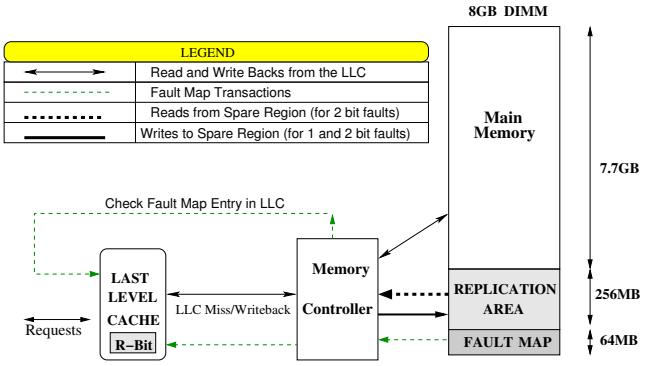

Figure 9: Memory System with ArchShield

The data-structures for ArchShield are kept in main memory. For 8GB memory, the Fault Map requires 64MB storage, and the Replication Area requires 256MB storage, for a total storage overhead of 320MB. Thus, ArchShield provides remaining 7.7GB (or 96% of the 8GB memory) available as visible address space.

4. EXPERIMENTAL METHODOLOGY

4.1 Configuration

We use an in-house memory system simulator for our studies. The baseline configuration is described in Table 2. There are 8 cores sharing an 8MB LLC. The memory system contains two channels, each with one 8GB DIMM. We perform virtual to physical translation using a first touch policy, with 4KB page size. The Fault Map entries are cached on a demand basis and evicted using LRU replacement of LLC. We assume an error-rate of 10^{-4}, and that faulty cells are spread randomly across the memory space. For accessing replicated region, we add extra 3 DRAM cycles for parsing the tag-store, and one additional DRAM cycle for access to overflow set.

4.2 Workloads

We use a representative slice [15] of 1 billion instructions for each benchmark from the SPEC2006 suite. We perform evaluations by executing the benchmark in rate mode, where all the eight cores execute the same benchmark. Table 3 shows the characterization of the workloads used in our study. The Read and Write MPKI of these workloads indicate their memory activity. Workload footprint is computed by the number of unique (4KB) pages touched by the workload. As we use 8 copies of the benchmark, the total footprint is increased by 8x. We perform timing simulation till all the benchmarks in the workload finish execution, and measure the execution time as the average execution time of all 8 cores.

Table 2: Baseline System Configuration

Processors	
Number of cores	8
Processor clock speed	3.2 GHz
Last Level Cache	
L3 (shared)	8MB
Associativity	8 way
Latency	24 cycles
Cache line size	64Bytes
DRAM 2x8GB/channel-DDR3	
Memory bus speed	800MHz (DDR3 1.6GHz)
Memory channels	2
DIMM capacity per channel	8GB
Ranks per channel	2
Banks per rank	8
Row Buffer Size	8KB (DIMM)
Bus width	64 bits per channel
t_{CAS}-t_{RCD}-t_{RP}-t_{RAS}	9-9-9-36

Table 3: Benchmark Characteristics (Rate Mode)

Class	Workload	Read MPKI	Write MPKI	Footprint
High MPKI	mcf	74.24	12.75	10.5 GB
	lbm	31.89	23.9	3.2 GB
	soplex	26.98	5.35	2 GB
	milc	25.75	9.68	4.5 GB
	libquantum	25.42	2.71	256 MB
	omnetpp	20.8	8.22	1.1 GB
	bwaves	18.71	1.45	1.5 GB
	gcc	16.62	9.29	682 MB
	sphinx	12.33	1.06	139 MB
	GemsFDTD	9.79	4.96	5.4 GB
	leslie3d	7.55	2.25	619 MB
	wrf	6.68	2.39	492 MB
	cactusADM	5.29	1.54	1.27 GB
	zeusmp	4.79	1.75	1.5 GB
	bzip2	3.63	1.26	2.47 GB
	dealII	2.98	0.39	52 MB
	xalancbmk	2.21	1.61	1.4 GB
Low MPKI	hmmer	0.94	0.91	16 MB
	perlbench	0.85	0.14	185 MB
	h264ref	0.71	0.39	66 MB
	astar	0.64	0.44	22 MB
	gromacs	0.59	0.19	60 MB
	gobmk	0.42	0.23	148 MB
	sjeng	0.39	0.31	1.3 GB
	namd	0.07	0.02	42 MB
	tonto	0.07	0.02	15 MB
	calculix	0.02	0.001	16 MB
	gamess	0.016	0.004	10 MB
	povray	0.01	0	11 MB

