HaRMony: Heterogeneous-Reliability Memory and QoS-Aware Energy Management on Virtualized Servers

Konstantinos Tovletoglou
ktovletoglou01@qub.ac.uk
Queen's University Belfast, UK

Lev Mukhanov
l.mukhanov@qub.ac.uk
Queen's University Belfast, UK

Dimitrios S. Nikolopoulos
dsn@vt.edu
Virginia Tech, USA

Georgios Karakonstantis
g.karakonstantis@qub.ac.uk
Queen's University Belfast, UK

Abstract

The explosive growth of data increases the storage needs, especially within servers, making DRAM responsible for more than 40% of the total system power. Such a reality has made researchers focus on energy saving schemes that relax the pessimistic DRAM circuit parameters at the cost of potential faults. In an effort to limit the resultant risk of critical data disruption, new methods were introduced that split DRAM into domains with varying reliability and power. The benefits of such schemes may have been showcased on simulators but have neither been implemented on real systems with a complete software stack, nor have been combined with any energy-reliability OS management policies. In this paper, we are the first to implement and evaluate HaRMony, a heterogeneous-reliability memory framework, in conjunction with QoS-aware energy management policies on a server with a complete virtualization stack. HaRMony overcomes the practical restrictions stemming from default hardware specifications, which were neglected in prior works, by introducing a software-based memory interleaving scheme. Furthermore, we expose the capabilities of HaRMony to the QEMU-KVM hypervisor through two unique policies. The first policy enables the hypervisor to seek the most power efficient DRAM circuit parameters based on the server availability requested by the user. The second policy enables users to exploit the inherent application error-resiliency by allowing them to limit the error protection mechanisms and allocate data structures on variably-reliable memory domains. Our evaluation shows that HaRMony reduces the performance overhead incurred due to disabling hardware interleaving from 29.3% down to 1.1% and leads to 17.7% DRAM energy savings and 8.6% total system energy savings on average in case of native execution of 28 benchmarks on an ARMv8-based server. Finally, we demonstrate that our QoS-aware scaling governor integrated with QEMU-KVM can dynamically scale the DRAM parameters, while reducing the system energy by 8.4% and meeting the targeted QoS even under extreme temperatures.

ACM Reference Format:
Konstantinos Tovletoglou, Lev Mukhanov, Dimitrios S. Nikolopoulos, and Georgios Karakonstantis. 2020. HaRMony: Heterogeneous-Reliability Memory and QoS-Aware Energy Management on Virtualized Servers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20), March 16–20, 2020, Lausanne, Switzerland.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3373376.3378489

1 Introduction

The rapid increase of virtualized Cloud services and the exponential growth of generated data [7] drives the need for higher memory density and bandwidth in common and emerging Edge servers. As a result, recent projections forecast that the DRAM subsystem will soon be responsible for more than 40% of the overall power consumption within most multicore systems [24], highlighting the need for new energy-efficient memory schemes. However,reducing the DRAM energy consumption is challenging due to the conventional adoption of pessimistic DRAM circuit parameters. In fact, DRAM manufacturers, in an effort to limit the potential faults, adopt a high *operating supply voltage* (V_{DD}) and *refresh rate* (T_{REFI}), which are selected according to the assumed rare worst-case conditions [10]. Such an approach might guarantee error-free storage of data, but the incurred power and performance overheads raise doubts about its viability, especially as the spread in cell retention times within- and across- DRAM chips increases. Such a challenge has attracted the interest of many studies, the majority of which proposed to relax T_{REFI} for the cells with high retention times [4, 5, 41, 42, 55]. Nonetheless, prior

ASPLOS '20, March 16–20, 2020, Lausanne, Switzerland

ACM ISBN 978-1-4503-7102-5/20/03…$15.00
https://doi.org/10.1145/3373376.3378489

studies [57] revealed that the retention time of each cell varies dynamically [54, 57], which limits the effectiveness of the above schemes since the error-free properties of the cells cannot be guaranteed. To address any potential memory errors, schemes like *error detection and correction codes* (ECC) [40, 51, 71] or *checkpoint and restart* (CR) mechanisms [21, 30] could be deployed, however such schemes incur power and performance overheads.

Alternative studies [20, 44, 55, 66] tried to limit these overheads by exploiting the *inherent error resilient properties* of various applications and storing only few *critical data* structures in well protected arrays operating under the nominal parameters, that are consuming increased power, while allowing resilient/non-critical data to be stored on low energy, less reliable arrays [45, 68]. A scheme [44], that gained a lot of attention in this context, splits the memory space into a region operated under nominal circuit parameters and a region operated under relaxed parameters which may be more prone to errors but consumes less energy.

The benefits of *Heterogeneous-Reliability Memory* (HRM) have been showcased on simulators [34, 44] but have never been implemented and evaluated on real systems with a complete virtualization stack. As a result, previous studies have not considered fundamental architectural specifications that may impede the realization of HRM in real systems. In fact, in modern DRAM subsystems, *Hardware Memory Interleaving* (HardInterlv) is typically being used for distributing consecutive memory accesses across multiple memory channels and thus improving performance. Such a mechanism essentially hinders the implementation of the HRM framework as it does not allow splitting of memory into regions with varying reliability, thus making all existing related works practically unrealistic if HardInterlv is not addressed properly. Furthermore, it is not yet clear how to automatically manage the energy-reliability trade-offs enabled by HRM to meet a specified *Quality of Service* (QoS) and how to expose such a framework to users, especially in real virtualized systems. Prior works [44, 45] rely mainly on user-guided management of the energy-reliability trade-offs in non-virtualized systems without any dynamic adaptation of DRAM circuit parameters and QoS guarantees. Some works [45, 55] may have sketched possible software support for HRM, but have never implemented and evaluated any governor that automatically scales DRAM operating parameters.

In this paper, we present HaRMony, the first practical implementation of HRM on actual servers with a complete software stack. We provide fully functional HRM domains in the Linux kernel and make them transparently available in the KVM/QEMU virtualization stack. We overcome the challenge of servers lacking hardware support to allocate and manage HRM by a new software-defined memory interleaving scheme. Our scheme enables effective memory power and reliability management at the granularity of DIMMs with minimal performance overhead.

Our contributions can be summarized as follows:

- We quantify the cost of disabling hardware interleaving on a real server showing that it can lead to an execution time overhead of up-to 228%. To address this challenge, we implement HaRMony, an HRM framework, that enables the selective allocation of data on variably-reliable domains. HaRMony utilizes a novel software-based memory interleaving technique, which limits the performance overhead caused by disabling HardInterlv.
- We provide a new and intuitive API for using HRM during both bare-metal execution in Linux and virtualized execution in the QEMU-KVM hypervisor.
- We present two novel policies to manage energy-QoS trade-offs using HRM on virtualized servers: An automated policy which is transparent to the user; and a user-guided policy. We present a first of its kind governor that can transparently and dynamically adjust the DRAM circuit parameters at varying temperatures to maximize energy savings, while meeting the QoS targeted by the user.
- We present a new experimental memory thermal testbed that allows us to evaluate the efficiency of HaRMony operating various temperatures.
- We evaluate the energy-QoS management policies and governor in terms of their performance overhead and energy savings using a wide range of applications from several domains. Our study shows that HaRMony decreases the average performance overhead of a naive HRM implementation from 29.3% down to 1.1% during bare-metal execution of 28 SPEC CPU2006 benchmarks and down to 6.1% in 10 virtualized workloads. Finally, HaRMony saves on average 8.6% and 8.4% of the system energy for native and virtualized executions, respectively. Such savings are shown for the first time on a real server and substantially exceed the savings (about 1%) demonstrated in previous simulator-based studies [44].

The rest of the paper is organized as follows. Section 2 describes the DRAM background and related challenges, while Section 3 presents the proposed framework. Section 4 presents the implementation of HRM on a real server and discusses the evaluated workloads. Section 5 analyses the evaluation results. Section 6 discusses related research studies. Finally, conclusions are drawn in Section 7.

2 Background and Challenges

2.1 DRAM Basics

DRAM Organization. A main memory subsystem based on DRAM is organized into *memory channel units* (MCUs), as shown in Figure 1. MCUs support a number of DRAM modules, i.e., *Dual In-line Memory Modules* (DIMMs). Each DIMM consists of ranks, usually two. Ranks house a number

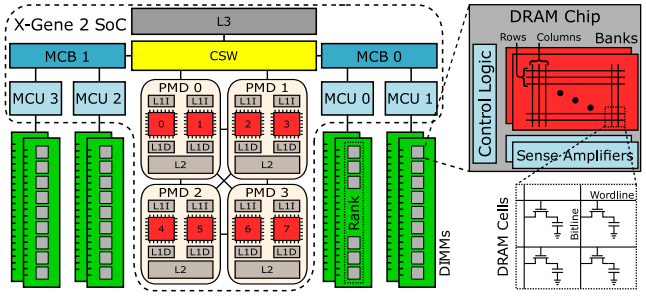


Figure 1. AppliedMicro X-Gene 2 processor block diagram and internal structure of the DRAM hierarchy.

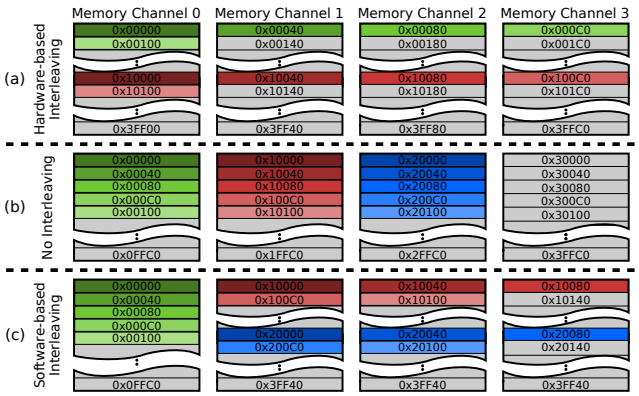


Figure 2. Allocations on a system with *(a)* HardInterlv; *(b)* HardInterlv disabled and *(c)* SoftInterlv.

of banks each of which is a two-dimensional array composed of cells. DRAM cells consist of an access transistor and a capacitor that holds the data in the form of electric charge.

