Reliability-Aware Data Placement for Heterogeneous Memory Architecture

Manish Gupta[1] Vilas Sridharan[2] David Roberts[3] Andreas Prodromou[1]
Ashish Venkat[1] Dean Tullsen[1] Rajesh Gupta[1]
[1]CSE Dept.
University of California San Diego
{manishg, aprodrom, asvenkat, tullsen, rgupta}@cs.ucsd.edu
[2]RAS Architecture [3]AMD Research
Advanced Micro Devices, Inc.
{Vilas.Sridharan,David.Roberts}@amd.com

ABSTRACT

System reliability is a first-class concern as technology continues to shrink, resulting in increased vulnerability to traditional sources of errors such as single event upsets. By tracking access counts and the Architectural Vulnerability Factor (AVF), application data can be partitioned into groups based on how frequently it is accessed (its "hotness") and its likelihood to cause program execution error (its "risk"). This is particularly useful for memory systems which exhibit heterogeneity in their performance and reliability such as Heterogeneous Memory Architectures – with a typical configuration combining slow, highly reliable memory with faster, less reliable memory. This work demonstrates that current state of the art, performance-focused data placement techniques affect reliability adversely. It shows that page risk is not necessarily correlated with its hotness; this makes it possible to identify pages that are both hot and low risk, enabling page placement strategies that can find a good balance of performance and reliability.

This work explores heuristics to identify and monitor both hotness and risk at run-time, and further proposes static, dynamic, and program annotation-based reliability-aware data placement techniques. This enables an architect to choose among available memories with diverse performance and reliability characteristics. The proposed heuristic-based reliability-aware data placement improves reliability by a factor of 1.6x compared to performance-focused static placement while limiting the performance degradation to 1%. A dynamic reliability-aware migration scheme, which does not require prior knowledge about the application, improves reliability by a factor of 1.5x on average while limiting the performance loss to 4.9%. Finally, program annotation-based data placement improves the reliability by 1.3x at a performance cost of 1.1%.

1. INTRODUCTION

Transient faults due to single event upsets (SEUs) are of critical concern in the reliability of computer systems [5, 15, 41]. Technology scaling and reduced operating voltages have increased the susceptibility of semiconductor devices to SEUs [18, 67]. These faults can result in spurious bit flips that can corrupt the architectural state leading to catastrophic system crashes [3, 47], unexpected loss of data, and critical security vulnerabilities and exposures [14, 36, 66].

Prior research has proposed the use of error correction codes (ECC) to handle such errors for conventional DDRx memory [10,21]. ECC has enabled efficient technology scaling for conventional DDRx memory, packing more bits per unit area while maintaining a sustainable rate of uncorrectable errors [52,59,60]. Despite advances in the capacity of DDRx memory, the ever-evolving bandwidth requirements of emerging manycore CPU, GPU, and accelerator architectures have prompted architects to explore alternative memory technologies and ways to organize memory/compute systems [65].

The 3D die-stacking technology has enabled the stacking of DRAM chips directly on compute units using Through-Silicon Vias (TSVs) [4, 11, 27, 29, 39, 57]. These die-stacked memory organizations provide 4X-8X higher bandwidth than conventional DDRx memory [7, 9]. However, die-stacked memory is limited in terms of capacity, higher cost, and higher power dissipation [17]. In order to obtain the dual benefits of high bandwidth and greater capacity, architects have proposed a memory organization that combines different types of memory into a single *Heterogeneous Memory Architecture* (HMA) [22, 40]. For example, an HMA system could employ on-package die-stacked memory to provide high bandwidth, and yet enable greater capacity via an off-package DDRx memory. Many data placement techniques have been proposed for such architectures in order to maximize performance in spite of the latency and throughput differences of the component memories [2, 7, 23, 32].

Prior work on these heterogeneous memory systems have focused on the two axes of heterogeneity: capacity and performance. However, these systems are also highly heterogeneous in their reliability characteristics, i.e., error rate and fault tolerance. Data placement strategies that only consider the first two axes of heterogeneity can have a dramatically negative effect on reliability. Die-stacked memory has higher fault rate owing to higher bit density and new failure modes (e.g., TSV failure) [43, 44]. Moreover, die-stacked memory typically employs weaker error correction than conventional DDRx memory due to cost and complexity constraints [24]. This paper makes the case for a reliability-aware data placement strategy that enables sustained operation on an HMA system at peak bandwidth, and simultaneously mitigates risk due to transient faults.

This paper shows that a naïve performance-focused data placement technique which places frequently accessed memory pages (hot pages) in die-stacked memory results in lower overall reliability. Figure 1 shows the performance and reliability characteristics of a variety of placement techniques that place different proportions of hot pages in die-stacked

memory – the loss in reliability to achieve full performance can be quite extreme. By analyzing memory pages for their hotness and vulnerability, we devise effective reliability-aware placement strategies for HMA.

To quantify the vulnerability of memory pages we use *Architectural Vulnerability Factor* (AVF) analysis [42]. The AVF of a memory page is defined as the probability that a transient fault will result in an observable program error. Hence, a page with higher AVF *(high-risk)* is more likely to result in incorrect program execution than a page with lower AVF *(low-risk)*. *A key result of this research is that page hotness (access rate) is not necessarily correlated with AVF.* In fact, we show that applications can have as much as 39% of their memory pages that are both hot and low-risk. This property raises the possibility of devising placement schemes that preserve both reliability and performance on a heterogeneous memory architecture – that is, we can operate in the top right region of Figure 1, a region inaccessible to purely performance-focused placements. Consequently, we make the following contributions:

- By quantitative categorization of application data into quadrants in the hotness-risk spectrum, we show that applications can have as much as 39% of their memory footprint in the *hot* and *low-risk* category.
- We establish simple heuristics to allow for low-cost hotness and risk monitors for data in memory, and further develop a *heuristic-based* static data placement technique that improves reliability by 1.6x, sacrificing only 1% in performance.
- The reliability-aware *dynamic migration* schemes we propose eliminates the need for prior profiling and improve reliability by 1.5x on average, while limiting performance loss to 4.9%.
- We demonstrate the efficacy of *program annotation-based* data placement. We show that by annotating only a handful of program structures, typically 1 - 6, we improve reliability by 1.3x on average, while incurring a marginal performance loss of 1.1%.

2. BACKGROUND

In this section, we review several key reliability terms, formulas, and derive specific equations important to this work. Furthermore, we discuss relevant background on Heterogeneous Memory Architecture (HMA). While this work uses the specific example of the AMD HBM and DDRx memory, we note that the reliability-aware techniques developed in this work are general to any memory architecture which exhibits heterogeneity in terms of both performance and reliability. For example, these techniques are applicable to systems that integrate non-volatile and other emerging memory technologies into the hierarchy since they exhibit heterogeneity along both axes.

2.1 FIT-Rate, AVF, and SER

FIT-Rate. A device's Failure-in-Time (FIT-Rate) or raw failure rate due to single event upsets (SEUs) depends on circuit characteristics and the neutron flux in its environment. For example, the Cielo super computer, located in Los Alamos, New Mexico, is exposed to six times higher neutron flux than Hopper, located in Oakland, California [12]. The

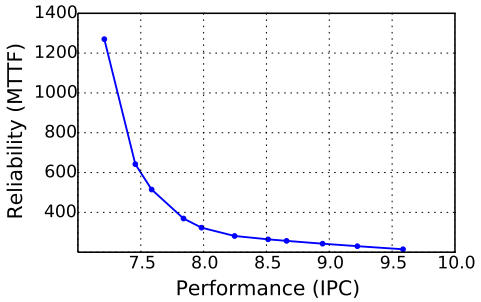


Figure 1: Reliability vs. Performance for HMA averaged over astar, cactusADM, mix1 benchmarks (methodology in Section 4) running on a 16 core multicore. Individual points reflect different proportions of hot pages placed in the faster stacked RAM.

circuit characteristics that affect FIT rates are cross-section area, the amount of charge required to toggle a bit (cause fault), and charge collection efficiency [55]. Shrinking technology results in a smaller amount of charge to toggle a bit and a reduced cross-section area. With every new process technology, the number of bits per unit area increases exponentially resulting in a rise in raw FIT rates due to SEUs [18].

Architectural Vulnerability Factor (AVF). AVF of a hardware structure is defined as the probability that a transient fault will result in an observable program error. Mukherjee, et al. propose a technique to estimate AVF of a processor [42]. That work tracks and groups the bits of a hardware structure into two categories: (a) those necessary for architecturally correct execution (ACE bits), and (b) the remaining as un-ACE bits. A fault in an ACE bit will result in a program-observable error in the absence of error-correction techniques, while a fault in un-ACE bits will go unnoticed. For example, all branch predictor bits are un-ACE because a fault in the predictor bits does not impact correctness. Furthermore, a bit can be ACE for only a fraction of the total program execution time and un-ACE for the rest of the time. For example, a physical register ($R1$) written at the beginning of the execution, read half-way, and dead thereafter is in ACE state for only half of the total execution time. The AVF for $R1$ is the fraction of the total execution time it is in ACE state. Similarly, the AVF of a hardware structure Mi with bit size B_{Mi} over a period of N cycles is expressed using the following equation.

$$AVF_{Mi} = \frac{\sum_{n=0}^{N}(ACE\ bits\ in\ Mi\ at\ cycle\ n)}{B_{Mi} \times N} \quad (1)$$

Soft Error Rate (SER). SER of a hardware structure Mi is the product of its FIT rate and AVF (see Equation 2). SER is the probability of an uncorrectable error resulting in incorrect program execution.

$$SER_{Mi} = FIT_{Mi} \times AVF_{Mi} \quad (2)$$

Equation 2 scales the FIT for transient uncorrectable errors with AVF. The FIT and AVF values in the equation capture the probability of a transient uncorrectable error and of the application reading the erroneous bit, respectively. In Equation 2, then, the probability of incorrect program execution scales down with the incidence of erroneous bits that are never read or overwritten before being read.