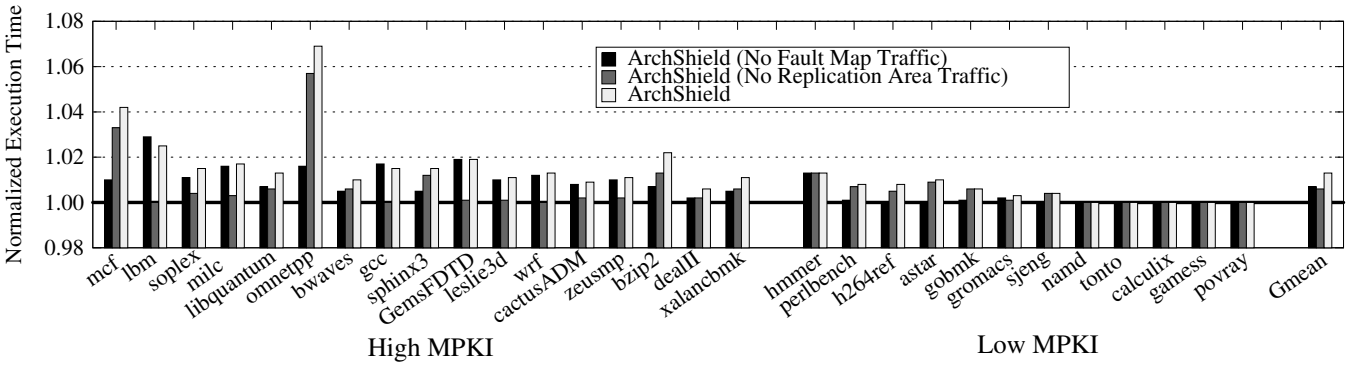

Figure 10: Impact on Execution Time for three ArchShield configurations: 1. Ideal Fault Map, 2. No extra writes, 3. Realistic

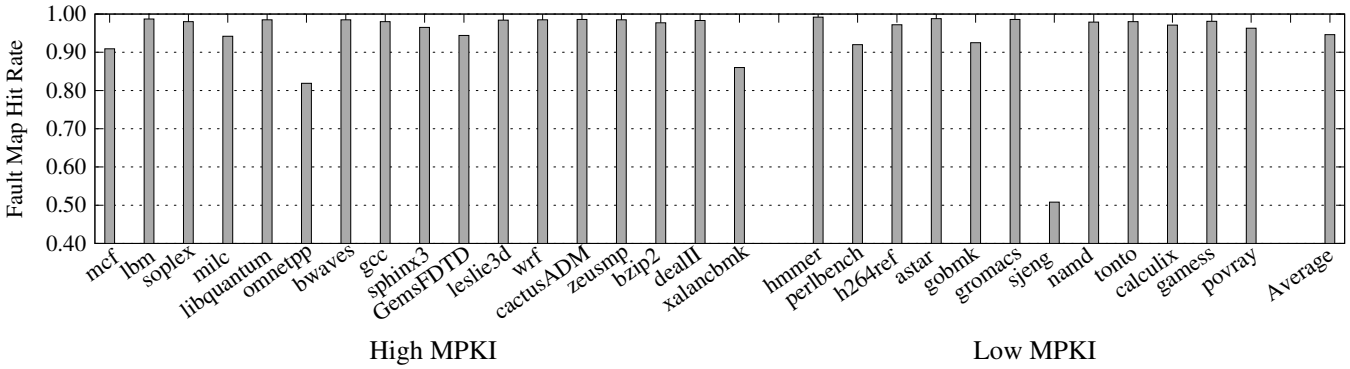

Figure 11: Fault Map Hit Rate in Last Level Cache

5. RESULTS

5.1 Impact on Execution Time

ArchShield has two sources of performance overhead. One is caching of the Fault Map. A read operation for a line from main memory will not complete until the Fault Map entry is available. So, Fault Map miss in the the LLC causes increase in the read latency. The other is the extra traffic due to updates to the Replication Area. To, better understand the performance implications from these two factors, we conducted experiments with three ArchShield configurations. First, we assumed an ideal Fault Map (which does not consume LLC area or memory traffic). Second, we assume that the extra traffic for the Replication Area is ignored. Third one is ArchShield with realistic Fault Map and Replication Area.