DRAM Circuit Parameters. Each capacitor of the DRAM cells leaks electric charge through the transistor and eventually may lead to a complete loss of the stored information. The period during which the information stored in a DRAM cell can be retrieved correctly is called the retention time of that cell. To maintain the integrity of the data stored in cells, the charge of each capacitor must be periodically refreshed.

A refresh mechanism is employed to regularly replenish the charge by issuing a refresh command. During the refresh DRAM is not available, leading to both performance and power penalties even when the memory is idle. Conventionally, the refresh rate [32] of each row in DDR3 DRAMs is set by manufacturers to T_{REFI} = 64 *ms*. Such a T_{REFI} is set based on the minimum retention time of all the cells, which is estimated based on assumed worst-case operating conditions. However, recent studies [37, 42] have shown that only a very small number of cells needs to be refreshed that often.

Interleaving. To improve the memory performance, modern servers implement memory interleaving [22, 74]. Interleaving enables the uniform distribution of consecutive accesses across different MCUs, maximizing the memory bandwidth. In modern servers, interleaving is implemented at the hardware level (HardInterlv), so that the bandwidth of all the channels can be utilized concurrently. Figure 2a illustrates how physical memory addresses are distributed across the MCUs when HardInterlv is enabled in a system with 4 MCUs. We see that the physical addresses of each allocation, e.g. `0x00000-0x00100` (colored in green), are spread uniformly across all MCUs enabling the parallel use of the channels and minimizing the number of channel conflicts.

2.2 Heterogeneous-Reliability Memory

A popular scheme to allow operation under relaxed DRAM circuit parameters is HRM, where the memory is split into two domains: *a reliable domain* which contains DIMMs operating under nominal parameters and *a variably-reliable domain* which operates under relaxed (low power but prone to errors) parameters [44]. In such a scheme, data can be allocated on the different reliability domains depending on their critically (usually specified by users). Particularly, the reliable domain is used to store the critical data, such as the OS data or application data that cannot tolerate errors. The critical data are allocated in this domain, since any error that occurs in such data may result in an abnormal program behavior or even a system crash. While less critical data can be allocated in the variably-reliable domain, where DIMMs operate under relaxed parameters, and thus reduce the DRAM power. However, the user should ensure that errors in these data are handled by software and will not result in catastrophic program failures.

Recent schemes [26, 44] evaluated HRM in simulators but have never investigated if it is possible to implement it on a real system with a fully-fledged OS. In fact, many of the proposed approaches [14, 23, 40, 42] require modifications of the memory controller, which deem them impractical [39, 48].

Challenge I: Hardware-based Interleaving. Furthermore, HardInterlv poses a challenge to the implementation of HRM since all data are distributed across MCUs when it is enabled. Thereby, due to HardInterlv, the critical data are also distributed across all MCUs, even those that operate under relaxed DRAM parameters. As a result, the integrity of the critical data may be compromised, which may lead to system or program crashes. To address this challenge, each MCU should have a separate, distinguishable, address space, which implies that HardInterlv must be disabled.

However, disabling HardInterlv (*NonInterlv*) may result in a performance degradation for memory-bound applications. In fact, our evaluation of the NonInterlv configuration showed that such an overhead can reach up-to 228%. This overhead can be attributed to the decreased utilized

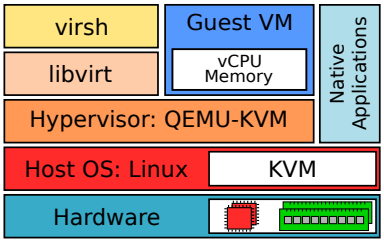


Figure 3. The QEMU-KVM virtualization stack.

memory bandwidth as the MCUs are not utilized in parallel. For example, Figure 2b illustrates how the physical addresses are distributed across MCUs for the NonInterlv configuration. Specifically, we see that all the addresses from `0x00000-0x00100` range (colored in green) are assigned to a single MCU, as opposed to the HardInterlv configuration. Thus, the memory bandwidth in this case is limited by the maximum bandwidth of one MCU compared to the four MCUs that are available when HardInterlv is enabled.

2.3 Operating System and Virtualization

In modern cloud-based data centers, the cores and memory are virtualized to enable running *virtual machines* (VMs) and shared between several users. A typical virtualization stack consists of the underlying hardware, the host *operating system* (OS), the virtualization framework (e.g. KVM and QEMU) and management tools (e.g. libvirt), as shown in Figure 3.

All state-of-the-art servers supporting virtualization use hypervisors to create, run and coordinate VMs. Linux OS integrates virtualization capabilities through a module, which allows the Linux kernel to operate as a hypervisor, the most popular of which is KVM [11]. QEMU [3] runs jointly with KVM and orchestrates the management of VMs, creating of virtual devices, checkpointing and restarting of VMs.

Libvirt is a virtualization management library, which is used with both QEMU-KVM and a range of hypervisors. It consists of three utilities namely - an API library, a daemon (libvirtd) and a command line tool (virsh). It enables seamless integration of the virtualization with the hypervisor, allowing to mask the underlying hardware for any target.

Challenge II: Integration with the software stack. To simplify integration of the HRM frameworks with the software stack, an intuitive programming interface for working with HRM needs to be introduced. Such an interface should expose low-level functions to the user and extend the OS to enable data allocation on different reliability domains.

2.4 QoS: Reliability and Availability

It is expected that applications and VMs run on error-free nodes in data centers. However, failures may manifest due to various sources [47, 60], which make a node unavailable for some period of time. It is evident that in HRM, where V_{DD} or T_{REFI} are relaxed for one of the memory domains, DRAMs are more prone to failures. Thereby, an application or VM which has allocated data on a memory domain operating under relaxed parameters may result in a wrong output or even a crash when affected by a DRAM error.

Typically, manufacturers measure the reliability of hardware using the *Mean Time Between Failures* (MTBF). While the node downtime due to a failure is quantified as the *Mean Time to Recover* (MTTR). We use these metrics to estimate the availability or QoS of a VM:

$$\text{QoS(VM)} = 1 - \frac{\text{MTTR}}{\text{MTBF}} \tag{1}$$

Both MTTR and MTBF depend on a type of the running workload or VM, while MTBF is affected by the inherent hardware reliability. Public cloud providers use *Service Level Agreements* (SLAs) to ensure a desired availability guarantee for each VM. Essentially, the SLA defines the maximum downtime of the provided cloud service for a certain period of time. For example, operators of data centers providing a 99.9% SLA need to ensure that the downtime of each server is less than 43 minutes per month, while for data centers with a 97% SLA, the acceptable downtime is about 21 hours.

Challenge III: Runtime management of the Energy-QoS trade-offs. To ensure the seamless operation of a server and satisfy the cloud user's requested QoS, there is a need for new runtime QoS-aware management schemes that scale the DRAM circuit parameters to improve energy efficiency, while handling errors that may compromise availability of VMs.

Such schemes should be designed to adjust the DRAM settings considering the different operating conditions (e.g. temperature) that may affect DRAM reliability. To meet the targeted QoS and provide a guarantee for the SLA, the HRM framework shall also be combined with error handling mechanisms, such as ECC to correct a number of errors [40, 51, 71] or CR mechanisms for VMs [21, 30] to recover a server in case of a crash. However, such schemes have never been evaluated jointly within a real server under varying operating conditions. This evaluation is essential since some error handling mechanisms, such as CR, may significantly degrade the availability of a server and thus QoS if not configured properly.

Challenge IV: Experimental Evaluation. Relaxing DRAM parameters may result in an increased number of errors, especially in case of high DRAM temperatures. Therefore, any attempt to develop the mechanisms discussed above requires an evaluation of the error behavior for DRAM operating under various temperatures on a real server.

In the past, experimental FPGA setups were used to characterize the DRAMs under different temperatures and T_{REFI} by using thermal chambers [33, 41]. However, such setups cannot be used to evaluate DRAM reliability while running a complete virtualization stack. Moreover, a server cannot

be tested within a thermal chamber, as in the case of an FPGA board, due to numerous components that are sensitive to high temperatures. Therefore, there is a need to build a custom thermal server testbed to evaluate the discussed mechanisms.

3 Proposed HRM System Design

In this section, we present HaRMony and its software-based interleaving scheme that reduces significantly the performance overhead incurred due to disabling HardInterlv (Challenge-I). We also introduce the interface for exposing HRM capabilities to the user within a modern software stack (Challenge-II), as well as the automated and user-guided policies for energy-reliability management that we integrate with HaRMony.

3.1 Software-based Interleaving

Operating systems utilize virtual addresses for data allocation, which are translated to physical addresses when the accesses are forwarded to the main memory. The MCU translates the requested physical addresses to the location in a specific DIMM using an interleaving function. Note that after disabling HardInterlv, the OS handles the allocation process and can use the bandwidth available from multiple MCUs by allocating pages across the MCUs. This way, we can still enable memory interleaving at the software level by mapping the continuous virtual address space to the segments of physical addresses corresponding to different MCUs. We call this scheme *Software-based Memory Interleaving* (SoftInterlv). SoftInterlv allows on-the-fly selection of the interleaving function and the number of MCUs that will be interleaved for each allocation. Note that the minimum stride available for applying interleaving in the memory management of the Linux OS is at the page level.