2.2 Heterogeneous Memory Architecture

A Heterogeneous Memory Architecture consists of multiple memory modules (e.g., an HMA system could combine on-package die-stacked DRAM with off-package conventional DDRx memory). The genesis of HMA is a result of the high bandwidth and capacity requirement of emerging workloads running on a highly parallel manycore CPU, GPU, and accelerator architectures, prompting hardware architects to mix-and-match 3D high-bandwidth memories with conventional off-package DDRx.

Off-package memory is a 2D arrangement of DRAM chips, where each chip renders a fixed number of bits every clock cycle (e.g., 4 bits are rendered in an x4 arrangement and 8 bits are rendered in an x8 arrangement). A collection of x8 DRAM chips operating in lockstep renders a 64-bit word on the data bus, with each chip contributing an 8-bit subset. An additional 8-bit chip provides single-bit correction and double-bit detection (ECC). Moreover, in an x4 DRAM arrangement, a stronger symbol-based correction can be enabled using the ChipKill [10] technology which distributes ECC bits across different DRAM chips.

Die-stacked memory with a single DRAM chip is capable of rendering a 128-bit word every clock cycle, thereby providing higher bandwidth. However, direct implementation of ECC- and ChipKill-based Reliability, Availability and Serviceability (RAS) have proven to be inefficient for die-stacked memory owing to significant bandwidth loss and/or area overhead [25]. Sim, et al. [58] propose a modification to the conventional ECC-based design for die-stacked DRAM to provide ChipKill-level RAS at a moderate cost in performance, but they incur a high capacity overhead due to the duplication of dirty blocks in adjacent banks. Micron's Hybrid Memory Cube (HMC) [49] provides strong reliability for its 3D die-stacked memory. However, it requires special DRAM chips and custom memory controller designs. Although considerable progress has been made in terms of low-cost innovative solutions for improving the reliability of die-stacked memories [24, 28, 43], the gap between the reliability characteristics of die-stacked and off-package DRAM has continued to widen [45], thereby suggesting that the heterogeneity in reliability will continue to exist between different memories in an HMA system.

3. EXPERIMENTAL METHODOLOGY

In the following subsections, we describe our simulation framework and workloads used in this study. We also explain the usage of real-world memory failure data from a field study on a large-scale system.

3.1 Simulation Framework

In this subsection, we describe our simulation framework which estimates both performance and reliability using Ramulator [35], a trace-driven DRAM simulator that provides cycle-accurate performance models for different memory standards such as DDR3/4, LPDDR3/4, GDDR5, and HBM. The trace file includes the number of intervening non-memory instructions, program counter, memory address, and request type (instruction/data, read/write, etc) for every memory request. We generate memory traces using SimPoints [54] and PinPlay [34] to capture the representative regions of an application. Furthermore, to only capture the main memory activity, we perform cache filtering using Moola [53].

Processor	Values
Number of cores	16
Core Frequency	3.2GHz
Issue width	4-wide out-of-order
ROB size	128 entries
Caches	Values
L1 I-cache (private)	32KB, 2-way set-associative
L1 D-cache (private)	16KB, 4-way set-associative
L2 cache (shared)	16MB, 16-way set-associative
Low-reliability Memory	Values (HBM)
Capacity	1GB
Bus frequency	500Mhz (DDR 1.0GHz)
Bus width	128 bits
Channels	8
Rank	1 Rank per Channel
Banks	8 Banks per Rank
ECC	SEC-DED [21]
High-reliability Memory	Values (DDR3)
Capacity	16 GB
Bus frequency	800 MHz (DDR 1.6 GHz)
Bus width	64 bits
Channels	2
Ranks	1 Rank per channel
Banks	8 Banks per Rank
ECC	single-ChipKill [10]

Table 1: System Configurations.

We extend Ramulator to simulate two levels of heterogeneous memory and compute the AVF of the data in memory at a 4 KB page granularity. We specifically target a 16-core system that resembles AMD's Opteron with a heterogeneous memory architecture that consists of AMD's High Bandwidth Memory (HBM) and conventional DDRx memory as on-package and off-package memories, respectively. The detailed system configuration is outlined in Table 1.

3.2 FIT-Rate Analysis

We use memory FIT rates from an AMD field study on a large-scale system (Jaguar) located at Oak Ridge National Laboratory [60]. The Jaguar system consists of 18,688 two-socket nodes. Each socket contains a 6-core AMD Opteron processor. The system has 2.69 million DRAM devices [48]. The average FIT Rate for various DRAM components (bit, row, column, word, bank, rank) are reported over a period of 11 months, for both transient and permanent faults. We use the transient FIT rate data as an input to an event-based DRAM fault simulator, i.e., FaultSim [44]. FaultSim's event-based design and real-world failure statistics make it a fast and accurate tool for our reliability studies. We configure FaultSim for our memory architecture and run 100K and 1 million fault simulations for SEC-DED and ChipKill respectively to ensure high precision levels. In each simulation, a fault is injected in a bit, word, column, row, or bank based on their FIT rates, a selected error-correction scheme is applied, and the outcome is recorded as detected, corrected, or uncorrected error. An error occurs when the failed bit is used in the software. We use the probability of uncorrected errors

Bench/Mix	mix1	mix2	mix3	mix4	mix5
mcf	3x	2x		1x	
lbm	2x	3x		1x	
milc	2x			1x	
omnetpp	1x		2x		
astar	2x		1x		
sphinx	1x		2x		
soplex	2x	3x		3x	
dealII		3x	1x	1x	3x
libquantum	2x		1x	3x	
leslie3d			2x	1x	3x
gcc	1x		2x	1x	
GemsFDTD		2x	2x	1x	1x
bzip		1x	1x	2x	3x
bwaves					1x
cactusADM		2x	2x	1x	5x

Table 2: Mixed workload description.

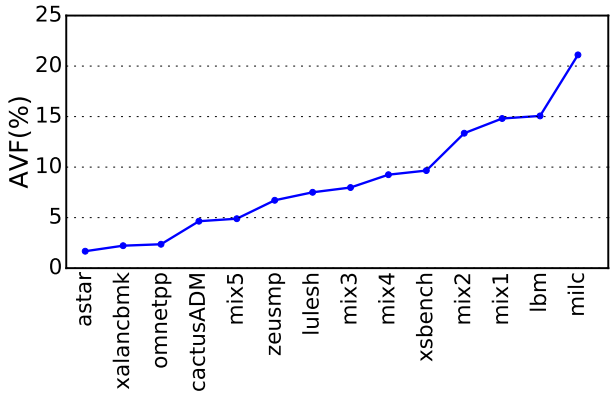


Figure 2: Average AVF of memory for the SPEC [19] benchmarks, ProxyApps [20], and mix workloads using only DDRx memory. The AVF of memory varies from 1.7% (astar) to 22.5% (milc).

due to transient faults to measure the failure probability of our HMA architecture.

3.3 Workloads

We evaluate our reliability-aware techniques using seven benchmarks from the SPEC CPU2006 [19] benchmark suite. We also use two benchmarks from the US Department of Energy (DoE) for evaluating HPC systems: XSBench [61] and LULESH [1]. We create homogeneous workloads from these nine benchmarks, where 16 copies of the same benchmark are run in parallel. Each copy has its own memory pages and different copies of the same workload don't share pages. Additionally, we mix benchmarks from high, medium, and low AVF applications to create five realistic datacenter workloads as shown in Table 2.

4. MOTIVATION

In this section, we motivate the use of AVF to evaluate the vulnerability of a memory page, and then quantitatively show that page vulnerability and hotness have a weak correlation.

4.1 AVF Analysis of Memory Pages

The AVF of a bit is the fraction of time that bit is in Architecturally Correct Execution (ACE) state (see Equation 1). While in an ACE state, any extraneous changes to the bit's value will result in incorrect execution. Figure 3 (a) shows a bit in memory written and read twice during program execution. After the bit is written for the first time, by request WR1, it is in the ACE state until it is read by request RD1. Any transient error (particle strike) in between WR1 and RD1 could result in incorrect execution. Hence, the AVF of the bit goes up by a fraction of (t_{RD1}/t_{total}). The bit is read again by RD2 which further adds (t_{RD2}/t_{total}) to the effective AVF. The bit is in non-ACE state from RD2 to WR2. The AVF of this bit in memory for the entire execution can be given by the following equation.