Figure 10 shows the execution time of the three ArchShield configurations. The execution time is normalized to the baseline with fault-free memory. The bar labeled Gmean shows the geometric mean over all the workloads. On average, ArchShield causes an execution time increase of 1%.[2] The Fault Map and Replication Area are each responsible for approximately half of the performance loss. However, the impact depends on the workloads. For several workloads the performance loss is primarily because of extra traffic to the Replication Area. For *omnetpp*, the performance loss is due to non-ideal Fault Map.

[2]In our analysis we have assumed that the performance loss due to the unavailable memory capacity (4%) is negligible, which is accurate given the footprint of our workload. However, for workloads with larger footprints there may be a minor (negligible) performance loss due to reduced capacity.

5.2 Fault Map Hit Rate Analysis

The locality of the Fault Map is central to efficient operation of ArchShield. Given that each line of Fault Map contains information about 128 contiguous lines, we expect high spatial and temporal locality for the Fault Map line in the LLC. Figure 11 shows the hit rate of the LLC for Fault Map accesses. On average, the Fault Map hit rate for LLC is 94%.

For benchmarks that have high MPKI, the Fault Map hit rate is reduced. This happens because the cache is contended for both the demand lines as well as the lines from the Fault Map. For example, *omnetpp* has a Read MPKI of 20.8, and FM hit rate of 82%, hence it has the highest performance degradation with ArchShield. Other high MPKI workloads such as *mcf* and *xalancbmk* show similar behavior. For *sjeng*, the low hit rate of the Fault Map does not impact performance because it has very low MPKI, hence the system performance is not sensitive to memory performance. Overall, the Fault Map caching for ArchShield is quite effective as only three benchmarks out of 29 show a FM hit rate of less than 90%,

We also analyzed the occupancy of Fault Map entries in the LLC. We found that, on average, 6% of the LLC contains lines from the Fault Map. Thus, the spatial locality of Fault Map entries helps the Fault Map to get high hit rate without occupying significant area in the LLC. Note that, when we perform cache replacement in the LLC we do not differentiate between lines from the main memory and lines from the Fault Map. So, even a simple demand-based caching policy for the Fault Map works quite well.

5.3 Analysis of Memory Traffic

In addition to the normal memory traffic from LLC misses and writebacks, ArchShield increases the memory traffic due to extra

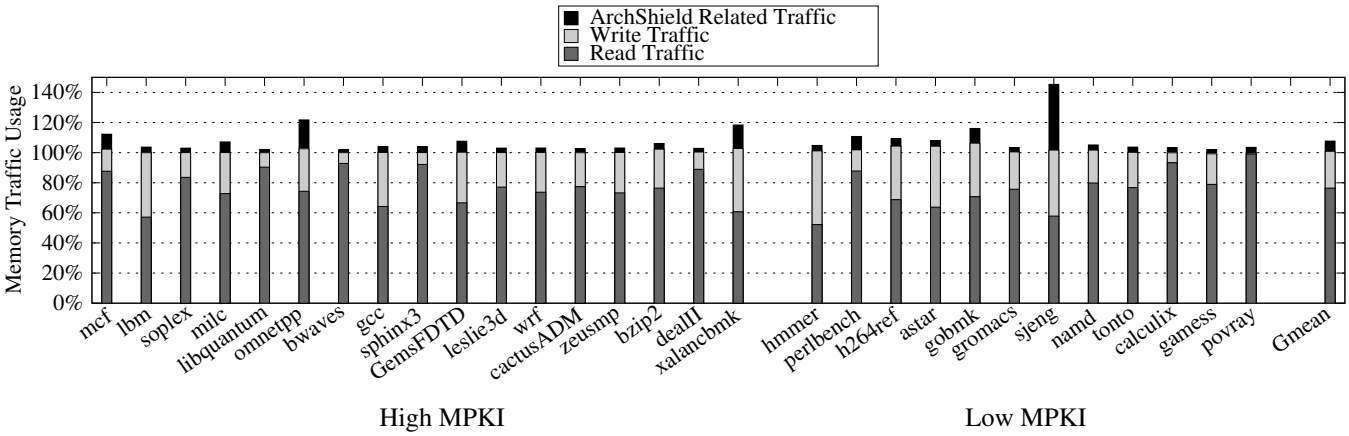

Figure 12: Memory Traffic Breakdown with ArchShield

activity. In particular, the memory traffic is increased because of Fault Map misses in the LLC and the extra writes to the Replication Area for the faulty lines. Furthermore, caching the Fault Map entries in the LLC may increase the LLC miss rate and writebacks for the demand accesses.