Figure 2c presents an example of a system with 4 MCUs where HardInterlv is disabled. In this figure, we illustrate how consecutive virtual addresses, `0x00000-0x00100` (colored in green), are mapped to memory from one MCU. When we enable SoftInterlv the allocated data, for example, the data pointed by the addresses from `0x10000-0x10100` range (colored in red) depicted in Figure 2c, are distributed across 3 MCUs. This way, we increase the memory bandwidth since the memory accesses are handled in parallel. Such a mechanism addresses Challenge-I and reduces the performance overhead incurred when we disable HardInterlv.

Linux Kernel Modifications. To implement the proposed SoftInterlv, we extend the *Non-Uniform Memory Access* (NUMA) interface that is available in Linux OS. Systems with multiple sockets may have varying memory access time, which depends on the NUMA distance between the socket and an accessed MCU. In such systems, a NUMA domain is a group of DRAM modules connected to the same socket. We use fake NUMA domains and bind each of them with a separate MCU, even though the memory access time is identical for each server core. The latest Linux kernel already exposes an interface to create fake NUMA domains and assign cores and ranges of memory to such domains. To enable this on the ARMv8 architecture used in our study, we ported the code from the x86-64 architecture and made the necessary changes to suit the ARM architecture.

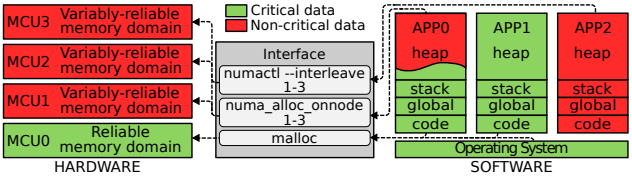


Figure 4. The scheme of data allocation in HRM.

We introduce a fake NUMA domain for each MCU, which is assigned a specific range of memory addresses. In our framework, the NUMA domains are associated with one of the reliability domains either the *Reliable Memory* (RelMem) or the *Variably-reliable Memory* (VarMem) domain. The MCUs that operate under nominal parameters are part of the RelMem domain, while the MCUs operating under relaxed parameters are part of the VarMem domain. By default, applications allocate data in the first NUMA domain that we always assign to the RelMem domain.

When memory allocations fill up the capacity of the memory of the RelMem domain, we set the parameters on one MCU belonging to the VarMem domain to the nominal ones. By doing so, we can dynamically increase the capacity of the RelMem domain depending on the requirements. Note that even though previously the parameters of the VarMem MCU were relaxed, the subsequently allocated data will be reliably stored in this MCU after reinstating the nominal ones.

Our framework can be adopted in most servers that have a separate memory address space for each memory channel or servers where memory channel interleaving can be disabled. HaRMony can be also used in multi-socket systems, since it does not restrict the use of the original NUMA interface.

3.2 Allocation Interface

By default, the Linux kernel and every application, which uses the standard system allocation functions, i.e., `malloc`, allocate data in RelMem, as depicted in Figure 4 for *APP 1*. This is achieved by the NUMA distances, where the RelMem domain appears as a local domain, i.e., small NUMA distance, to the processors and the VarMem domain as a remote memory domain, i.e., high NUMA distance. In this way, the default NUMA policy for allocations is that all allocations are targeting the local/RelMem domain, unless explicitly requesting memory from the remote/VarMem domain. To provide this capability in HaRMony, the user can explicitly request memory from a specific domain for a whole application through

the NUMA interface using the `numactl` command. Particularly, the parameter `---membind` can be passed to `numactl` to define the reliability domain that will be used to allocate data. In Figure 4, *APP 2* is an example of such an allocation, where all the application data is assigned to the VarMem domain.

We enable SoftInterlv for the memory in HaRMony by extending the `numactl` command with a new parameter `---interleave`. Thus, we can utilize the memory channel parallelism and exploit the bandwidth of multiple MCUs, similar to HardInterlv. We can enable SoftInterlv across all the MCUs or only across the MCUs that belong to the VarMem domain. The first option will allow us to achieve the highest utilization of memory parallelism. While for the second option, the reliable MCUs are used only to store critical data and are not bloated with data from the VarMem domain.

To choose the memory domain for a specific data allocation within an application, we extend the standard memory allocation interface, i.e., `malloc`, with a new interface, `numa_alloc_onnode`, that allocates data in a particular NUMA domain. Similar to `numactl`, we pass a parameter in `numa_alloc_onnode` to enable SoftInterlv across MCUs. The programmer can manually allocate less-critical data structures in the VarMem domain using this interface by modifying the source code of an application. Figure 4 illustrates such an allocation, where *APP 0* allocates a part of the heap data in the VarMem domain.

To use HRM, the programmer should make an additional analysis of workloads to identify the criticality of the data structures and select a reliability domain where to allocate these structures. Previously, several works [58] have analyzed the error resiliency of various workloads and introduced automated methods to classify the error resiliency of data structures using analytical models [45, 68]. We clarify that the analysis of the data criticality is out of the scope of this paper. Our framework enables the evaluation of the techniques proposed in the previous studies on a real server and provides a unique realistic interface which exposes the capabilities of the HRM framework to users (partly addressing Challenge-II).

3.3 HRM and Error Handling Policies

As we discussed in Section 2.4, intelligent error handling mechanisms should be implemented in a server to maintain a high QoS. In our study, we utilize and evaluate three different mechanisms at the hardware and software layers. We utilize the hardware ECC as a first error handling mechanism, which is typically available in server-grade DRAMs. There are different types of ECC but the vast majority of DRAMs implement a *single-error-correction, double-error-detection code* (SECDED). In systems that utilize such DRAMs, the Linux system logging process records all errors, either correctable or uncorrectable, detected by ECC and provides information about a specific memory location where an error occurred:

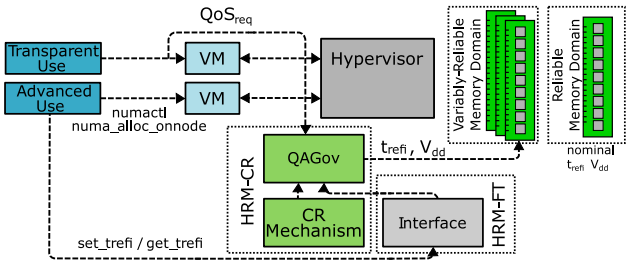


Figure 5. QoS-aware management of DRAM settings.

specifically, the MCU, rank, bank, row and column. This information enables us to identify which VM(s) have been affected by a manifested error by tracking down the memory error location and the memory domain in which each VM has allocated its data.

In HaRMony, ECC provides sufficient protection against errors in the data allocated in the RelMem domain, minimizing the possibility of system/VM crashes. However, the DIMMs assigned to the VarMem domain, operating under relaxed circuit parameters, are prone to more errors, especially double/multiple-bit errors which usually cannot be corrected by ECC. Therefore, to ensure high QoS for VMs allocating data in the VarMem domain, it is essential to implement advanced error handling mechanisms in addition to ECC.

Note that the default policy of Linux OS in case of an uncorrectable error is to shut down the entire system to avoid propagation of the error to the critical OS data. Such a policy results in a solid overhead since the system should be restarted each time an uncorrectable error is detected, even when this error has been obtained in the data allocated by a running application or VM. Since we are interested in exploiting ways to limit the conventional overheads, we change the default policy and allow the hypervisor to continue executing the error-free VMs, while interrupting execution of the VMs that have been affected by uncorrectable errors.

To further limit the overheads associated with restarting the VMs affected by errors, we implement a *checkpoint and restart* (CR) mechanism to recover VMs from an intermediate checkpoint. We call such an automated policy, that is transparent to users, HRM-CR. We also introduce an alternative more aggressive policy, HRM-FT, which enables users to exploit the inherent fault tolerant properties of applications. Using HRM-FT, advanced users can disable the automatic restarting of VMs when errors occur and thus avoid paying any recovery cost. Such a user-guided policy requires users to classify the data (critical or less-critical/error-resilient) [68] and choose which data to allocate in a specific memory domain. Note that any other complementary resilient scheme can be chosen to be implemented [45].

Below we provide details of an OS governor and both policies implemented on our HRM framework. To the best of

our knowledge, we are the first to develop a governor with fully-automated and user-guided fault resilience policies on a real HRM system which provides the best trade-off between the DRAM energy efficiency and QoS.

3.3.1 HRM-CR Policy:

Recovery Mechanism. Our CR mechanism stores the state of a VM with a defined period in a reliable storage. However, such a mechanism may incur a significant performance overhead if not configured properly, since VM data and states need to be stored in a disk or transferred over network. Thus, it is necessary to find the optimal checkpointing interval, which minimizes the total overhead.

The Young/Daly formula [16, 72] is well established for identifying the *optimal checkpointing interval* (τ_{opt}). It has been verified, both theoretically and practically, and proven to provide the optimal checkpoint interval. The basic assumption of the formula is that the checkpoint duration and checkpoint intervals are constant throughout the execution. The τ_{opt} can be calculated using MTBF from Equation 2 and *time required to create a checkpoint* (δ).

$$\tau_{opt} = \sqrt{2 \times \delta \times \text{MTBF}} \tag{2}$$

If a CR mechanism is enabled for a VM, then the availability function of the VM (Equation 1) should be extended to take into consideration the fraction of the time that the system spends on checkpointing, as shown in Equation 3.

$$\text{QoS(VM)} = 1 - \frac{\text{MTTR}}{\text{MTBF}} - \frac{\delta}{\tau_{opt}} = 1 - \frac{\text{MTTR}}{\text{MTBF}} - \sqrt{\frac{\delta}{2\ \text{MTBF}}} \tag{3}$$

In our study, we use the QEMU hypervisor that implements CR functions. QEMU provides built-in functions, i.e., `savevm` and `loadvm`, to store and restore the state of a VM. Figure 5 presents the scheme of integration of the HRM-CR mechanism with the hypervisor. We keep two checkpoint instances taken at different times and swap them to have a working checkpoint in case QEMU crashes during the checkpointing process. If an uncorrectable error or crash has been obtained, we recover the VM state using the most recent checkpoint. During the checkpointing/restarting process, the node is not available to the user. The time which this process takes is accounted as the downtime[35].