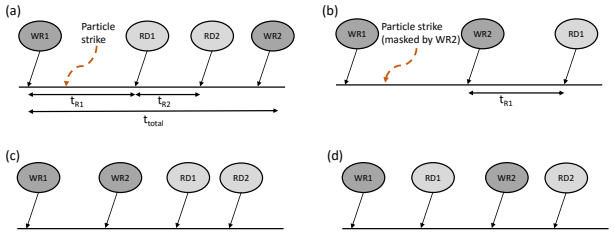


Figure 3: Architectural vulnerability factor (AVF) of four different cache lines in memory. While (a) shows an unmasked error, (b) illustrates how a write from CPU to memory overwrites its state masking the effect of particle strike. Figure (c) and (d) show that two lines of memory could have the same hotness levels, but different AVF depending upon the sequence of reads and writes.

$$AVF_i = \frac{t_{RD1} + t_{RD2}}{t_{total}} \quad (3)$$

Figure 3(b) shows a particle strike in between two writes, which will be masked as the correct value for RD1 will be over-written by WR2. Hence, the bit in Figure 3 (b) is in non-ACE state between WR1 and WR2. Two bits with the same number of reads and writes could potentially have very different AVFs, as shown by Figure 3 (c) and (d). However, a page placement policy that only takes into account page hotness will place these pages into die-stacked memory with equal likelihood, potentially hurting the overall RAS goals of the system.

In this work, we perform AVF analysis on memory at a cache line granularity because memory reads and writes occur at cache line granularity. We sum the AVF of individual cache lines to compose the AVF of a page and divide it by the size of the hardware structure as per Equation 1. Figure 2 shows the AVF of applications in increasing order. The figure demonstrates that the AVF varies significantly among applications. Thus, motivating potential reliability benefits from AVF-aware application-specific data placements.

4.2 Hotness-Risk Correlation

In order to obtain the upper bound on performance for HMA systems, we first explore a profile-guided static page placement that profiles each workload to obtain page-granularity hotness statistics, and subsequently selects and places the top 1GB of hot pages in HBM, with the remaining pages going into the DDRx memory.

Figure 5 shows the results of performance-focused static placement. We observe that all workloads show an increase in performance (IPC on left y-axis) with performance-focused placement for HMA relative to only DDRx memory. On average, applications show 1.6x boost in performance over only DDRx memory. However, we also observe a drastic in-

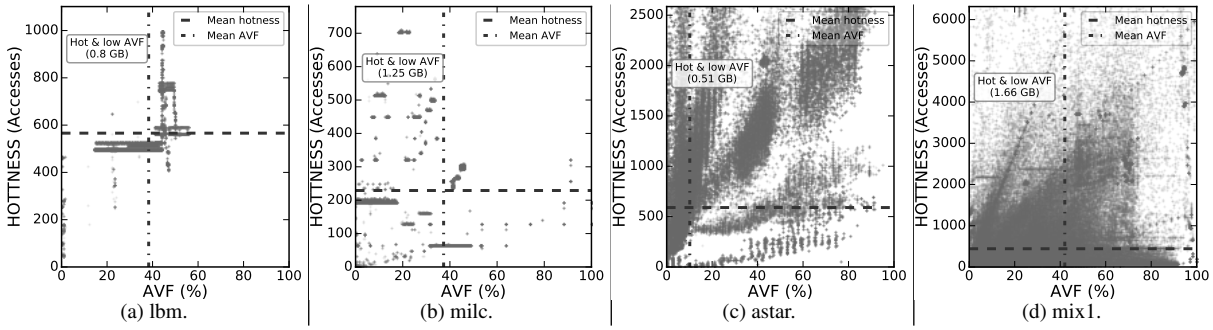


Figure 4: Page distribution of the entire memory footprint for four different workloads. Different workloads have different span across mean hotness and AVF. For the mix1 workload, there are 29.4% (1.66GB) of the total pages (5.64GB) that are both *hot* and *low-risk*. These pages are ideal candidates for memories with higher FIT rates and/or weaker correction.

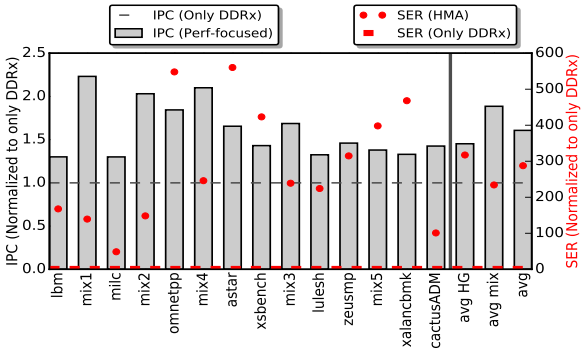


Figure 5: Performance-focused Data Placement for HMAs result in 1.6x boost in performance (IPC) and 287x increase in soft error rate (SER) relative to only DDRx memory.

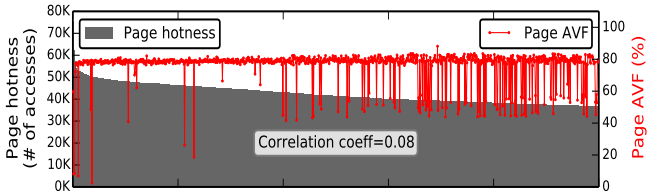


Figure 6: Page vulnerability (AVF) and page hotness (access counts) for 1000 most hot pages of a workload (mix1). Page AVF and hotness have a weak correlation ($\rho = 0.08$).

crease in soft error rate (SER on right y-axis) – 287x relative to only DDRx memory, potentially severely impacting system scalability. Thus, placing hot pages in stacked memory greatly exposes the system to the less reliable memory. If hotness and AVF are highly correlated, as these results seem to indicate, then we are forced to make difficult trade-offs between reliability and performance.

To understand the relationship between page hotness and vulnerability (risk), we measure both metrics on a single memory architecture that uses only DDRx memory. We place the entire memory footprint of a workload in DDRx memory and run the simulation using our modified Ramulator. We estimate page hotness using raw access counts (reads and writes) and page vulnerability by AVF analysis as described in Section 4.1.

Figure 6 shows the top 1000 hottest pages of a workload (mix1) arranged in decreasing order of their hotness. The left y-axis represents the page hotness measured using raw access counts and the right y-axis represents the page vulnerability measured as AVF percentage. The graph shows that most of the hot pages have AVF at around 80%. However, there are pages (in the top 1000 hot pages) which have AVF below 60% and as low as 5%. Hence, we conclude from Figure 6 that page hotness and AVF have a weak correlation. The correlation coefficient between hotness and AVF for the entire memory footprint is 0.08. Thus, there exists an opportunity of identifying hot pages with low-AVF (low-risk).

In order to quantify this opportunity, we divide the entire memory footprint into hot & cold pages and high- & low-AVF pages. We split the memory footprint of each workload around mean hotness and mean AVF values as shown by the scatter plots in Figure 4. Each memory page of the workload is plotted as a point on the scatter plot with its AVF along the x-axis and hotness along y-axis. The horizontal line represents mean hotness (access count) and the vertical line divides the memory footprint into low-AVF (low-risk) and high-AVF (high-risk) pages. These two lines divide the memory footprint into four sections: (i) hot and high-risk, (ii) hot and low-risk, (iii) cold and high-risk, (iv) cold and low-risk. We observe the presence of pages in all four quadrants in all workloads, although *lbm* is an outlier with few pages in the upper left. In general, we find a considerable number of pages in the upper left quadrant (hot and low-risk). For the *mix1* workload, we find 1.66 GB of pages to qualify as hot and low-risk pages. Such pages are ideal candidates for high-bandwidth low-reliability memory, such as HBM, to allow for a performance-focused and reliability-aware operation. For workloads we study in this work, we find that hot and low-risk pages account for anywhere between 9% and 39% of the entire memory footprint.

5. ORACULAR DATA PLACEMENT

An ideal data placement for an HMA system operates near the IPC of a performance-focused placement and near the SER of only DDRx memory. In this section, we explore oracular page placement schemes for HMAs with respect to both hotness (using raw access counts) and risk (using AVF). Specifically, we explore two oracular data placement schemes: (1) a highly reliability-focused scheme that minimizes SER by placing only *low-risk* pages in HBM, and (2) a more balanced scheme that aims at simultaneously minimizing SER while maintaining high IPC, by placing pages in the *hot* and *low-risk* quadrant in HBM. Finally, we suggest proxies for AVF estimation and evaluate data placement schemes based on these AVF proxies which do not require

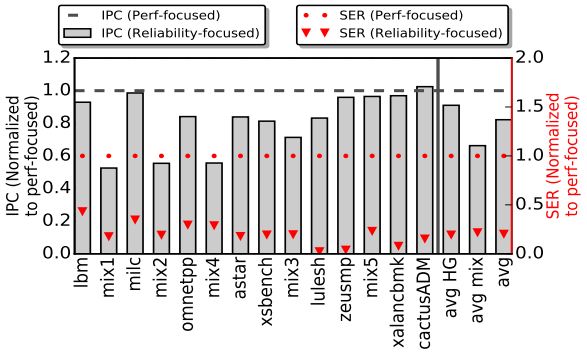


Figure 7: Naïve Reliability-focused data placement for HMAs reduces the SER by 5x, but suffers from 17% loss in performance relative to performance-focused data placement.