To capture the impact of ArchShield on memory traffic we divide the total memory traffic into three components. The read traffic emanating from LLC misses, the writebacks from LLC, and the traffic related to ArchShield (Fault Map and extra writes). Figure 12 shows the breakdown of these three components. The total memory traffic is normalized to the memory traffic with the fault-free memory.

The traffic due to ArchShield shows a negative correlation with Fault Map hit rate. The benchmark *sjeng* has the highest traffic overhead due to ArchShield of around 35%. This happens because of low hit rate of the Fault Map. However, as this benchmark has low MPKI, the impact on performance is insignificant. For *astar*, the traffic due to demand accesses is higher compared to the baseline because of extra LLC misses and writebacks due to caching of Fault Map entries.

Due to the replication of lines with fault cells, we can expect the writeback traffic to increase by 5.6%, as 5.6% of the lines are expected to have a faulty cell. On average, ArchShield increases the total memory traffic by 6%.

5.4 Analysis of Memory Operations

For lines with multiple faults, ArchShield requires that multiple accesses be done on a read: one to the normal location and the other to the Replication Area. The access to the Replication Area can itself result in multiple accesses, if the set in the Replication Area overflows to another set. However, this happens rarely. Table 4 shows the breakdown of memory operations in terms of number of accesses to memory. We analyze three operations: a read operation due to LLC miss, a writeback from LLC and a Fault Map miss in the LLC. All numbers are relative to the total memory operations.

Table 4: Analysis of Memory Operations

Transaction	1 Access(%)	2 Access(%)	3 Access(%)
Reads	72.13	0.02	~0
Writes	22.07	1.18	0.05
Fault Map	4.55	N/A	N/A
Overall	98.75	1.2	0.05

On average, 72.15% of all memory accesses are read operations, out of which only 0.02% accesses require two memory accesses. Thus, almost all read operations get satisfied with single access. Writebacks account for 23.3% of all memory operations on average. As we can expect 5.6% of lines to cause extra writes (due to replication), the number of writes that require two accesses are 5.6%*23.3%=1.18%. Only a negligible number of write operations require three accesses. On average, 4.55% of the memory operations are due to Fault Map miss, each of which get satisfied in one memory operation. Thus, ArchShield satisfies 98.75% of all memory operations with single memory access.

We also analyzed read latency for the baseline and ArchShield and found the change to be minor. ArchShield obtains an average read latency of 200 cycles and baseline 197 cycles. This 1.5% increase in the read latency causes the 1% reduction in performance.

5.5 Sensitivity of ArchShield to Bit Error-Rate

We have selected parameters for ArchShield to tolerate a bit error-rate of 10^{-4}. ArchShield can be tuned to handle a different error-rate. For example, to handle a bit error-rate of 10^{-5}, we can reduce the size of Replication Area by 8x, as we expect 10x fewer faulty cells. This reduces the storage overhead of ArchShield to 96MB, making 98.8% of memory capacity available for normal usage. Also, fewer faulty cells also reduces the traffic due to extra writes. The overall increase in execution time is 0.5%, instead of 1% at error-rate of 10^{-4}.

Conversely, to handle 2x higher error-rate (2×10^{-4}), the storage overhead would get doubled to 7%, making only 93% of memory capacity available for use. It will also cause higher performance degradation due to increased write traffic from replication, as 11% of the lines would require an extra write.

6. REDUCING DRAM REFRESH

Thus far, we have used ArchShield for tolerating only faulty cells due to manufacturing defects. However, ArchShield can also be leveraged for other DRAM optimizations as it can tolerate high error rates. For example, ArchShield can be used to reduce the refresh operations in DRAM systems.

Data is stored in DRAM cells by placing charge on each cell. DRAM cells are leaky and need periodic refresh to maintain data integrity. A refresh operation is performed by reading the row, precharging it and writing it back for all rows in the chip. DDRx DRAM chips follow JEDEC standards which mandate that all cells be refreshed within 64ms to prevent loss of data. As the memory

capacity increases, the total number of refresh operations increases as well. In fact, we are at a point at which refresh operations are going from negligible to significant. The memory throughput loss due to refresh is approximately 7% at 8Gb, it will increase to 14% at 16Gb, 28% at 32Gb, and more than 50% at 64Gb [12, 16]. Thus, the performance and power consumption of future DRAM systems will be severely limited by refresh operations.