QoS-Aware DRAM Scaling Governor. To make the policy transparent to the user, we implement a *QoS-aware DRAM scaling governor* (QAGov). In order to satisfy the requested SLA, QAGov measures regularly the downtime for each VM. If the downtime of a VM has exceeded the limit set for this VM and led to the violation of the SLA, then the DRAM circuit parameters of the VarMem domain are changed to less relaxed values (see Figure 5). QAGov regularly adjusts the DRAM circuit parameters to maximize energy savings, while satisfying the SLA specified by each user when running a VM.

$$T_{REFI} = \begin{cases} 64\ ms\ , & \text{initially} \\ T_{REFI} + F_{inc} \times \frac{T_{REFI\_MAX} - T_{REFI}}{T_{REFI\_MAX}}\ , & \text{QoS(VM)} > \text{QoS}_{\text{req}} \\ T_{REFI} \times F_{dec}\ , & \text{QoS(VM)} \leq \text{QoS}_{\text{req}} \end{cases} \tag{4}$$

Equation 4 defines how T_{REFI} is scaled during the execution of VMs to meet the availability requested by the user, QoS_{req}. F_{inc} and F_{dec} denote the scaling factors of T_{REFI} when it is increased (F_{inc}) or decreased (F_{dec}).

To improve the governor T_{REFI} scaling policy, we limit the maximum value of relaxed T_{REFI}, T_{REFI_MAX}, based on the measured DRAM temperature which significantly affects the retention time of cells [29]. In particular, we set T_{REFI_MAX} using Equation 5. This equation follows the well known exponential model that estimates the DRAM cell retention time as a function of temperature [29].

$$T_{REFI\_MAX} = 32 \times e^{-0.051 \times \theta} \tag{5}$$

The governor updates the considered DRAM parameters (e.g. T_{REFI}) based on the measured QoS and DRAM temperature regularly within a 5-minute period and after each VM restart to ensure that the QoS is not violated.

Note that any other varying condition that may affect DRAM reliability, e.g. aging, could be considered in the governor by monitoring the rate of correctable errors [60, 64], which is a separate complementary topic for future work.

3.3.2 HRM-FT Policy.

To address Challenge-II, we extend the NUMA interface to support HRM. We are the first to develop such an interface for a real HRM system, which is based on the ARMv8 architecture where the NUMA interface was not available by default in the latest Linux version.

At the initialization phase of a VM, the host OS exposes the available memory size of the reliability domains to the cloud users and let them select the size of reliable and variably-reliable memory domains. The user has access to the DRAM parameters via a new interface that can be used in a VM, `set_trefi` and `get_trefi`. Figure 5 shows the interfaces that are available for the *advanced use* of HRM-FT. Note that the `set_trefi` does not directly set the T_{REFI} for a server, the hypervisor chooses the T_{REFI} for the VarMem domain that is closer to the nominal settings among all the settings requested by users for this server. In this way, each VM allocating data in VarMem meets the requested QoS.

4 Experimental Evaluation Setup

In this section, we discuss the details of our novel experimental evaluation setup, addressing Challenge-IV.

4.1 Implementation on a Server

Server Platform. To enable our study, we implement the NonInterlv configuration and HaRMony on a commodity

Table 1. Specifications of the ARMv8-based X-Gene 2 server

Description	Value
# of Cores / Core Frequency / LLC	8 / 2.4 GHz / 8 MB
Last Level Cache	8 MB (shared)
# of Memory Controllers	4
Memory Size	8 GB/MCU, Total 32 GB
Memory Type	DDR3-1866, SECDED ECC
Memory Characteristics	2 rank/MCU, 8 banks/rank, f64 K rows/bank, 64 B cache line
Maximum Core Power	34 W
Maximum Memory Power	10.5 W
Percentage of Memory Power	23.6%

server featuring the *AppliedMicro/Ampere X-Gene 2* [63] ARMv8 processor (CPU), the specifications of which are provided in Table 1. All memory operations in X-Gene 2 are handled by 4 MCUs, which are divided in 2 groups of *Memory Controller Bridges* (MCBs), as shown in Figure 1. Each MCU can be populated with up to 2 DDR3 DIMMs running at 1866 MT/s. In our setup, we use 4 Micron DDR3 8GB DIMMs [49], one for each MCU, consisting of 72 DRAM chips. Recent evaluations [27, 28] have indicated that X-Gene 2 provides memory bandwidth similar to that of an Intel® Xeon® E5.

The X-Gene 2 server features a dedicated processor to enable power management, monitoring of sensors and configuration of system parameters. Those parameters include MCU initialization and memory operating parameters (such as T_{REFI} and V_{DD}). During the MCU initialization, we can define the level of HardInterlv, either across all 4 MCUs or across the 2 MCUs of each MCB or completely disable it. The default configuration of X-Gene 2 (as in almost all servers) is to have HardInterlv enabled across all MCUs.

The X-Gene 2 server has voltage and current sensors that are integrated on the X-Gene 2 motherboard. These sensors enables us to measure the power for the processor and each MCB. We calculate the total energy consumption of the system (E_{total}) by integrating the measured DRAM and CPU power (P_{total}) over the duration of the experiments (T) for each benchmark, as shown in Equation 6.

$$E_{total} = \int_0^T P_{total}(t)dt \tag{6}$$

Software Stack. X-Gene 2 runs CentOS 7 with the Linux kernel 4.11, which we extend to implement HaRMony, as described in the following sections. The QEMU hypervisor (version 3.0.50) available in our virtualization stack is built with support of KVM and the libvirt library (version 4.5.0).

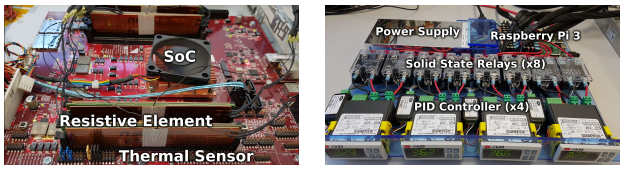
Refresh Rate. The maximum allowed T_{REFI} in the X-Gene 2 platform is 2.283 s (35× more than the nominal 64 ms) and each MCU can independently be configured. As we discuss in Section 3, the HRM-CR policy of our governor dynamically adjusts the T_{REFI} of the VarMem domain based on the specified QoS and measured DRAM temperature. In our study, we select the scaling factors $F_{inc} = 30$ and $F_{dec} = 0.7$ experimentally to slowly relax T_{REFI} and quickly recover it to the nominal values when the SLA is violated. Note that other DRAM circuit parameters (e.g. timing parameters) that were utilized by prior studies [38] can also be integrated in our governor, while this study focuses on the adjustment of T_{REFI}.

Operating Supply Voltage. V_{DD} can be controlled at the MCB level, i.e., per 2 MCUs. However, we do not adjust V_{DD} in the governor, since the scaling of V_{DD} does not affect the reliability of the server considerably. In our experiments, we reduced V_{DD} for the VarMem domain (MCB 1) from the default 1.5 V down to to 1.425 V (5% reduction). Note that this is the minimum voltage specified by the DIMM datasheet [49]. Our experiments indicated that any further reduction of V_{DD} results in an instantaneous crashing of the server. We attribute this to the fact that 1.425 V is the default voltage cutoff threshold adopted in the power regulators used in the specific board.

Temperature. To perform the experiments under controlled temperatures, we implement a unique temperature-controlled server testbed using heating elements [33]. Figure 6a shows the X-Gene 2 board with four DIMMs fitted with our custom adapters. Each adapter consists of a resistive element, with thermally conductive tape transferring the heat to all the chips of a DIMM in a uniform way, and a thermocouple to measure temperature. The temperature of each element is regulated by a controller board, as shown in Figure 6b, which contains a Raspberry Pi 3 [19], four closed-loop PID controllers [8], and eight solid-state relays controlling the resistive elements of each DIMM and rank independently.

4.2 HRM Configurations

We can configure HaRMony using different number of MCUs dedicated to the RelMem and VarMem domains. Our server has 4 MCUs, which implies that we can use 1 or 2 MCUs for the RelMem domain and 3 or 2 for VarMem, respectively. To maximize energy savings, in this study, we use the configuration of HRM where one MCU is assigned to the RelMem domain and 3 MCUs are used for the VarMem domain.



(a) (b)

Figure 6. *(a)* X-Gene 2 with the custom thermal adapters and *(b)* Temperature controller board.

Table 2. Evaluated workloads under the QEMU hypervisor.

(a) **Workloads sensitive to errors**

Workload	Domain	Memory (MB)	Execution time (min:sec)	MPKI
nw	Bioinformatics	312	1:18	11.03
kmeans	Machine Learning	277	3:15	1.87
backprop	Machine Learning	351	0:48	43.52
srad	Image Processing	393	1:38	12.27
bc	Graph Processing	9408	3:18	4.47
bfs	Graph Search	9408	1:03	4.38
pagerank	Website Ranking	176	0:39	33.29

(b) **Error-resilient workloads HRM-FT.**

blackscholes	Financial	626	2:30	0.58
x264	Video Processing	202	4:40	11.89
fluidanimate	Fluid Dynamics	72	0:18	0.79

4.3 Workloads

To evaluate the performance of HaRMony, we choose a set of workloads which stress the processor and memory in different ways. In our experiments, we use the *SPEC CPU2006 suite* [31] to quantify the impact of the disabled HardInterlv and proposed HRM on performance when running applications in a non-virtualized environment. We are using the *ref* data inputs, while running a single-threaded instance of the benchmark for each core of the 8 cores in parallel. This allow us to allocate larger memory footprints that range from 72MB up-to 13.1GB which are representative of various workloads and stress more the processor and memory. Figure 7e presents the memory size and the memory intensity of those benchmarks measured as *last-level cache misses per kilo-instructions* (MPKI).

