Memory Errors In Operating Systems – Problem and Solutions

Accompanying Paper for the Seminar Presentation

Robert de Temple
Friedrich Alexander Universität Erlangen-Nürnberg
robert.detemple@gmail.com

ABSTRACT

Reliability of is of growing concern in computational systems [6]. Studies have shown [7, 3, 5] that memory errors are a major source of hardware errors, comparable to CPU and HDD errors [5], and therefore a major source of system reliability problems. In this paper I present a short introduction to the causes and prevalence of main memory errors and showcase a selection of common approaches to counteract such memory errors. Furthermore I describe three recent approaches, namely Online Memory Assessment with Linux RAMpage, Dynamically Replicated Memory for PCM, as well as an analysis of the inherent error–resilience of the HPC–OS "Kitten" and Cray Linux Environment.

Keywords

memory, errors, DRM, ECC, RAMpage, PCM, DRAM

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

1. INTRODUCTION

Reliability of is of growing concern in computational systems [6]. Since the first studies on DRAM reliability, such as those published by F. A. Ziegler et al. in the 1970s [9], memory error rates have been thought to be very low in comparison to most other hardware components, and to be predominantly caused by transient radiation effects [3, 9]. But recent field studies conducted on hardware outside of lab environments state that this might not be the case [7, 3, 5]. They argue that memory errors are a major source of permanent and sporadic hardware errors, comparable in frequency to CPU and hard drive faults [5, 8]. These results underline the importance of memory errors in system reliability considerations.

The goal of this paper is to provide a broad overview of the current knowledge about the prevalence and causes of memory errors, as well as techniques for memory error mitigation. I provide a short introduction to the causes and prevalence of main memory errors, and discuss the recent shift in the error model. I then give an introduction to a selection of error correction and detection techniques in wide use today, and showcase three recent approaches for memory error mitigation. Two software solutions, namely on-line memory assessment with *Linux RAMpage* and the lightweight HPC-OS *Kitten* exemplifying the prospect of inherent error-resilience of the operating system and finally *Dynamically Replicated Memory* as a hardware solution for future phase-change memory devices.

2. RAM ERRORS

Understanding the causes, type and rate of memory errors is of utmost importance for the design or application of systems for error detection, correction or tolerance. The following subsections provide an introduction to DRAM technology, the theoretical background for its most common causes of soft errors as well as the results from several field studies on the rate and characteristics of DRAM errors. They close with an outlook on the error model of PCM (phase-change memory), as the dominance of DRAM might come to an end in the medium term, if PCM technology matures as predicted [4].

2.1 DRAM

The most prevailing technology used to build main memory in PCs, Servers and high performance computers alike is DRAM (dynamic random-access memory) [3]. Because of this reason most research and preventive measures concerning memory faults have been focused on this technology. The first DRAM chips became widely available in the early 1970s. The basic principle of DRAM has mostly been left unaltered since that time, except for the continuous advances in structure size, new integrated controllers as well as data buses. DRAM saves data through the individual charge of capacitors in an array latched through low-drain FE-transistors. Figure 1 showcases such a DRAM array.

Early on spontaneous, transient (soft) DRAM-errors emerged in computing practice. Older memory technologies, such as core memory, scarcely exhibited such errors, yet DRAM technology suffered under relatively high rates of these errors of unknown origin. Ziegler et al. [9] investigated the causes through systematic exposure of DRAM chips to active radiation sources. His paper published in 1979 as well as several follow-up studies in later years showed that the main cause of the observed soft errors lay in radiation. First and foremost alpha-particles emitted through radioactive decay in

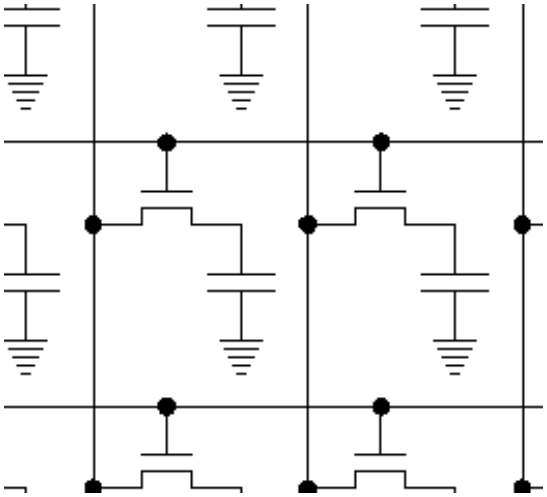

Figure 1: Array of capacitors in DRAM, individually controlled by MOSFETs. Access is multiplexed through bitlines and wordlines.