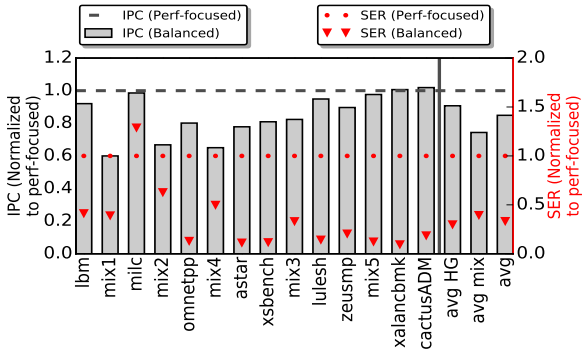


Figure 8: Balanced data placement for HMAs reduces the SER by 3x, but suffers from 14% performance loss relative to performance-focused data placement.

the full complex calculation of per-cache-line AVF over a page lifetime.

5.1 Reliability-focused Data Placement

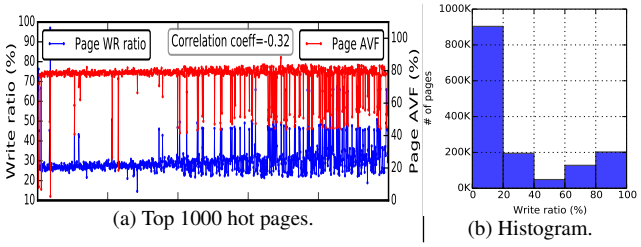
In contrast to performance-focused data placement, a naïve reliability-focused data placement scheme profiles each workload to obtain page-granularity AVF statistics and subsequently places all *low-risk* pages (i.e., pages with AVF below a certain threshold) in HBM with the remaining pages going into DDRx memory. These pages correspond to the leftmost quadrants on the scatter plots shown in Figure 4. Note that an AVF-focused placement does not take hotness into account and could potentially end up selecting pages from both the top-left and bottom-left quadrants.

Figure 7 shows the performance (IPC on left x-axis) and reliability (SER on right x-axis) of reliability-focused data placements for all workloads. On average, we observe that the soft error rate is reduced by 5x, but the performance degrades by 17% relative to performance-focused data placement (described in Section 4.2). This performance loss is due primarily to the placement of *cold* and *low-risk* pages in HBM, and the number of such pages depends on the individual bandwidth requirement and hotness span across pages for each workload.

We further note that the workloads in Figure 7 are arranged in decreasing order of their MPKI (Misses Per Kilo Instructions). Therefore, workloads in the left half of the graph are bandwidth-intensive and the ones on the right half are latency-sensitive. Moving hot pages from HBM to off-package DDRx memory affects the performance of bandwidth-intensive workloads more than latency-sensitive workloads, thereby explaining the greater performance loss of workloads in the left half of Figure 7. Outliers include *lbm* and *milc* that lose only 6% and 1% in performance respectively, due to reliability-focused data placement. This is due to uniform access counts to most pages, as shown in Figure 4, making their performance relatively insensitive to which pages get moved to HBM.

5.2 Balanced Data Placement

In order to optimize for both performance and reliability, we next explore a balanced data placement scheme that selects and places *hot* and *low-risk* pages in HBM. Figure 8 shows the performance (IPC on left x-axis) and reliability (SER on right y-axis) of such a balanced scheme for all



(a) Top 1000 hot pages. (b) Histogram.

Figure 9: (a) The graph shows the write ratio for the corresponding pages in Figure 4d. The graph suggests that there is a negative correlation between page AVF and write ratio ($\rho = -0.32$). (b) The histogram shows number of pages with write ratio categorized into bins – 1-20%, 21-40%, and so on. The workload has mostly read heavy pages. However, there are 125K and 200K pages which are write heavy as shown in the final two bins.

workloads. We observe that the SER is reduced by 3x, but the performance drops by an average of 14%. This performance result is only slightly better than the naïve reliability-focused placement, due in large part to restricting ourselves to a single quadrant, even if not many pages exist in that quadrant. Thus, this is still a conservative policy, never putting high-risk pages in HBM.

Moreover, such a static policy relies on oracular knowledge of AVF, but exact AVF estimation or prediction is a hard problem. Walcott, et al. [64] use IPC as a heuristic to estimate AVF of instruction queue, load-store queue, and register files. They demonstrate that higher IPC leads to lower AVF. However, that heuristic is not useful for evaluating individual memory pages.

5.3 Heuristics for AVF Estimation

A memory page that has been written more often than read is likely to have lower AVF than a page that has mostly reads. Recall that most periods of data "deadness" end in a write, so more writes indicate more dead intervals. A high ratio of writes to reads indicates the likelihood of longer dead intervals. Figure 9a shows the write ratio (Wr/Rd ratio) on the left y-axis and AVF on the right y-axis of the top 1000 hot pages for the *mix1* workload. We see a much stronger correlation between AVF and write ratio than we see between AVF and hotness. We measure a negative correlation between write ratio and AVF of -0.32, which is not a heavy correlation, but strong enough to give us some opportunity to easily get a rough approximation of AVF. Using this correlation information, we classify a page with low writes relative to reads

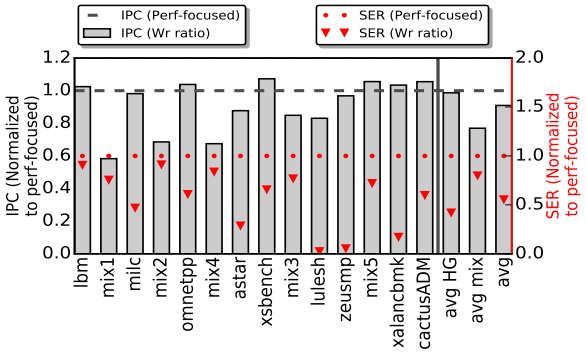


Figure 10: Wr ratio data placement for HMAs reduces the SER by 1.8x, while experiencing 8.1% loss in performance relative to performance-focused data placement.

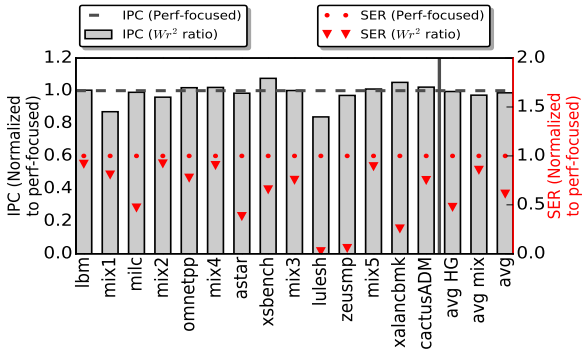


Figure 11: Wr^2 ratio for HMAs reduces the SER by 1.6x, at a marginal performance loss of 1% relative to performance-focused placement.

as a high-risk page, and a page with high writes relative to reads as low-risk. Therefore, we propose using write ratio as a proxy for AVF to estimate the risk rating of memory pages.

5.4 AVF Heuristic-based Data Placement

In this section, we demonstrate the efficacy of two data placement techniques based on (a) Wr ratio, and (b) Wr^2 ratio. The former is measured as Wr/Rd and the latter is measured as Wr^2/Rd. While both serve as proxies for AVF, the latter places an additional weight on page hotness. While this may not be immediately obvious, consider that the Wr ratio is unitless, but the Wr^2 ratio has a unit of *accesses*. We find pages with high Wr and Wr^2 ratio (for the two respective heuristics), which place the identified pages in HBM while the rest go into DDRx memory.

5.4.1 Top Wr Ratio Data Placement

Our AVF heuristic-based placement scheme leverages page-level Wr ratio statistics obtained from prior profile runs in order to fill the HBM memory with pages with high Wr ratio, while placing the rest in off-package DDRx memory. As shown in Figure 10, a top Wr ratio-based data placement results in an average improvement of 1.8x in SER, while sacrificing 8.1% in performance, relative to performance-focused data placement. Although the top Wr ratio-based data placement does improve performance relative to a reliability-focused data placement, it still sometimes selects cold, albeit low-risk, pages for HBM, thereby limiting the potential performance gains.

5.4.2 Top Wr^2 Ratio Data Placement.

As discussed earlier, the Wr^2 ratio still serves as a proxy for AVF, but also biases more heavily towards pages with a higher absolute number of writes, thereby avoiding cold pages. For example, consider two pages p1 and p2 with write-read ratios as 4:1 and 400:200 respectively. A Wr ratio-based placement favors placing p1 in HBM over p2, while a Wr^2 ratio-based placement favors placing p2 in HBM over p1, owing to the higher absolute number of accesses. From Figure 11, we observe an average reduction of 1.6x in SER, with the performance gap narrowing to just 1%, when compared to performance-focused data placement.