To evaluate HRM integrated with the virtualization stack and introduced policies (HRM-CR and HRM-FT), we make experiments with two sets of VMs that execute workloads with different fault-resiliency characteristics. Table 2 presents the characteristics of these workloads, in terms of their domain, memory size, execution time and MPKI. The first set of VMs (Table 2a) is composed of four workloads from the *Rodinia benchmark suite* [12] and three workloads from the *Ligra graph processing framework* [62] that cannot tolerate errors during their execution, since memory errors in such workloads may result in application crashes. The second set of VMs (Table 2b) includes three workloads from the *Parsec benchmark suite* [6]. Each application from this set exhibits error-resilient properties in some of their data structures, as it has been shown previously [58].

Overall, the considered benchmarks are representative since they not only cover typical benchmarks, HPC and data analytics workloads, but also stress the DRAM with varying memory access rates and memory footprints [28]. Although some of the investigated benchmarks have a small memory footprint, their memory access rates and patterns (sequential, random, stride) are representative of applications executed on systems with bigger memory capacity.

Note that the dynamic DRAM power depends on the access rate of each benchmark, while the idle DRAM power depends on the system memory capacity (irrespective of the workload's memory footprint) [65]. Increasing the memory capacity will increase the idle DRAM power consumption due to the increased memory that needs to be refreshed. In this case, HaRMony will lead to even higher energy savings due to the applied relaxed refresh rate on potentially more memory. The dynamic DRAM power consumption will not change irrespectively of the memory capacity's increase (assuming that the workload and the number of memory controllers remain the same).

5 Experimental Results

In this section, we evaluate the performance, power, energy efficiency and QoS for three memory configurations of the ARMv8 server: *i)* the default HardInterlv configuration; *ii)* the NonInterlv configuration and *iii)* HaRMony.

5.1 Native Execution

We first run the SPEC CPU2006 benchmarks natively and evaluate the performance, power and energy efficiency of the aforementioned configurations. The following experiments are conducted without using the thermal testbed, thus the DRAM temperature is low (under 50°C) and the server is not affected by errors during any experiment.

5.1.1 Performance Evaluation

Figure 7a compares the execution time of benchmarks running on the server that uses the NonInterlv configuration and HaRMony. In this figure, we present the execution time of each benchmark as a percentage of the execution time of the benchmark running on the server with the baseline memory configuration, in which HardInterlv is enabled by default. Note that, in Figure 7, we use *SPEC* identification numbers to designate benchmarks. We see that the NonInterlv configuration incurs a performance overhead of 29.3% (geometric mean), which is expected as the memory bandwidth is not fully utilized. While HaRMony allows us to reduce this overhead down to 1.1% and 5.4% by enabling SoftInterlv across 4 and 3 MCUs, respectively.

Notably, the highest performance overheads are obtained for `462.libquantum` and `470.lbm`, which can reach up to 212% and 228% for the NonInterlv configuration, respectively. Similarly for HaRMony, the same benchmarks incur the highest overhead, up-to 10.2% and 8.6%, respectively, when 4 MCUs are interleaved. To investigate if the performance overhead is related to a high memory access intensity, we measure MPKI of each workload. However, we have not discovered a strong correlation between MPKI and performance

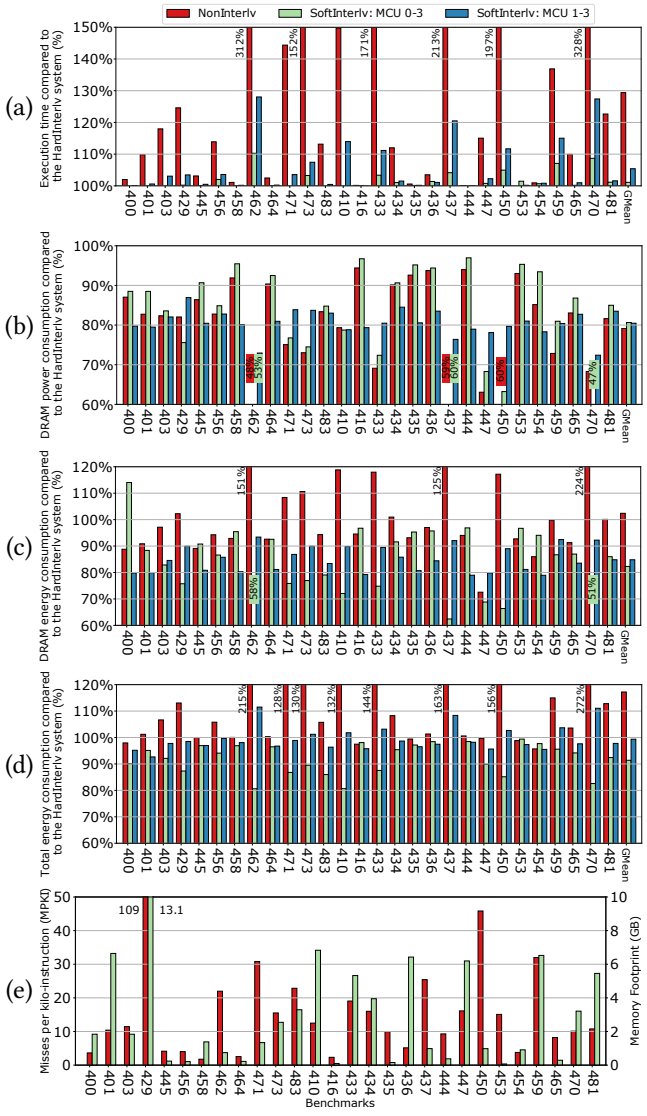


Figure 7. *(a)* The execution time, *(b)* the average DRAM power, *(c)* the DRAM energy and *(d)* the total energy (processor and memory) of the NonInterlv system and HaRMony normalized to the time/power/energy of the HardInterlv configuration for native benchmark runs. *(e)* MPKI and memory footprint measured for each benchmark.

overhead (see Figure 7e). We suggest that the high performance overhead observed for these particular benchmarks could be explained by specific memory access patterns that raise the number of bank conflicts and thus increase the average memory access latency.

5.1.2 Power and Energy Evaluation

To achieve the maximum power savings, we configure the MCUs/DIMMs assigned to the VarMem domain using the most low-power settings for V_{DD} and T_{REFI}. We lower V_{DD}

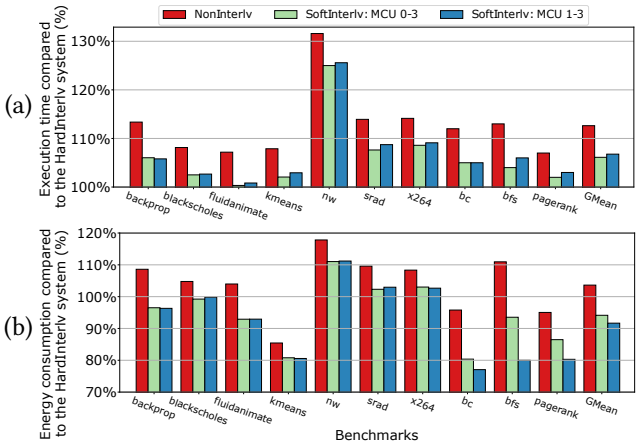


Figure 8. *(a)* The execution time and *(b)* the total energy (processor and memory) of the NonInterlv configuration and HaRMony normalized to the time/energy of the HardInterlv system for virtualized workloads.

down to 1.425 V for the second MCB (MCU 2 and 3) and increase T_{REFI} to 2.283 s, the maximum available T_{REFI}, for MCU 1, 2 and 3.

Figure 7b depicts the power of DRAM operating under scaled circuit parameters for the NonInterlv configuration and HaRMony. We present these measurements as a percentage of the memory power measured for the baseline configuration (HardInterlv is enabled) when DRAM operates under the nominal V_{DD} and T_{REFI}. We observe that the geometric mean of the DRAM power reduction is similar for HaRMony with either 3 or 4 interleaved MCUs, at 19.4% and 19.5%. The NonInterlv system has even higher DRAM power reduction at 20.9%, which we attribute to a low memory access rate obtained when there in no any memory interleaving.

Interestingly, the benchmarks that incur the highest performance overhead for NonInterlv and HaRMony (`462.libquantum` and `470.lbm`), consume also the highest DRAM power. Such results may indicate that these benchmarks have a high number of bank conflicts, which, as we assume, increase the DRAM power, since in case of a conflict a new row is opened, and thus pre-charged, the row is accessed. In other words, the performance degradation obtained when we disable HardInterlv may be explained not only by the reduction in the bandwidth but also by the varying memory access patterns.

Figure 7c shows the DRAM energy for the NonInterlv configuration and HaRMony. Similar to power measurements, we present all the energy measurements as a percentage of the energy consumed by DRAM on the server with the baseline configuration, in which HardInterlv is enabled and DRAM operates under the nominal circuit parameters. We see that the geometric mean of the DRAM energy consumption of the NonInterlv memory configuration is higher by

2.4% compared to the energy consumption of the baseline configuration. By contrast, we see that the geometric mean of the DRAM energy reduction is 17.7% and 15.2% for HaRMony with 4 and 3 interleaved MCUs, respectively, compared to the server which uses the baseline DRAM configuration. Moreover, in HaRMony with 4 MCUs interleaved only 5 benchmarks, i.e., 458, 416, 436, 444 and 453, have DRAM energy reduction less than 5%.

Figure 7d shows the total energy of the processor and memory for the NonInterlv configuration and HaRMony, similarly to Figure 7c but for the entire system (processor and memory). We observe that the geometric mean of the system energy consumption for the NonInterlv configuration is 17.2% higher, while HaRMony with 3 MCUs interleaved is 0.6% lower than the baseline system for the considered benchmarks. These results indicate that any implementation of HRM system that does not mitigate the performance overhead or that is not optimally configured will not be energy efficient. HaRMony with 4 MCUs interleaved can achieve 8.6% total system energy savings (geometric mean across benchmarks). To the best of our knowledge, we are the first to demonstrate the energy savings which can be achieved by HRM on a real server. Moreover, when HRM is properly configured these energy savings exceed by far the ones demonstrated in prior simulator-based studies (1% savings) [44].