Fortunately, only a small number of bits in the DRAM row have low retention time, and the average retention is in the range of few (tens) of seconds. We can leverage this information to develop efficient refresh mechanisms. Figure 13 shows the probability of bit failure as a function of retention time [17]. At a refresh rate of 256ms, approximately 1000 DRAM cells fail, whereas at 1s refresh interval, the probability of cell failure increases to 0.5×10^{-4}. For an 8GB ECC-DIMM 3.9 million out of 77 billion cells is expected to have retention failures if they are refreshed at 1s interval.

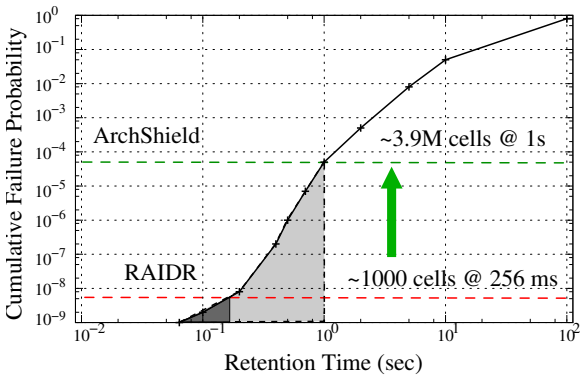

Figure 13: Exploiting retention characteristics of DRAM for efficient refresh. RAIDR operates at 4x longer refresh time interval. ArchShield can operate at 16x longer refresh interval.

The variation in retention time can be used to develop multi rate refresh algorithms, whereby rows containing cells with low retention time are refreshed at a higher rate whereas the rest of the memory is refreshed at a lower rate. Unfortunately, this scheme suffers from the key drawback that even if the row contains one cell with low retention time the entire row is subjected to faster refresh rate. A row buffer typically contains few tens of thousands of DRAM bits. Therefore, this technique inherently cannot tolerate a rate of more than 1 weak cell (cell with low retention time) every few tens of thousands of cells, otherwise almost the entire memory gets subjected to higher refresh rate, reducing the effectiveness of multi-rate refresh schemes.

Another challenge with multi-rate refresh algorithms is to store retention time information efficiently. Even if we have a bit associated with each row indicating if the row contains a weak cell or not, it will still consume an overhead of several megabits. A recent work called RAIDR [12] developed an efficient bloom filter implementation to reduce the storage overhead of storing retention time of different rows. However, RAIDR still suffers from the inherent problem of all multi-rate refresh algorithms, in that, even a single weak cell in the DRAM row subjects the entire row (8KB, 64K cells) to a normal refresh rate of 64ms. The problem is worsened by false positives from bloom filter, when the rate of weak cell is increased. Thus, RAIDR is effective only for a very small weak cell probability. The version proposed reduces the refresh rate to 256ms, handling only 1000 weak cells.

At the refresh interval of 1s, the bit error rate is $0.5 \cdot 10^{-4}$, so RAIDR is unable to tolerate such a high rate of weak cell and will lose almost all of its refresh savings. ArchShield, on the other hand, is architected for error rate of up-to 10^{-4}, which means we can use it to lower refresh rate in the regime of 1 second. To reduce refresh operations, ArchShield will need to do retention time profiling (similar to RAIDR) and then populate the Fault Map and Replication Area with the information about the weak cells. Then rather than having multiple rate of refresh, ArchShield can simply use uniform rate of refresh of 1 second and thus reduce the refresh related penalties by a factor of 16.

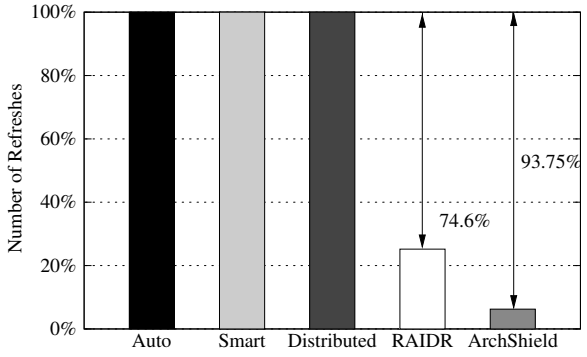

Figure 14: Effectiveness of different refresh saving schemes

Figure 14 compares the number of refresh operations performed per unit time (say 1 second) by different techniques. Auto refresh, Distributed Refresh, and Smart Refresh [18] have similar refresh rate in practice. RAIDR reduces the rate of refreshes by approximately 4x, whereas ArchShield reduces refresh operations by 16x. Thus, ArchShield is useful not only for tolerating faulty cells, but it can also be used effectively to optimize DRAM operations.