While the techniques we describe in this section are oracular (assume full knowledge of data access before placement), in Section 7 we show the viability and effectiveness of a static placement that is driven by minimal annotation of a program's *hot* and *low-risk* structures.

6. DYNAMIC MIGRATION MECHANISMS

While our heuristics allow AVF approximation without full AVF modeling, any static technique is inherently limited by pre-execution profiling. This section seeks to explore the opportunity for dynamic reliability-aware migration mechanisms. Specifically, we leverage insights developed in Section 5 to transform a state-of-the-art performance-focused migration mechanism into a reliability-aware migration mechanism. We also examine the applicability of a recently proposed algorithm for memory activity tracking to reduce hardware cost in our scenario.

6.1 Performance-focused Migration

We model our performance-focused migration mechanism based on the HMA architecture proposed by Meswani, et al. [40]. The migration scheme employs a set of hardware counters that keep track of raw access counts (reads and writes combined) for each page in memory. At the end of pre-defined intervals, these counters are used to generate migration pairs to replace cold pages from on-package HBM with hot pages in off-package DDRx memory. The hotness criteria is determined using a threshold – any page with raw access counts greater than the set threshold value is deemed to be hot and therefore, eligible for migration.

Migration Interval. Migrating pages at run-time has a performance cost. The cost of migrating a page from one memory to another is governed by the slowest memory in the system. Therefore, it is important to select migration intervals carefully. We perform a sweep over the interval length to evaluate its impact on performance. We select three different workloads for our evaluation. Each workload is characterized by different memory intensities (low, medium and high), and different memory access patterns. Our results (shown in Figure 13) reveal that an interval length of 100ms leads to the best performance, in agreement with the identified interval by Meswani, et al. [40].

To eliminate cold start effects in our simulations (which emulate a period in the middle of execution), we assume a good pre-measurement placement, in this case one that starts with the top hot pages from our oracular static placement scheme in HBM, before we begin execution of a simpoint. While this is a good overall placement, it is not necessarily the best per-interval placement. The set of top hot pages

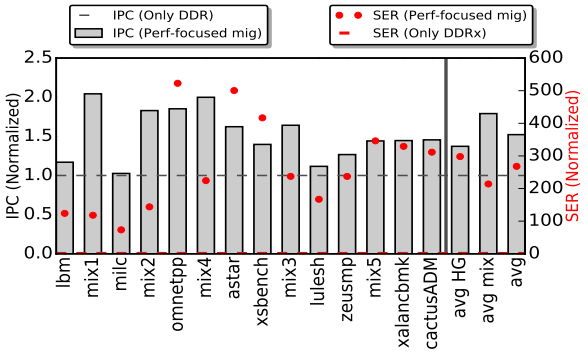


Figure 12: Performance-focused migrations improve IPC by 1.52x relative only DDRx memory. However, the soft error rate is increases by 268x relative to only DDRx memory.

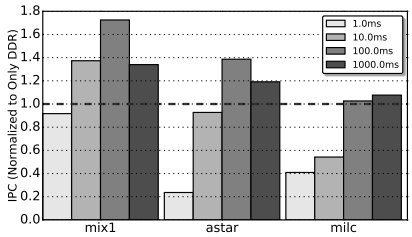


Figure 13: Interval sweep to determine suitable migration frequency.

changes considerably from interval to interval, triggering an average of 47,014 migrations every interval.

Hotness Threshold. The total number of migrations is also a function of the defined hotness threshold, i.e. how hot a page has to be in order to be eligible for migration to HBM. One of the insights from our static exploration is that there is significant variation in page hotness across different workloads. Hence, choosing a hardwired value as a threshold cannot serve every application fairly. We use dynamic mean page hotness levels during each interval to determine the threshold to be used during migration. All pages in slow memory above mean page hotness are marked as candidates for migration to faster high-bandwidth memory.

Performance and Reliability. Figure 12 shows the IPC (left y-axis) and SER (right y-axis) for our performance-focused migration scheme. We observe that the dynamic migration scheme is only 5.8% slower (1.52x vs. 1.6x), and achieves a minor reduction of 1.11x in SER relative to our static oracular performance-focused data placement. The performance degradation comes from both the migration cost and the inability to react quickly to execution changes due to a long measurement interval. Note that subsequent results in this section will be normalized to these values (dynamic performance-based migration) since that represents the current state of the art for dynamic schemes.

6.2 Reliability-aware Migration

In this section, we propose minor modifications to the performance-focused migration mechanism, in order to transform it into a reliability-aware mechanism, by leveraging insights from Section 5.3. By simply splitting the hardware counters into two sets, one for reads and one for writes, it is possible to measure a page's hotness (reads + writes) as well as its risk, defined as Wr/Rd. Since we have a precise measurement for page hotness, we no longer need to calculate

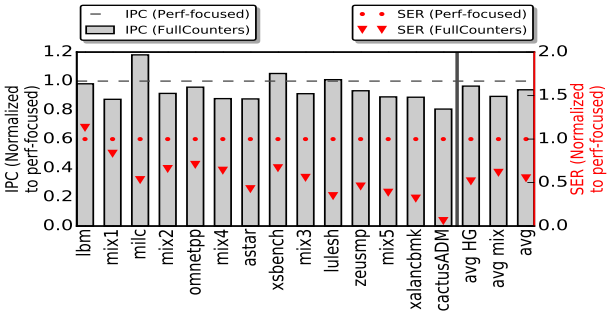


Figure 14: Reliability-aware dynamic migration (FullCounters): On average reliability-aware migration using FullCounters reduces the soft error rate 1.8x at a cost of 6.0% relative to performance-focused migrations.

Wr^2/Rd. We use mean hotness and risk values as thresholds to divide memory pages accessed during the interval into hot and cold, and high-risk and low-risk, respectively. Similar to the performance-focused migration scheme, to eliminate cold start effects, we assume an initial placement of the top hot and low-risk pages from our static oracular placement in HBM, before we begin the execution of a simpoint. Further, at each interval, our mechanism attempts to exchange all cold or high-risk pages currently residing in HBM with hot and low-risk pages from DDRx memory. We refer to this reliability-aware migrations mechanism as Full Counter-based (FC-based) mechanism.

Figure 14 shows the IPC (left y-axis) and SER (right y-axis) for reliability-aware migrations using Full Counters (FC) relative to performance-focused migrations. We observe varying levels of performance loss across different workloads. The *milc* workload shows a slight speedup because it incurs fewer migrations than a performance-focused scheme. On average, we observe a reduction of 1.8x in SER, at a performance loss of 6% when compared to performance-focused migrations. Therefore, we show that at a very nominal cost in performance, a heuristic-based run-time risk estimation technique such as our migration scheme can substantially improve the reliability of an HMA system.

6.3 Hardware Cost Analysis

In this section, we discuss the hardware cost of the proposed dynamic reliability-aware migration scheme that uses Full Counters (FC). Our FC-based reliability-aware migration mechanism maintains two sets of counters for each page in memory to count the number of reads and writes. Our exploration reveals that 6-bit counters are sufficient to hold the largest access counts we observe. For generality, we employ 8-bit counters in our design. We further assume the counters to be saturating, so they do not overflow after an access count exceeds the maximum possible value. Given our assumptions, the activity tracking (identifying migration candidates) part of this mechanism requires 16 bits per 4K pages. In our example 17 GB HMA, (1 GB HBM and 16 GB DDRx – See Table 1), we have a total of 4.25M pages, for which we need 8.5 MB to store our Full Counters. This implies that our reliability-aware migration scheme requires an additional storage of 4.25 MB for activity tracking than a fully performance-focused migration scheme.

Static (Oracle)				Heuristic (Oracle)				Dynamic Migrations					
Reliability-focused [Section 5.1]		Balanced [Section 5.2]		Wr ratio [Section 5.4.1]		Wr^2 ratio [Section 5.4.2]		Reliability-aware (Full Counters) [Section 6.2]		Reliability-aware (Cross Counter) [Section 6.4]		Program Annotations [Section 7]	
IPC	SER	IPC	SER	IPC	SER	IPC	SER	IPC	SER	IPC	SER	IPC	SER
17%	5.0x	14%	3.0x	8.1%	1.8x	1%	1.6x	6%	1.8x	4.9%	1.5x	1.1%	1.3x

Table 3: Summary of static, heuristic, dynamic migration, and programmer annotation based data placement. All static placements/migration schemes are normalized to that of the performance-focused static placement/migration scheme. IPC and SER numbers show the IPC degradation and SER improvement relative to respective performance-focused placement.

6.4 Hardware Cost Optimization

In this section, we explore techniques to reduce the overall hardware cost of our reliability-aware migration scheme, by combining our FC-based tracking mechanism with a low cost hotness tracking approach based on the "Majority Element Algorithm" (MEA) [50].

6.4.1 Dynamic Migrations via Cross Counters

To design a cost-effective migration mechanism, we first deconstruct our dynamic reliability-aware mechanism into two distinct component units that optimize two different aspects – performance and reliability. While a performance unit replaces cold pages residing in HBM with hot pages from slower DDRx memory to provide higher overall bandwidth, the reliability unit replaces high-risk pages residing in HBM with low-risk pages from DDRx memory to improve the overall soft error rate. To this end, we propose the "Cross Counters" (CC) reliability-aware migration mechanism that implements a combination of MEA and Full Counters.