5.2 Evaluation of HRM for virtualized workloads

To investigate the efficacy of HaRMony integrated with the QEMU hypervisor, we run the virtualized workloads presented in Table 2 on the server that uses the three configurations discussed above. Similar to the native execution of benchmarks, we are not using the thermal testbed for the performance, power and energy evaluation and no errors occurred during those experiments.

5.2.1 Performance Evaluation

Figure 8a depicts the execution times, which we present similarly to Figure 7a, of the virtualized workloads running on the server with the NonInterlv memory configuration and HaRMony. Our first observation is that SoftInterlv enables us to reduce the average performance overhead incurred due to disabled HardInterlv from 12.6%, which is obtained for the NonInterlv configuration, down to 6.1% and 6.7% when 4 and 3 MCUs are interleaved, respectively.

Overhead of the Checkpointing. To estimate the optimal checkpointing interval (τ_{opt}), we need to measure the time which is required to make a checkpoint (δ) and the *Mean Time Between Failures* (MTBF) according to Equation 3. Table 3 presents the measured δ across different workloads. We measured δ by making 4 checkpoints for each workload with different intervals, which are 1, 5, 15, 30 and 60 minutes. We found that δ is not affected by the checkpointing interval. Following this observation, for the rest of the experiments

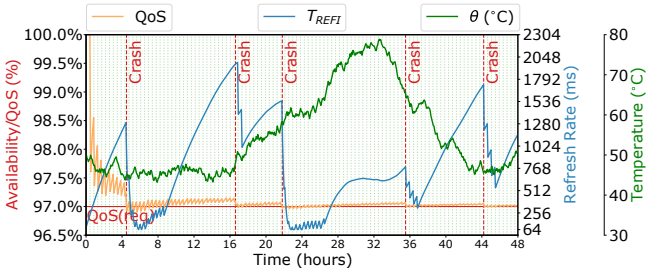


Figure 9. T_{REFI} scaled by QAGov when running a VM with the x264 benchmark at different temperatures.

we consider a constant δ for each workload. A side notice is that δ in our system is considerably high as we use hard disk drives for the storage of the checkpoint data and not solid state drives, which would provide a much faster writing speed.

As MTBF is affected by the DRAM circuit parameters, which are changed by the governor at runtime, it is hard, if not impossible, to measure it. However, Equation 3 can be solved for the worst-case/minimum MTBF that the server can tolerate if we fix the QoS at the lowest value that the user has requested, the *Mean Time to Recover* (MTTR) and δ for each VM. We measure MTTR by recovering VMs after 1, 5, 15, 30 and 60 minutes of execution. Similar to δ, we discovered that MTTR is constant for each workload and does not depend on when we start recovery. Table 3 presents the measured MTTR and calculated minimum MTBF that the server can tolerate for $\text{QoS}_{\text{req}} = 97\%$. Finally, with all the required information shown in Table 3, we estimate τ_{opt} for $\text{QoS}_{\text{req}} = 97\%$ and each workload (see Table 3). We observe that τ_{opt} varies by up to 2.5× across workloads, which mainly depends on the duration of the checkpointing process, as shown by our experiments. A system with a faster checkpointing would allow to have more frequent checkpoints at the same overhead.

Table 3. Workload-aware parameters of the HRM-CR policy.

Workload	Average δ (seconds)	Average MTTR (seconds)	Min MTBF (minutes) for $\text{QoS}_{\text{req}} = 0.97$	τ_{opt} (minutes) for $\text{QoS}_{\text{req}} = 0.97$
nw	44	54	463	26
kmeans	46	52	484	27
backprop	55	57	569	32
srad	62	56	635	36
bc	38	54	409	23
bfs	42	53	445	25
pagerank	51	52	528	30
blackscholes	30	55	334	18
x264	30	58	344	19
fluidanimate	22	52	260	14

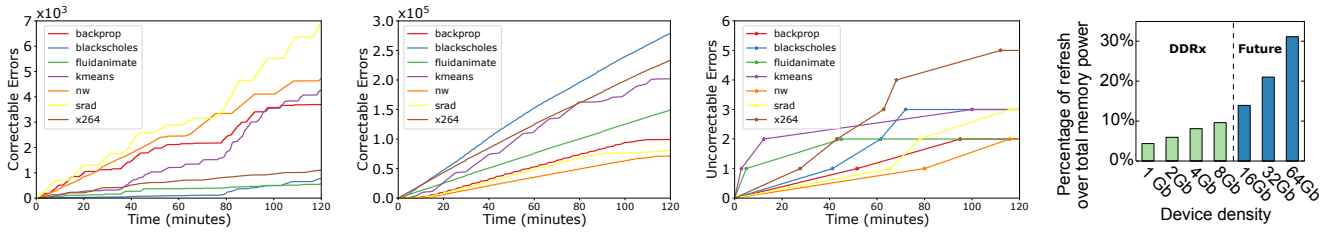


Figure 10. The number of errors manifested when using the HRM-FT policy: *(a)* correctable at 60°*C*, *(b)* correctable at 80°*C* and *(c)* uncorrectable at 80°*C*. *(d)* The percentage of the DRAM power consumed by the refresh for different densities of chips.

5.2.2 Power and Energy Evaluation

Figure 8b shows the total energy of the server (DRAM and processor) measured for the virtualized workloads when we enable the HRM-CR policy which dynamically adjusts T_{REFI}. We present all the energy measurements as a percentage of the energy consumed by the server which uses the baseline configuration, in which HardInterlv is enabled and DRAM operates under the nominal circuit parameters. We see that the geometric mean of the system energy savings is 5.9% and 8.4% for HaRMony when 4 and 3 MCUs are interleaved, respectively. We observe that HaRMony with 3 MCUs is more energy efficient for virtualized workloads, and this can be attributed to the fact that both systems have similar performance overheads for the investigated benchmarks. Thus, we conclude that HaRMony with either 4 ir 3 MCUs interleaved is effective for running virtualized workloads.

Scaling of Power and Energy Savings. We implement HRM on an energy efficient ARMv8-based server which makes the obtained energy savings even more remarkable. Although our study indicates that we can save up to 8.6% of the system energy on average, by using the HRM implemented on 1Gb DDR3, these savings will be much higher for future generations of DRAM chips. It was previously shown that the DRAM refresh power grows exponentially with the density of DRAM devices [42]. To evaluate this growth, we use the methodology presented by the manufacturer of DIMMs [65] and results presented in one of the previous works [42]. Figure 10d shows the percentage of the DRAM power spent on refresh for current and future DRAM chips. We see that the refresh power will reach almost 30% of the total DRAM power for 64Gb DRAM chips, which is almost 9× higher than the same contribution for 1Gb DRAM chips used in our study.

In our study we use 4 DIMMs for a total of 32 GB memory, while the board can have up to 8 DIMM slots for a total of 512 GB of DRAM memory. By populating all the slots, the average system energy savings provided by HRM can reach even higher as the ratio of memory to total power will be increased. Thus, the energy savings enabled by HRM scales up not only with the density of DRAM chips but also with the number and size of DRAM devices.

5.2.3 Availability Evaluation

Our QAGov scales T_{REFI} dynamically during the lifetime of VMs based on the measured QoS and DIMM temperature. Figure 9 demonstrates how the governor scales T_{REFI} for a VM executing the x264 workload when the DRAM temperature varies from 45°*C* up to 80°*C*. On the left-axis (orange) of this figure, we show the availability of the VM (QoS). We depict the minimum QoS requested by the user, QoS_{req}, using a solid horizontal red line. The right-axis (blue) shows T_{REFI} that is changed dynamically by QAGov, while the second right-axis (green) shows the DRAM temperature, θ, of the DIMMs. The light colored vertical lines indicate timestamps when the VM is checkpointed, while the dark lines indicate the moments when the VM detected an uncorrectable error and has recovered to a previous checkpoint. As expected, we see that QAGov relaxes T_{REFI} as long as QoS(VM) > QoS_{req}. Initially, QoS(VM) starts at 100% and then it decreases with time, since each checkpoint makes the system unavailable for δ seconds. It is seen that if the VM crashes and needs to be restarted, QoS(VM) may be lower than QoS_{req}; in this case the governor reduces T_{REFI} quickly until QoS(VM) > QoS_{req}. Interestingly, in Figure 9 we see that the VM has crashed at the 16^{th} hour, however QoS(VM) has only momentarily been lower than QoS_{req}. As a result, the relaxed T_{REFI} has not been changed significantly, which demonstrates that our governor works properly. Furthermore, we observe that QAGov has limited T_{REFI_MAX}, which does not exceed 768ms, in a period from 26^{th} to 33^{th} hour since the DRAM temperature has increased up to 80°*C*, demonstrating that QAGov is effective even at extreme DRAM temperatures.

5.2.4 Evaluation of HRM-FT

The use of HRM-FT enables us to operate VarMem under the maximum T_{REFI}, which is available on our server, and thus mitigate the 3% performance overhead introduced by the CR mechanism. Figures 10a and 10b show the number of correctable errors manifested when running different benchmarks with enabled HRM-FT (4 MCUs are interleaved using SoftInterlv) at 60°*C* and 80°*C*, respectively. Figure 10c presents the number of uncorrectable errors, which we observe only when the DRAM temperature rises to 80°*C*. We

see that the number of correctable and uncorrectable errors vary across benchmarks by up to 6× and 2.5×, which implies that program inherent features may significantly affect DRAM reliability. Furthermore, we also measure how many times out of the total number of experiments we observe incorrect executions for each workload. We consider the execution of a program as incorrect if the output of the program differs from the expected/golden output. The percentage of executions that are incorrect can be as low as 0.5% (`fluidanimate`) and can reach up to 18% (`x264`) at 80°C. Thus, HRM-FT allows the users to achieve higher energy savings than the HRM-CR policy, however the user should implement additional software methods to handle DRAM errors if they affect the reliability of a specific application.