7. RELATED WORK

With reducing feature size, memory reliability has become a growing concern, and several recent research studies have looked at improving memory reliability. In this section, we compare the work most related to ours from the areas of DRAM reliability, error correction in Phase Change Memory (PCM), and enabling SRAM caches to operate at low voltage.

7.1 Multi-bit Error Correction in DRAM

We can tolerate a high error-rate by employing multi-bit error correction in DRAM memories. To tolerate an error-rate in the regime of 100ppm, we need three bit error correction, i.e. ECC-3 for each word (ECC-4 if we want soft error protection). Employing such high levels of error correction would require storage overhead of 37% of memory space. This would need the DIMM to have three extra ECC chips, resulting in prohibitive cost. It will also result in lower performance due to higher decode latency of ECC-4. Figure 15 shows the normalized execution time with ECC-4 decode latency of 15 cycles. On average, ECC-4 increases execution time by 5%, compared to 1% with ArchShield.

A recent work, Virtual and Flexible ECC (VFECC) [19], allows systems to implement high levels of ECC without relying on ECC DIMMs. It incorporates the ECC storage within the main memory. Unfortunately, VFECC does not reduce the storage overhead associated with high levels of error correction, as the ECC level is not dependent on the number of faults in the word. To implement ECC-3, VFECC would still need to dedicate about 37% of memory capacity, reducing the effective size of the 8GB DIMM to 5.6GB, making it unappealing for practical implementations.

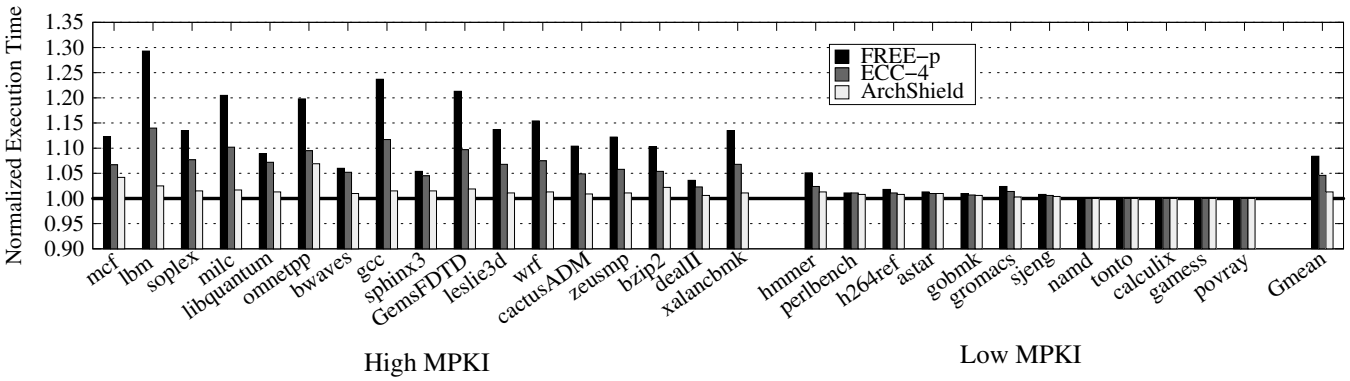

Figure 15: Execution time impact of different schemes. Providing ECC-4 per word incurs prohibitive storage overhead (37% memory capacity), whereas the read-before-write of FREE-p can cause significant performance degradation.

7.2 Error Correction in PCM

Several recent studies have looked at error correction in PCM memories. These solutions range from replicating pages with faulty cells [20], to correcting hard errors with pointers or data inversion [21, 22], to efficiently using non-uniform levels of error correcting pointers [23], to sparing lines with faulty cells with embedded pointer [24]. All of these schemes rely on non-traditional DIMMs, and have extra bits associated with each page or line. Whereas, we want a scheme that works well with existing DIMMs. The work that is most closely related to our proposal is FREE-p (Fine Grained Remapping with ECC and Embedded Pointers) [24].