MEA is shown to be a more cost-effective hotness tracking approach since it favors recency by tracking relative updates to a limited set of most recently used pages, in comparison to a Full Counter-based approach that keeps track of raw access counts for every addressable page in the system. We model our MEA-based counter mechanism based on the MemPod architecture proposed by Prodromou, et al. [50]. Specifically, using a 32-entry MEA map and a 64 KB remap table cache, our performance unit can identify up to 32 globally hot pages every 50μs.

Despite the potential for overall hardware cost reduction, an MEA-based tracking approach (or any modification thereof) does not efficiently (in terms of both performance and hardware cost) capture absolute read and write counts over large intervals, which is especially important for a reliability-aware mechanism that tracks page risk. Therefore, we continue to use Full Counter-based tracking in our reliability unit, albeit only for a small fraction of the pages in HBM. These counters operate as described in the previous reliability-aware dynamic mechanism, tracking both read and write counts.

6.4.2 Hardware Cost of Cross Counters

In contrast to our FC-based reliability-aware migration scheme, our CC-based scheme employs Full Counters for only risk tracking in HBM. Specifically, our CC-based scheme uses 16-bit counters to track risk in all 262K pages in our example 1 GB HBM, thereby significantly reducing the hardware cost of Full Counters to just 512 KB. The MEA-based tracking mechanism we model from prior work requires no more than 100 KB of storage to track hotness of the most recently used pages, and a 64 KB cache for its remap table that keeps track of recently migrated pages. Overall, our Cross Counters-based migration mechanism requires 676 KB of additional storage, a significant reduction from the 4.25 MB additional storage required by our FC-based mechanism.

6.4.3 Cross Counters Implementation

In order to determine the appropriate migration interval for the MEA-based tracking mechanism, we first perform an interval sweep similar to the one we perform for our FC-based tracking mechanism (see Figure 13). However, in this case, we use smaller intervals set at 50 and 100 μs, since prior work shows that MEA-based migration mechanisms work well with smaller intervals that allow them to push a small number of pages frequently to HBM [50]. Our interval sweep reveals an optimal migration interval of 50μs for an MEA-based mechanism with 32 counters and 4 KB pages, which is in agreement with the migration interval identified by prior work. We call this interval the *MEA-interval*.

On the other hand, FC-based migration mechanisms have been shown to work well with larger intervals. This is due to the fact that they attempt to migrate a very large number of pages at once [40]. In our specific case of run-time risk estimation where we inspect the dynamic Wr ratio, it is important to accumulate a considerable number of reads and writes to make accurate assessments, which is only possible via larger intervals. In our Cross Counter-based migration scheme, we set the two managing units to operate at different intervals. The performance unit (MEA) migrates a very small set of hot pages to HBM every 50μs (MEA-interval). The reliability unit, on the other hand, performs run-time risk estimation for every page in HBM, every 100ms (FC-interval).

In our implementation, migrations are orchestrated by the performance unit. Based on the hotness criteria established by MEA, the performance unit migrates hot pages into HBM from DDRx memory. However, the performance unit also queries the reliability unit to check if there are high-risk pages waiting to be moved to DDRx memory. If such pages exist, migrations are performed in both directions – the performance unit migrates identified hot pages from DDRx memory to HBM, and the reliability unit migrates identified high-risk pages in the other direction. In case no pending high-risk pages exist (for example during the first FC-interval), the performance unit continues to migrate hot pages every MEA-interval, but the reliability unit only makes the counters available every FC-interval. At FC-interval, both performance and reliability units work together to move cold and

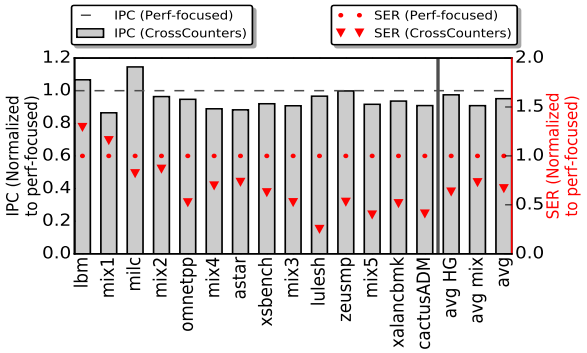


Figure 15: Reliability-aware dynamic migration (CrossCounter): On average reliability-aware migration using CrossCounters reduces the soft error rate 1.5x at a cost of 4.9% relative to performance-focused migrations.

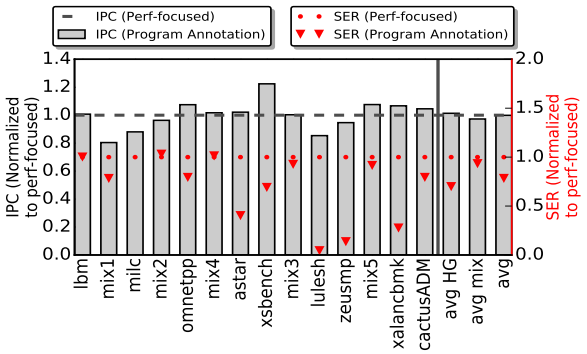


Figure 16: Program annotation reduces SER by 1.3x at a cost of 1.1% relative to perf-focused migration oracular placement.

high-risk pages out of HBM.

Figure 15 shows the IPC (left y-axis) and SER (right y-axis) results for the CC-based reliability-aware migration mechanism. On average, we observe that the SER is reduced by 1.5x, at a performance loss of 4.9% when compared to performance-focused migrations. In contrast, an FC-based reliability-aware migration mechanism incurs a 6% loss in performance, but achieves an overall SER reduction of 1.8x (See Table 3). The relative increase in SER is a direct consequence of allowing hot and potentially high-risk pages to migrate to HBM at the smaller MEA-intervals. However, if a page gets classified as high-risk at a later FC-interval, it is migrated back to DDRx memory, thereby mitigating risk as much as possible, at a coarser granularity.

While this policy works well with high-risk pages that are hot over short bursty intervals and tend to stay in DDRx memory for most of the execution, workloads such as *astar* have a great number of high-risk pages that stay hot for the entire execution. For such workloads, we keep bringing pages in and then removing them after an interval, once we have the opportunity to measure risk, resulting in some reliability exposure and greater migration overhead.

Furthermore, since MEA counters favor the temporal behavior of an access pattern, we observe significant performance improvement in workloads with striding patterns. For example, we observe an 11% performance improvement with the *cactusADM* workload compared to FC-based migrations. However, owing to CC's coarse-grained risk mitigation, we observe a 20% increase in SER for that workload. On average, CC effectively reduces the soft error rate by 1.5x at less

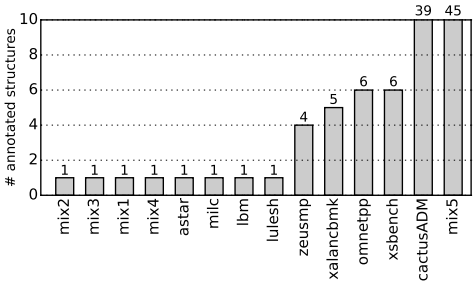


Figure 17: Number of annotated program structures. For most workloads annotating only a handful of program structures provides 1GB of potentially hot and low-risk pages.

than 4.9% loss in performance, with a minimal hardware cost of 676 KB, when compared to a fully performance-focused migration mechanism.

7. RELIABILITY-AWARE PROGRAM ANNOTATIONS

This section explores the viability of risk mitigation via program annotations that allow pinning of hot and low-risk data structures in HBM, with no significant performance degradation and hardware cost. These program annotations can be made by identifying data structures that are frequently accessed and yet do not remain live for a substantial duration of the program's execution. In our implementation, the program's ELF loader instructs the memory controller to pin annotated data structures to HBM, and further mark them as "pinned pages", so that they are unaffected by the migration policies implemented in the system.

As shown in Figure 16, it is possible to achieve a significant reduction of 1.3x in SER by minimal annotation of program structures, at a mere 1.1% degradation in performance, and with no additional hardware overhead, when compared to a fully performance-focused static oracular data placement. In fact, as shown in Figure 17, we can achieve this level of SER reduction with as few as 1 program annotation for most workloads and with an average of 8 annotations across all workloads. Note that this is highly influenced by two outliers, *cactusADM* and *mix1* that need 39 and 45 annotations respectively. We note that the annotations themselves can be performed either using domain expertise, and/or with the help of profile-guided compilation. In this work, we take advantage of prior profiling to perform minimal program annotation.

These results suggest that with careful program annotation, we can achieve a significant reduction in SER, close to that of a heuristics-based static oracular data placement. Supplementing such an annotation-driven static data placement scheme with a reliability-aware migration mechanism could potentially further improve the overall reliability of the system.

8. RELATED WORK

Naithani, et al. demonstrate that reliability-aware scheduling for heterogeneous multicores increases system reliability by as much as 60.2% when applications are scheduled using application vulnerability among available core choices [46]. Jiao, et al. propose techniques that consider the effect of data to predict and reduce SEUs in functional units [30, 31].