Source code modifications. To use the HRM-FT policy, it is sufficient to introduce minimal changes to the application source code. For example, we added 28 lines of code to allocate the application data of `x264` in the VarMem domain, which is negligible in comparison with the size of the original code (21000 lines). We also discovered that the performance overhead introduced by the source code changes (`malloc` is substituted with `numa_alloc_onnode`) is insignificant.

6 Related Work

Heterogeneous Reliability Memory. The use of HRM [45, 46] and placement of data in each type of memory based on the required reliability [26, 44, 45, 55] has gained a lot of attention in recent years. HRM can be also used to enable energy efficient approximate computing [13, 50, 59, 68, 69]. However, those schemes have never been implemented and evaluated on a real system, and have not addressed any practical limitations, as we do in this study. Furthermore, fault-injection [45] tools were utilized for investigating their error resiliency, which may not correspond to a realistic DRAM error behavior.

Memory Interleaving. An analysis of different DRAM architectures [15] has shown the advantages of interleaving, especially in future DRAMs. To increase the memory bandwidth, one of the prior works has proposed a hardware programmable memory interleaver [25]. However, such an approach requires intrusive hardware modifications and system software support. Bank-level interleaving schemes [43, 73] have been proposed to enable partitioning of data across banks at the software level. Even though, they may reduce significantly the number of bank conflicts triggered by parallel accesses to DRAM, the proposed techniques interleave data only at the bank level.

Relaxed DRAM circuit parameters. There are several studies [4, 5, 14, 41, 42, 53, 70] that proposed to relax T_{REFI} and improve DRAM energy efficiency. Most of those schemes require an initial characterization of the cell retention time. However, the interference between neighboring cells [1, 52, 56] and the resultant *variable cell retention time* (VRT) [54, 57, 57] make these schemes impractical, since they also require intrusive hardware modifications. Other schemes try to exploit the implicit refresh of each access [23, 67] and error-resilient application properties [34], which are orthogonal to our work. Some studies have proposed to relax DRAM circuit parameters and use ECC to handle all manifesting errors [40, 51]. These studies imposed an assumption that all errors are corrected by ECC and did not propose mechanisms to handle uncorrectable errors. A few research studies [10, 17, 18, 36] have investigated the effect of V_{DD} scaling on DRAM reliability. It has been shown that V_{DD} does not have a significant impact on DRAM reliability if it is close to the nominal threshold.

The error characterization of DRAM operating under scaled T_{REFI} has been conducted using embedded processors [2] or FPGAs in many studies [33, 34, 41], however these studies did not consider the effect of a full system stack on DRAM reliability, that can be systematically investigated only on real systems.

Previous works have also tried to relax DRAM timings parameters and thus improve performance and energy efficiency of DRAM [9, 38, 61]. In our future work, we will extend our policies to control these parameters and investigate the energy gain that we can achieve on a real server.

7 Conclusion

In this paper, we present HaRMony, a heterogeneous-reliability memory framework, implemented on a real ARMv8-based server with a fully-fledged virtualization stack. We demonstrate that HaRMony effectively addresses challenges, such as hardware memory interleaving, that hampers implementing HRM. To enable reliable execution of virtualized and non-virtualized workloads, we integrate HaRMony with several mechanisms, such as the VM checkpointing/restarting mechanism, which automatically handle DRAM errors. We design the first of its kind QoS-aware governor that dynamically adjusts DRAM circuit parameters to minimize the system energy consumption, while meeting the requested QoS. Our results show that HaRMony enables 8.6% system energy savings on average for non-virtualized and 8.4% for virtualized workloads, ensuring the seamless server operation even under extreme temperatures.

Acknowledgments

The authors thank the anonymous reviewers and our shepherd Michael Swift for their insightful comments and suggestions. This work was funded by the H2020 Framework Program of the European Union through the UniServer Project (Grant Agreement 688540, http://www.uniserver2020.eu) and OpreComp project (Grant Agreement 732631, http://oprecomp.eu).

References

[1] Zaid Al-Ars, Said Hamdioui, and Ad J van de Goor. 2004. Effects of bit line coupling on the faulty behavior of DRAMs. In *VLSI Test Symposium, 2004. Proceedings. 22nd IEEE*. IEEE, 117–122.

[2] Seungjae Baek, Sangyeun Cho, and Rami Melhem. 2014. Refresh now and then. *IEEE Trans. Comput.* 63, 12 (2014), 3114–3126.

[3] Fabrice Bellard. 2005. QEMU, a fast and portable dynamic translator.. In *USENIX Annual Technical Conference, FREENIX Track*, Vol. 41. 46.

[4] Ishwar Bhati, Mu-Tien Chang, Zeshan Chishti, Shih-Lien Lu, and Bruce Jacob. 2016. DRAM refresh mechanisms, penalties, and trade-offs. *IEEE Trans. Comput.* 65, 1 (2016), 108–121.

[5] Ishwar Bhati, Zeshan Chishti, Shih-Lien Lu, and Bruce Jacob. 2015. Flexible auto-refresh: Enabling scalable and energy-efficient DRAM refresh reductions. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 235–246.

[6] Christian Bienia and Kai Li. 2011. *Benchmarking modern multiprocessors*. Princeton University Princeton.

[7] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems* 25, 6 (2009), 599–616.

[8] Carel. 2012. ir33 universale electronic control. www.carel.com/product/ir33-universale

[9] Kevin K Chang, Abhijith Kashyap, Hasan Hassan, Saugata Ghose, Kevin Hsieh, Donghyuk Lee, Tianshi Li, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. 2016. Understanding latency variation in modern DRAM chips: Experimental characterization, analysis, and optimization. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 44. ACM, 323–336.

[10] Kevin K Chang, A Giray Yağlıkçı, Saugata Ghose, Aditya Agrawal, Niladrish Chatterjee, Abhijith Kashyap, Donghyuk Lee, Mike O'Connor, Hasan Hassan, and Onur Mutlu. 2017. Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1, 1 (2017), 10.

[11] Jianhua Che, Congcong Shi, Yong Yu, and Weimin Lin. 2010. A synthetical performance evaluation of openvz, xen and kvm. In *2010 IEEE Asia-Pacific Services Computing Conference*. IEEE, 587–594.

[12] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*. Ieee, 44–54.

[13] Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*. ACM, 113.

[14] Zehan Cui, Sally A McKee, Zhongbin Zha, Yungang Bao, and Mingyu Chen. 2014. DTail: a flexible approach to DRAM refresh management. In *Proceedings of the 28th ACM international conference on Supercomputing*. ACM, 43–52.

[15] Vinodh Cuppu, Bruce Jacob, Brian Davis, and Trevor Mudge. 1999. A performance comparison of contemporary DRAM architectures. In *ACM SIGARCH Computer Architecture News*, Vol. 27. IEEE Computer Society, 222–233.

[16] John T Daly. 2006. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future generation computer systems* 22, 3 (2006), 303–312.

[17] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R Hanebutte, and Onur Mutlu. 2011. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM international conference on Autonomic computing*. ACM, 31–40.

[18] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F Wenisch, and Ricardo Bianchini. 2012. MultiScale: memory system DVFS with multiple memory controllers. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*. ACM, 297–302.

[19] Raspberry Pi Foundation. 2016. Raspberry Pi 3 Model B. https://www.raspberrypi.org/

[20] Shrikanth Ganapathy, Adam Teman, Robert Giterman, Andreas Burg, and Georgios Karakonstantis. 2015. Approximate computing with unreliable dynamic memories. In *IEEE 13th International New Circuits and Systems Conference, NEWCAS 2015, Grenoble, France, June 7-10, 2015*. IEEE, 1–4. https://doi.org/10.1109/NEWCAS.2015.7182027

[21] Rohan Garg, Komal Sodha, Zhengping Jin, and Gene Cooperman. 2013. Checkpoint-restart for a network of virtual machines. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 1–8.

[22] Mohsen Ghasempour, Aamer Jaleel, Jim D Garside, and Mikel Luján. 2016. Dream: Dynamic re-arrangement of address mapping to improve the performance of DRAMs. In *Proceedings of the Second International Symposium on Memory Systems*. ACM, 362–373.

[23] Mrinmoy Ghosh and Hsien-Hsin S Lee. 2007. Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D die-stacked DRAMs. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 134–145.

[24] Bharan Giridhar, Michael Cieslak, Deepankar Duggal, Ronald Dreslinski, Hsing Min Chen, Robert Patti, Betina Hold, Chaitali Chakrabarti, Trevor Mudge, and David Blaauw. 2013. Exploring DRAM organizations for energy-efficient and resilient exascale memories. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 23.

[25] Manil Dev Gomony, Benny Akesson, and Kees Goossens. 2013. Architecture and optimal configuration of a real-time multi-channel memory controller. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 1307–1312.

[26] Manish Gupta, Vilas Sridharan, David Roberts, Andreas Prodromou, Ashish Venkat, Dean Tullsen, and Rajesh Gupta. 2018. Reliability-aware data placement for heterogeneous memory architecture. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 583–595.

[27] Linley Gwennap. 2016. X-Gene 3 Challenges Xeon E5. (2016). https://www.linleygroup.com/uploads/x-gene-3-white-paper-final.pdf

[28] Linley Gwennap. 2017. Performance Arms X-Gene 3 for Cloud. (2017). https://www.linleygroup.com/uploads/x-gene-3-for-cloud.pdf

[29] Takeshi Hamamoto, Soichi Sugiura, and Shizuo Sawada. 1998. On the retention time distribution of dynamic random access memory (DRAM). *IEEE Transactions on Electron devices* 45, 6 (1998), 1300–1309.