FREE-p decommissions a line with faulty cells (more than what can be handled by the per-line ECC) and stores a pointer in the line to point to the spare location. It relies on the read-before-write characteristics of PCM memory to read the pointer before writing to the line (to avoid destroying the pointer). While this may be a reasonable assumption for PCM because of high write latency, such read-before-write operations cause significant performance degradation in DRAM memories. Figure 15 compares the performance of FREE-p with ArchShield. We implemented the *Baseline* FREE-p system. FREE-p causes 8% performance degradation on average (and sometimes as high as 29%, such as for lbm), whereas ArchShield causes negligible performance impact. Furthermore, FREE-p assumes a fault indicator bit with each line, which is not present in traditional DIMMs. The performance of FREE-p can be improved by caching the embedded pointer (using *pCache*, *pIndexCache*), however FREE-p would still incur the high decoding latency of multi-bit ECC. As such multi-bit ECC decoding delay is not present in ArchShield, it will avoid the associated performance penalties of multi-bit ECC.

7.3 Low-Voltage SRAM Caches Using ECC

Reducing the supply voltage of an SRAM cell increases the probability of the cell becoming erroneous. Several recent studies [**?**, 25–29] have looked at means to tolerate such errors, so as to enable large SRAM caches to operate at low voltages. However, the constraints for cache and memory are quite different. For example, the deterministic latency requirement for main memory makes it impractical to implement complex schemes [27, 28] that require accessing multiple banks in parallel and combining the results. Furthermore, the most recent work in this area [29] showed that even at very low voltages only a small percentage of lines have more than one-bit error, so low voltage operation can be enabled without significant performance degradation by simply discarding lines with multi-bit error. Unfortunately, discarding random lines from the address space is not a viable option for main memories. While disabling can be performed at a cache line granularity in SRAM caches, the OS must disable the entire page for faulty lines. Thus, even if 1% of the lines are deemed faulty, given that a typical page of 4KB contains 64 lines, such page-level disabling would cause most of the pages to be decommissioned. Thus, the constraints of deterministic latency and coarse-grained disabling make main memory reliability a more difficult problem than for SRAM caches.

7.4 Software Techniques for Reliability

Memory errors can be tolerated in software as well. For example, with memory page retirement [30, 31], the OS can retire a faulty page from the memory pool, once such a faulty page is detected. Unfortunately, these schemes operate at a coarse granularity of page size. Given that the typical page size is 4KB (32Kb), these schemes are unable to tolerate error-rates higher than one error for every several tens of thousand of bits. To operate at high error-rate, a fine grained approach such as at word-granularity or line-granularity is needed.

8. SUMMARY

Scaling of DRAM memories has been the prime enabler for higher capacity main memory system for the past several decades. However, we are at a point where scaling DRAM to smaller nodes has become quite challenging. If scaling is to continue, future memory systems may be subjected to much higher rate of errors than current DRAM systems. Traditional techniques such as row sparing or ECC DIMM do not tolerate high error-rates efficiently. Unfortunately, tolerating high error rates while concealing the information about faulty cells within the DRAM chips results in high overhead. To sustain DRAM scaling, efficient hardware solutions for tolerating high error-rates must be developed. To that end, this paper makes the following contributions:

1. We propose *ArchShield*, an architectural framework that exposes the information about faulty cells to the hardware. It uses a *Fault Map* to track lines with faulty cells, and employs *Selective Word Level Replication (SWLR)*, whereby only faulty words are replicated for fault tolerance.

2. We show that embedding the data structure of ArchShield in memory still provides most (96%) of the memory capacity available for normal usage, even at high error-rate.

3. We show that the performance degradation of ArchShield from extra traffic due to Fault Map and SWLR is only 1%. This is achieved by demand-based caching of Fault Map entries on processor chip, and by architecting the replication structure to reduce access latency.

4. We show that ArchShield can also be leveraged for reducing refresh operations in DRAM memories. ArchShield can reduce the effective refresh time of DRAM from 64ms to 1 second, thus reducing the refresh related overheads of latency and power by a factor of 16.

ArchShield can be implemented by making minor changes to the memory controller and the last level cache. ArchShield can be deployed with commodity DIMMs and does not requires any changes the existing memory interfaces. Similarly, ArchShield does not require any changes to the operating system, except for limited visibility to the address space. As system scale down to sub 20nm regime, we believe such cross-layer solutions for handling errors would become essential for future systems.

Acknowledgments

Thanks to Saibal Mukhopadhyay for discussions on DRAM scaling. Moinuddin Qureshi is supported by NetApp Faculty Fellowship and Intel Early Career Award.

9. REFERENCES

[1] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *ISCA-36*, 2009.

[2] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *ISCA-36*, 2009.

[3] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ISCA-36*, 2009.