Venkat, et al. demonstrate that heterogeneity in multicores can be exploited not only for improved performance [63] but also for increased security [62].

Gupta, et al. present an aging-aware mechanism to handle the accumulation of permanent faults for an HMA system by tuning the access rate as the system ages [16]. However, they don't consider the effect of data vulnerability on transient errors. Gottscho, et al. propose Software-Defined ECC (SWD-ECC). SWD-ECC uses information theory to heuristically recover from detectable unrecoverable errors (DUEs) [13]. Flikker [38] lowers the refresh rate for low-risk data for DDRx memory. Flikker requires low-risk data to be annotated using programmer hints, a strategy we use in this paper for more optimal data placement, in terms of both performance and reliability.

The motivation for HMA configurations stem from the need for more capacity and bandwidth than what current die-stacked DRAM technology can provide [65]. The die-stacked portion of HMAs is typically managed as either a large, high-bandwidth Last Level Cache (LLC), or as Part-of-Memory (PoM), in which case the stacked capacity is available to the software. Several studies explore cache organizations for HMAs [9, 26, 37, 51]. When organized as a cache, stacked DRAM introduces new challenges when compared to traditional SRAM caches. Qureshi, et al. [51] demonstrate that traditional optimizations tailored for SRAM memories are not applicable for DRAM caches and result in degraded performance, requiring engineers to revisit and reconsider cache optimizations with DRAM technology in mind. A critical DRAM cache optimization is the design of intelligent tag stores that avoid multiple cache accesses for tag and data.

While die-stacked DRAM caches are shown to significantly improve the performance of latency-limited applications, they only marginally improve capacity-limited applications. When the stacked capacity is instead exposed to the application in a PoM configuration, capacity-limited applications also benefit in performance. Recent works [8, 40, 50, 56] propose mechanisms to manage stacked memory in a PoM configuration. These proposals follow a similar approach to the memory management problem: Identify data that are predicted to be *hot* (frequently accessed) in the near future and move it to fast memory. Since stacked memory is PoM, these mechanisms need to move some data out of fast and into slow memory before replacing it with the predicted hot data. CAMEO [8] attempts to strike a balance between managing stacked memory as a cache/PoM hybrid.

Meswani, et al. [40] propose a migration mechanism that employs HW as well as SW to manage HMAs. Predicting future hot data (4KB pages) is done via a set of access counters keeping an access count per page. At predefined time intervals, the pages with the highest access counts exceeding a set threshold are migrated (swapped) into fast memory. MemPod [50] is a recently proposed HW-based mechanism for memory management in PoM configurations. The proposed architecture clusters fast and slow memory channels into independently-operating memory "Pods" and only permits intra-Pod migrations. While flexibility is still technically restricted, MemPod's architecture offers enough options for pages to migrate leading to better fast memory utilization. MemPod's prediction for future hot data is done via the "Majority Element Algorithm" (MEA) algorithm [6, 33]. In this paper, we cross-breed both the above techniques (HMA and MemPod) to implement a cost-effective Cross Counters-based reliability-aware migration mechanism.

Sim, et al. [56] propose a hardware memory management mechanism designed to operate without high area overheads for bookkeeping structures. Memory is grouped into "segments" with only one fast memory location per segment. Memory pages can only migrate to the fast location of their segment, replacing the currently-residing page. This segmented approach significantly restricts the flexibility of this migration mechanism, occasionally leading to counterproductive operation – multiple hot pages in the same segment will keep evicting each other from fast memory, while the fast memory of a cold segment does not contribute towards improving performance. However, restricting migrations in this fashion requires very small bookkeeping structures, and removes the complexity of finding a fast memory location based on possibly more complex algorithms.

9. CONCLUSIONS

This paper presents low-cost heuristics to mark risky pages to better manage a heterogeneous memory system, balancing both performance and overall reliability. Our static heuristic-based reliability-aware policy that selects hot and low-risk pages for high-bandwidth low-reliability memory reduces SER by 1.6x compared to a conventional performance-focused data placement policy at a cost of 1% in performance. We devise two dynamic data migration policies. We show that monitoring hotness via Full Counters and risk via run-time heuristics results in 1.8x reduction in soft error rate at a performance loss of 6% and an additional hardware overhead of 4.25 MB, as compared to a performance-focused dynamic migration policy. To reduce the hardware overhead, we propose "Cross Counters", that use full counters for risk estimation in HBM and MEA counters to track hotness throughout the system. We demonstrate that Cross Counters reduce the soft error rate by 1.5x at a performance loss of 4.9% with a nominal hardware overhead of 676 KB, as compared to a performance-focused dynamic migration policy. We further show that it is possible to achieve a significant reduction of 1.3x in SER with minimal program annotation. These results open up new opportunities for optimization of performance, capacity, and reliability.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful insights. This research was supported by the U.S. Department of Energy (DoE). This work is also supported by the NSF Expedition in Computing grant CCF-1029783 and NSF grant CNS-1652925. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the DoE or NSF. AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

10. REFERENCES

[1] Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Technical Report LLNL-TR-490254.

[2] Neha Agarwal, David Nellans, Mark Stephenson, Mike O'Connor, and Stephen W. Keckler. Page placement strategies for gpus within heterogeneous memory systems. *ASPLOS*, 2015.

[3] R. Baumann. Soft Errors in Commercial Semiconductor Technology: Overview and Scaling Trends. *IEEE 2002 Reliability Physics Tutorial Notes, Reliability Fundamentals*, 2002.

[4] Bryan Black, Murali Annavaram, Ned Brekelbaum, John DeVale, Lei Jiang, Gabriel H. Loh, Don McCaule, Pat Morrow, Donald W. Nelson, Daniel Pantuso, Paul Reed, Jeff Rupley, Sadasivan Shankar, John Shen, and Clair Webb. Die Stacking (3D) Microarchitecture. In *MICRO*, Washington, DC, USA, 2006. IEEE Computer Society.

[5] Franck Cappello et al. Toward Exascale Resilience. *Int. J. High Perform. Comput. Appl.*, 2009.

[6] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 2004.

[7] Moinuddin K. Qureshi Chiachen Chou, Aamer Jaleel. BATMAN: Maximizing bandwidth utilization of hybrid memory systems. *Tech Report*, TR-CARET-2015-01(1):297–310, March 2015.

[8] Chiachen Chou, Aamer Jaleel, and Moinuddin K. Qureshi. Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, pages 1–12, Washington, DC, USA, 2014. IEEE Computer Society.

[9] Chiachen Chou, Aamer Jaleel, and Moinuddin K. Qureshi. BEAR: Techniques for Mitigating Bandwidth Bloat in Gigascale DRAM Caches. In *ISCA*, New York, NY, USA, 2015. ACM.

[10] T. J. Dell. A White Paper on the Benefits of Chipkill-correct ECC for PC server Main Memory. IBM, Technical Report, November, 1997.

[11] Xiangyu Dong, Yuan Xie, Naveen Muralimanohar, and Norman P. Jouppi. Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC, Washington, DC, USA, 2010. IEEE Computer Society.

[12] Flux Calculator. http://www.seutest.com/fluxcalculator.htm.

[13] Mark Gottscho. *Opportunistic Memory Systems in Presence of Hardware Variability*. PhD thesis, University of California, Los Angeles, 2017.

[14] Sudhakar Govindavajhala and Andrew W. Appel. Using memory errors to attack a virtual machine. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, SP '03, pages 154–, Washington, DC, USA, 2003. IEEE Computer Society.

[15] Manish Gupta, Daniel Lowell, John Kalamatianos, Steven Raasch, Vilas Sridharan, Dean Tullsen, and Rajesh Gupta. Compiler techniques to reduce the synchronization overhead of gpu redundant multithreading. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, pages 65:1–65:6, New York, NY, USA, 2017. ACM.

[16] Manish Gupta, David Roberts, Mitesh Meswani, Vilas Sridharan, Dean Tullsen, and Rajesh Gupta. Reliability and Performance Trade-off Study of Heterogeneous Memories. In *MEMSYS*, New York, NY, USA, 2016. ACM.

[17] Per Hammarlund, Alberto J. Martinez, Atiq A. Bajwa, David L. Hill, Erik Hallnor, Hong Jiang, Martin Dixon, Michael Derr, Mikal Hunsaker, Rajesh Kumar, Randy B. Osborne, Ravi Rajwar, Ronak Singhal, Reynold D'Sa, Robert Chappell, Shiv Kaushik, Srinivas Chennupaty, Stephan Jourdan, Steve Gunther, Tom Piazza, and Ted Burton. Haswell: The fourth-generation intel core processor. *IEEE Micro*, 2014.

[18] Jörg Henkel, Lars Bauer, Nikil Dutt, Puneet Gupta, Sani Nassif, Muhammad Shafique, Mehdi Tahoori, and Norbert Wehn. Reliable On-chip Systems in the Nano-era: Lessons Learnt and Future Trends. DAC '13, New York, NY, USA, 2013. ACM.

[19] John L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*.