[30] Paul H Hargrove and Jason C Duell. 2006. Berkeley lab checkpoint/restart (blcr) for linux clusters. In *Journal of Physics: Conference Series*, Vol. 46. IOP Publishing, 494.

[31] John L Henning. 2006. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News* 34, 4 (2006), 1–17.

[32] JEDEC Solid State Technology Association. 2010. JESD79-F: DDR3 SDRAM Specification. https://www.jedec.org/sites/default/files/docs/JESD79-3F.pdf

[33] Matthias Jung, Deepak M Mathew, Carl C Rheinländer, Christian Weis, and Norbert Wehn. 2017. A Platform to Analyze DDR3 DRAM's Power and Retention Time. *IEEE Design & Test* 34, 4 (2017), 52–59.

[34] Matthias Jung, Éder Zulian, Deepak M Mathew, Matthias Herrmann, Christian Brugger, Christian Weis, and Norbert Wehn. 2015. Omitting refresh: A case study for commodity and wide i/o drams. In *Proceedings of the 2015 International Symposium on Memory Systems*. ACM, 85–91.

[35] Christos Kalogirou, Panos K Koutsovasilis, Manolis Maroudas, Christos D Antonopoulos, Spyros Lalis, and Nikolaos Bellas. 2017. Edge and Cloud Provider Cost Minimization by Exploiting Extended Voltage and Frequency Margins.. In *PARCO*. 814–823.

[36] Georgios Karakonstantis, Debabrata Mohapatra, and Kaushik Roy. 2012. Logic and Memory Design Based on Unequal Error Protection for

Voltage-scalable, Robust and Adaptive DSP Systems. *Signal Processing Systems* 68, 3 (2012), 415–431. https://doi.org/10.1007/s11265-011-0631-9

[37] K. Kim and J. Lee. 2009. A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs. *IEEE Electron Device Letters* 30, 8 (Aug 2009), 846–848. https://doi.org/10.1109/LED.2009.2023248

[38] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu. 2015. Adaptive-latency DRAM: Optimizing DRAM timing for the common-case. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 489–501. https://doi.org/10.1109/HPCA.2015.7056057

[39] Gunther Lehmann and Manfred Menke. 2003. Partial refresh for synchronous dynamic random access memory (SDRAM) circuits. US Patent 6,665,224.

[40] Chung-Hsiang Lin, De-Yu Shen, Yi-Jung Chen, Chia-Lin Yang, and Cheng-Yuan Michael Wang. 2015. Secret: a selective error correction framework for refresh energy reduction in DRAMs. *ACM Transactions on Architecture and Code Optimization (TACO)* 12, 2 (2015), 19.

[41] Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu. 2013. An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 60–71.

[42] Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. 2012. RAIDR: Retention-aware intelligent DRAM refresh. In *ACM SIGARCH Computer Architecture News*, Vol. 40. IEEE Computer Society, 1–12.

[43] Lei Liu, Zehan Cui, Mingjie Xing, Yungang Bao, Mingyu Chen, and Chengyong Wu. 2012. A Software Memory Partition Approach for Eliminating Bank-level Interference in Multicore Systems. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT '12)*. ACM, New York, NY, USA, 367–376. https://doi.org/10.1145/2370816.2370869

[44] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G Zorn. 2012. Flikker: saving DRAM refresh-power through critical data partitioning. *ACM SIGPLAN Notices* 47, 4 (2012), 213–224.

[45] Yixin Luo, Sriram Govindan, Bikash Sharma, Mark Santaniello, Justin Meza, Aman Kansal, Jie Liu, Badriddine Khessib, Kushagra Vaid, and Onur Mutlu. 2014. Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 467–478.

[46] Mitesh R Meswani, Sergey Blagodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel H Loh. 2015. Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 126–136.

[47] Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu. 2015. Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field. In *Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '15)*. IEEE Computer Society, Washington, DC, USA, 415–426. https://doi.org/10.1109/DSN.2015.57

[48] Micron Technology. 1994. TN-04-30: Various Methods of Dram Refresh. https://downloads.reactivemicro.com/Electronics/DRAM/DRAM%20Refresh.pdf

[49] Micron Technology. 2015. MT18JSF1G72AZ-1G9 - 8GB. https://www.micron.com/products/dram-modules/udimm/part-catalog/mt18jsf1g72az-1g9

[50] Sparsh Mittal. 2016. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)* 48, 4 (2016), 62.

[51] Prashant J Nair, Dae-Hyun Kim, and Moinuddin K Qureshi. 2013. ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 72–83.

[52] Yoshinobu Nakagome, M Aoki, S Ikenaga, M Horiguchi, S Kimura, Y Kawamoto, and K Itoh. 1988. The impact of data-line interference noise on DRAM scaling. *IEEE Journal of Solid-state circuits* 23, 5 (1988), 1120–1127.

[53] Minesh Patel, Jeremie S Kim, and Onur Mutlu. 2017. The reach profiler (REAPER): Enabling the mitigation of DRAM retention failures via profiling at aggressive conditions. *ACM SIGARCH Computer Architecture News* 45, 2 (2017), 255–268.

[54] Moinuddin K Qureshi, Dae-Hyun Kim, Samira Khan, Prashant J Nair, and Onur Mutlu. 2015. AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*. IEEE, 427–437.

[55] Arnab Raha, Hrishikesh Jayakumar, Soubhagya Sutar, and Vijay Raghunathan. 2015. Quality-aware data allocation in approximate DRAM. In *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. IEEE Press, 89–98.

[56] Michael Redeker, Bruce F Cockburn, and Duncan G Elliott. 2002. An investigation into crosstalk noise in DRAM structures. In *Memory Technology, Design and Testing, 2002.(MTDT 2002). Proceedings of the 2002 IEEE International Workshop on*. IEEE, 123–129.

[57] Phillip J Restle, JW Park, and Brian F Lloyd. 1992. DRAM variable retention time. *International Technical Digest on Electron Devices Meeting* (1992), 807–810.

[58] Adrian Sampson, André Baixo, Benjamin Ransford, Thierry Moreau, Joshua Yip, Luis Ceze, and Mark Oskin. 2015. Accept: A programmer-guided compiler framework for practical approximate computing. *University of Washington Technical Report UW-CSE-15-01* 1, 2 (2015).

[59] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. 2011. EnerJ: Approximate data types for safe and general low-power computation. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 164–174.

[60] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. 2009. 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)*. ACM, New York, NY, USA, 193–204. https://doi.org/10.1145/1555349.1555372

[61] Wongyu Shin, Jungwhan Choi, Jaemin Jang, Jinwoong Suh, Youngsuk Moon, Yongkee Kwon, and Lee-Sup Kim. 2016. DRAM-Latency Optimization Inspired by Relationship between Row-Access Time and Refresh Timing. *IEEE Trans. Comput.* 65, 10 (2016), 3027–3040.

[62] Julian Shun and Guy E. Blelloch. 2013. Ligra: A Lightweight Graph Processing Framework for Shared Memory. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '13)*. ACM, New York, NY, USA, 135–146. https://doi.org/10.1145/2442516.2442530

[63] Gaurav Singh, Greg Favor, and Alfred Yeung. 2014. Appliedmicro x-gene2. In *Hot Chips 26 Symposium (HCS), 2014 IEEE*. IEEE, 1–24.

[64] V. Sridharan and D. Liberty. 2012. A study of DRAM failures in the field. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 1–11. https://doi.org/10.1109/SC.2012.13

[65] Micron Technology. [n. d.]. TN-41-01: Calculating Memory System Power For DDR3. https://www.micron.com/~/media/Documents/Products/Technical%20Note/DRAM/TN41_01DDR3_Power.pdf

[66] Adam Teman, Georgios Karakonstantis, Robert Giterman, Pascal Andreas Meinerzhagen, and Andreas Peter Burg. 2015. Energy versus data integrity trade-offs in embedded high-density logic compatible dynamic memories. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*, Wolfgang Nebel and David Atienza (Eds.). ACM, 489–494. http://dl.acm.org/citation.cfm?id=2755864

[67] Konstantinos Tovletoglou, Dimitrios S Nikolopoulos, and Georgios Karakonstantis. 2017. Relaxing DRAM refresh rate through access

pattern scheduling: A case study on stencil-based algorithms. In *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 45–50.

[68] Vassilis Vassiliadis, Jan Riehme, Jens Deussen, Konstantinos Parasyris, Christos D Antonopoulos, Nikolaos Bellas, Spyros Lalis, and Uwe Naumann. 2016. Towards automatic significance analysis for approximate computing. In *Code Generation and Optimization (CGO), 2016 IEEE/ACM International Symposium on*. IEEE, 182–193.

[69] Swagath Venkataramani, Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Quality programmable vector processors for approximate computing. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.

[70] Ravi K Venkatesan, Stephen Herr, and Eric Rotenberg. 2006. Retention-aware placement in DRAM (RAPID): Software methods for quasi-non-volatile DRAM. In *The Twelfth International Symposium on High-Performance Computer Architecture, 2006*. IEEE, 155–165.

[71] Doe Hyun Yoon and Mattan Erez. 2010. Virtualized and flexible ECC for main memory. In *ACM SIGARCH Computer Architecture News*, Vol. 38. ACM, 397–408.

[72] John W Young. 1974. A first order approximation to the optimum checkpoint interval. *Commun. ACM* 17, 9 (1974), 530–531.

[73] Heechul Yun, Renato Mancuso, Zheng-Pei Wu, and Rodolfo Pellizzoni. 2014. PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 155–166.

[74] Zhao Zhang, Zhichun Zhu, and Xiaodong Zhang. 2000. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *Proceedings of the 33rd annual ACM/IEEE International Symposium on Microarchitecture*. ACM, 32–41.