[4] A. Allan. (2007, dec.) International technology roadmap for semiconductors (japan, ortc). [Online]. Available: http://www.itrs.net/Links/2007Winter/2007_Winter_Presentations/

[5] S. Hong, "Memory technology trend and future challenges," in *Electron Devices Meeting (IEDM), 2010 IEEE International*, dec. 2010, pp. 12.4.1 –12.4.4.

[6] K. Kim, "Future memory technology: challenges and opportunities," in *VLSI Technology, Systems and Applications, 2008. VLSI-TSA 2008. International Symposium on*, april 2008, pp. 5 –9.

[7] B. L. Jacob, S. W. Ng, and D. T. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2008.

[8] B. Gu, T. Coughlin, B. Maxwell, J. Griffith, J. Lee, J. Cordingley, S. Johnson, E. Karaginiannis, and J. Ehmann, "Challenges and future directions of laser fuse processing in memory repair," in *Proc. Semicon China*, 2003.

[9] E. McKinney, "Generalized birthday problem," *American Mathematical Monthly*, pp. 385–387, 1966.

[10] *TN-29-59: Bad Block Management in NAND Flash Memory*, Micron, 2011.

[11] R. Venkatesan, S. Herr, and E. Rotenberg, "Retention-aware placement in dram (rapid):software methods for quasi-non-volatile dram," in *HPCA-12*, 2006.

[12] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "Raidr: Retention-aware intelligent dram refresh," in *ISCA-39*, 2012.

[13] A. J. van de Goor, *Testing Semiconductor Memories: Theory and Practice*. John Wiley & Sons, Inc.

[14] N. K. Jha and S. Gupta, *Testing of Digital Systems*. Cambridge Univ. Press.

[15] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for accurate and efficient simulation," *ACM SIGMETRICS Performance Evaluation Review*, 2003.

[16] J. Stuecheli, D. Kaseridis, H. C.Hunter, and L. K. John, "Elastic refresh: Techniques to mitigate refresh penalties in high density memory," in *MICRO-43*, 2010, pp. 375–384.

[17] K. Kim and J. Lee, "A new investigation of data retention time in truly nanoscaled drams," *Electron Device Letters, IEEE*, vol. 30, no. 8, pp. 846 –848, aug. 2009.

[18] M. Ghosh and H.-H. S. Lee, "Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3d die-stacked drams," in *MICRO-40*, 2007.

[19] D. H. Yoon and M. Erez, "Virtualized and flexible ecc for main memory," in *ASPLOS-15*, 2010.

[20] E. Ipek, J. Condit, E. B. Nightingale, D. Burger, and T. Moscibroda, "Dynamically replicated memory: building reliable systems from nanoscale resistive memories," in *ASPLOS-15*, 2010.

[21] S. Schechter, G. H. Loh, K. Straus, and D. Burger, "Use ecp, not ecc, for hard failures in resistive memories," in *ISCA-37*, 2010.

[22] N. H. Seong, D. H. Woo, V. Srinivasan, J. Rivers, and H.-H. Lee, "Safer: Stuck-at-fault error recovery for memories," in *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, 2010, pp. 115–124.

[23] M. K. Qureshi, "Pay-As-You-Go: Low Overhead Hard-Error Correction for Phase Change Memories," in *MICRO-44*.

[24] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. Jouppi, and M. Erez, "FREE-p: Protecting non-volatile memory against both hard and soft errors," in *HPCA-2011*.

[25] D. Roberts, N. Kim, and T. Mudge, "On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology," in *Digital System Design Architectures, Methods and Tools, pp. 570-578, Aug. 2007*.

[26] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off cache capacity for reliability to enable low voltage operation," in *ISCA-2008*.

[27] A. Ansari, S. Gupta, S. Feng, and S. Mahlke, "Zerehcache: Armoring cache architectures in high defect density technologies," in *MICRO-2009*.

[28] A. Ansari, S. Feng, S. Gupta, and S. Mahlke, "Archipelago: A polymorphic cache design for enabling robust near-threshold operation," in *HPCA-2011*, feb. 2011.

[29] A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu, "Energy-efficient cache design using variable-strength error correcting codes," in *ISCA-2011*.

[30] C. Slayman, M. Ma, and S. Lindley, "Impact of error correction code and dynamic memory reconfiguration on high-reliability/low-cost server memory," in *Integrated Reliability Workshop*, 16 2006-sept. 19 2006.

[31] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic rays don't strike twice: understanding the nature of dram errors and the implications for system design," in *ASPLOS-17*.