[20] Mike Heroux, Rob Neely, and Sriram Swaminarayan. ASC Co-design Proxy App Strategy. 2016.

[21] M. Y. Hsiao. A Class of Optimal Minimum Odd-weight-column SEC-DED Codes. 1970.

[22] Intel. Knightslanding. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, pages 1–12, Washington, DC, USA, 2014. IEEE Computer Society.

[23] Pradeep Ramachandran Jayesh Gaur, Mainak Chaudhuri and Sreenivas Subramoney. Near-optimal access partitioning for memory hierarchies with multiple heterogeneous bandwidth sources. *HPCA*, 2017.

[24] Hyeran Jeon, Gabriel H. Loh, and Murali Annavaram. Efficient RAS support for die-stacked DRAM. In *2014 International Test Conference, ITC 2014, Seattle, WA, USA, October 20-23, 2014*, pages 1–10, 2014.

[25] Hyeran Jeon, Gabriel H Loh, and Murali Annavaram. Efficient ras support for die-stacked dram. In *Test Conference (ITC), 2014 IEEE International*, pages 1–10. IEEE, 2014.

[26] Djordje Jevdjic, Gabriel H Loh, Cansu Kaynak, and Babak Falsafi. Unison cache: A scalable and effective die-stacked DRAM cache. In *MICRO*. IEEE, 2014.

[27] Djordje Jevdjic, Stavros Volos, and Babak Falsafi. Die-stacked dram caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache. In *ISCA*, New York, NY, USA, 2013. ACM.

[28] Xun Jian, Vilas Sridharan, and Rakesh Kumar. Parity helix: Efficient protection for single-dimensional faults in multi-dimensional memory systems. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, HPCA '16, 2016.

[29] Xiaowei Jiang, Niti Madan, Li Zhao, Mike Upton, Ravishankar Iyer, Srihari Makineni, Donald Newell, Yan Solihin, and Rajeev Balasubramonian. CHOP: Adaptive filter-based DRAM caching for CMP server platforms. 2010.

[30] Xun Jiao, Yu Jiang, Abbas Rahimi, and Rajesh K. Gupta. Slot: A supervised learning model to predict dynamic timing errors of functional units. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, pages 1183–1188, 2017.

[31] Xun Jiao, Abbas Rahimi, Yu Jiang, Jianguo Wang, Hamed Fatemi, Jose Pineda de Gyvez, and Rajesh K Gupta. Clim: A cross-level workload-aware timing error prediction model for functional units. *IEEE Transactions on Computers*, preprint.

[32] Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. Heteroos: Os design for heterogeneous memory management in datacenter. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 521–534. ACM, 2017.

[33] Richard M Karp, Scott Shenker, and Christos H Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems (TODS)*, 2003.

[34] Chi keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa, and Reddi Kim Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *In PLDI âĂŹ05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 190–200. ACM Press, 2005.

[35] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Computer Architecture Letters*, PP(99):1–1, 2015.

[36] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 361–372, Piscataway, NJ, USA, 2014. IEEE Press.

[37] Yongjun Lee, Jongwon Kim, Hakbeom Jang, Hyunggyun Yang, Jangwoo Kim, Jinkyu Jeong, and Jae W Lee. A Fully Associative, Tagless DRAM Cache. In *ISCA*. ACM, 2015.

[38] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning. *ASPLOS*, 2011.

[39] Gabriel H. Loh and Mark D. Hill. Efficiently Enabling Conventional

Block Sizes for Very Large Die-stacked DRAM Caches. In *MICRO*, New York, NY, USA, 2011. ACM.

[40] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh. Heterogeneous Memory Architectures: A HW/SW Approach for Mixing Die-stacked and off-package Memories. In *HPCA*, 2015.

[41] Sarah E. Michalak, Kevin W. Harris, Nicolas W. Hengartner, Bruce E. Takala, Stephen Wender, and Others. Predicting the number of fatal soft errors in Los Alamos National Laboratory's ASC Q supercomputer. *Device and Materials Reliability, IEEE Transactions on*, 2005.

[42] Shubhendu S. Mukherjee, Christopher Weaver, Joel Emer, Steven K. Reinhardt, and Todd Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 29–, Washington, DC, USA, 2003. IEEE Computer Society.

[43] Prashant J. Nair, David A. Roberts, and Moinuddin K. Qureshi. Citadel: Efficiently protecting stacked memory from large granularity failures. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, pages 51–62, Washington, DC, USA, 2014. IEEE Computer Society.

[44] Prashant J. Nair, David A. Roberts, and Moinuddin K. Qureshi. Faultsim: A fast, configurable memory-reliability simulator for conventional and 3d-stacked systems. *ACM Trans. Archit. Code Optim.*, 12(4):44:1–44:24, December 2015.

[45] Prashant J. Nair, Vilas Sridharan, and Moinuddin K. Qureshi. XED: Exposing On-Die Error Detection Information for Strong Memory Reliability. *ISCA*, 2016.

[46] Ajeya Naithani, Stijn Eyerman, and Lieven Eeckhout. Reliability-Aware Scheduling on Heterogeneous Multicore Processors. In *HPCA*, 2015.

[47] Eugene Normand. Single Event Upset at Ground Level. *IEEE Transaction on Nuclear Science*, 1996.

[48] ORNL. Oak Ridge 'Jaguar' Supercomputer is World's Fastest. *Oak Ridge National Laboratory*, 2009.

[49] J. T. Pawlowski. Hybrid memory cube: Breakthroughdram performance with a fundamentally re-architected dram subsystem. Hot Chips, 2011.

[50] Andreas Prodromou, Mitesh Meswani, Nuwan Jayasena, Gabriel H. Loh, and Dean M. Tullsen. MemPod: A Clustered Architecture for Efficient and Scalable Migration in Flat Address Space Multi-Level Memories. *HPCA*, 2017.

[51] Moinuddin K. Qureshi and Gabe H. Loh. Fundamental Latency Trade-off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design. In *MICRO*, Washington, DC, USA, 2012. IEEE Computer Society.

[52] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. Dram errors in the wild: A large-scale field study. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 193–204, New York, NY, USA, 2009. ACM.

[53] Charles F. Shelor and Krishna M. Kavi. Moola: Multicore cache simulator. In *30th International Conference on Computers and Their Applications CATA-2015*, 2015.

[54] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. *SIGOPS Oper. Syst. Rev.*, 36(5):45–57, October 2002.

[55] Premkishore Shivakumar, Michael Kistler, Stephen W. Keckler, Doug Burger, and Lorenzo Alvisi. Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, DSN '02, Washington, DC, USA, 2002. IEEE Computer Society.

[56] Jaewoong Sim, Alaa R. Alameldeen, Zeshan Chishti, Chris Wilkerson, and Hyesoon Kim. Transparent hardware management of stacked dram as part of memory. In *MICRO*, Washington, DC, USA, 2014. IEEE Computer Society.

[57] Jaewoong Sim, Gabriel H. Loh, Hyesoon Kim, Mike O'Connor, and Mithuna Thottethodi. A Mostly-Clean DRAM Cache for Effective Hit Speculation and Self-Balancing Dispatch. In *MICRO*, Washington, DC, USA, 2012. IEEE Computer Society.

[58] Jaewoong Sim, Gabriel H. Loh, Vilas Sridharan, and Mike O'Connor. Resilient die-stacked dram caches. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 416–427, New York, NY, USA, 2013. ACM.

[59] Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, Kurt B. Ferreira, Jon Stearley, John Shalf, and Sudhanva Gurumurthi. Memory errors in modern systems: The good, the bad, and the ugly. *SIGARCH Comput. Archit. News*, 43(1):297–310, March 2015.

[60] Vilas Sridharan and Dean Liberty. A study of dram failures in the field. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 76:1–76:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[61] John R Tramm, Andrew R Siegel, Tanzima Islam, and Martin Schulz. XSBench - The Development and Verification of a Performance Abstraction for Monte Carlo Reactor Analysis. In *PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future*, Kyoto.

[62] Ashish Venkat, Sriskanda Shamasunder, Hovav Shacham, and Dean M. Tullsen. Hipstr: Heterogeneous-isa program state relocation. *SIGPLAN Not.*, 51(4):727–741, March 2016.

[63] Ashish Venkat and Dean M. Tullsen. Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 121–132, Piscataway, NJ, USA, 2014. IEEE Press.

[64] Kristen R. Walcott, Greg Humphreys, and Sudhanva Gurumurthi. Dynamic prediction of architectural vulnerability from microarchitectural state. *SIGARCH Comput. Archit. News*, 35(2):516–527, June 2007.

[65] Wm. A. Wulf and Sally A. McKee. Hitting the Memory Wall: Implications of the Obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, March 1995.

[66] Jun Xu, Shuo Chen, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. An experimental study of security vulnerabilities caused by errors. In *2001 International Conference on Dependable Systems and Networks (DSN 2001) (formerly: FTCS), 1-4 July 2001, Göteborg, Sweden, Proceedings*, pages 421–432, 2001.

[67] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, and B. Chin. IBM Experiments in Soft Fails in Computer Electronics (1978&Ndash;1994). *IBM J. Res. Dev.*, 1996.