contaminated chip packaging, secondly through secondary neutron and proton radiation caused by cosmic rays. Both types of radiation alter the charge in individual capacitors and MOSFET gates thereby causing data corruption [9].
The per-chip rate of soft errors in DRAM decreased consistently with new fabrication nodes as well as improved packaging even though the amount of memory capacity per chip grew as quickly as feature sizes shrunk [8, 7]. Figure 2 demonstrates the decline in soft error rates with reduced feature sizes. The reason for this unintuitive behavior is the

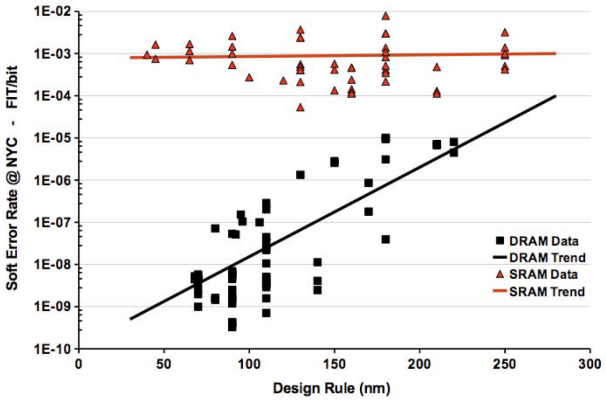

Figure 2: The evolution of DRAM soft errors compared to feature sizes and SRAM as a reference [8]. Error rates are given in FIT (Failures In Time = Failures per billion hours).

lack of scaling of cell capacity [8]. While the size of most DRAM features shrunk in all dimensions, which lead to a decreased critical charge for state alteration (Q_{crit}), capacity did not. The capacitors mostly scaled to become thinner and deeper, making them harder targets for radiation particles to hit, yet maintained a comparable capacity and therefore comparable Q_{crit} to older technology nodes, which easily offsets the common drawbacks in radiation resilience of smaller feature sizes. As shown in Figure 3 an additional, relative rise of multi-bit-errors due to the higher densities can be observed in lab tests [8].

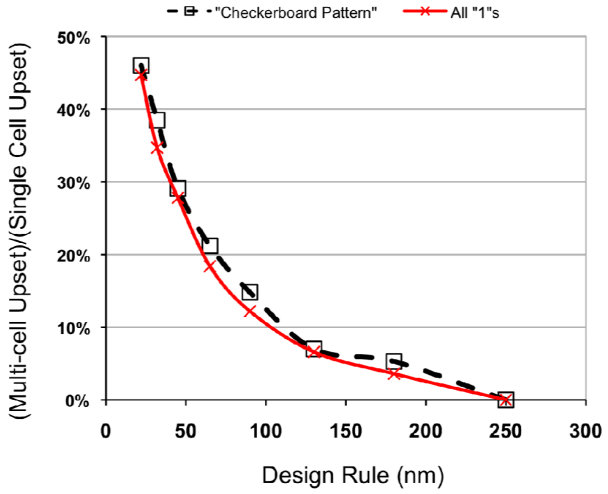

Figure 3: The evolution of DRAM multibit errors [8] as seen in lab testing. The relative probability of multiple cells upsets by single radiation particles rises with shrinking feature sizes.

As a result, the common understanding of DRAM-errors, up until recently, was almost exclusively based on Ziegler et al. [9, 8, 3] and can be summed up as follows:

1. Hard errors are uncommon in DRAM, soft errors are the most common type.
2. DRAM errors are caused by radiation and are therefore random – both in time and space
3. Error rates are low and dropping with new technology nodes, based on the accelerated measurements conducted by manufacturers.

2.2 Errors in HPC applications

Operating experience in the domain of high performance computing did not always reflect the predicted failure rates, as calculated with the data obtained with module tests [3]. This observation led to a series of field studies with the intention to investigate this matter. One known study, which examines the memory error rate in contemporary computing hardware, was published by Schroeder et al. in 2009 [7] with a strongly related follow-up study expanding on the results in 2012 by Hwang et al. [3]. These papers present a detailed analytical study of DRAM error characteristics obtained from a two-year long analysis of ECC error logs in high performance computers (HPC). The findings include:

1. An average 8 percent chance/year for each 1GiB Module of RAM to be affected by errors – easily several orders of magnitude higher than expected
2. A high rate of hard errors, between 60–80 percent, obscured by complex sporadic behavior – This questions radiation as the primary cause of memory errors in the field

3. A highly localized distribution of errors both in time and space. Especially a high prevalence of errors on kernel-occupied memory pages

These results question the then prevailing model of memory error causality in real-life applications. They show high error rates with high rates of hard errors, a result completely unexpected with the model established with Ziegler et al. Recent lab studies found error rates in the range of < 1 FIT[1] per Mbit [7, 3]. In comparison the results by Schroeder et al. translate to error rates of 25,000 - 75,000 FIT per Mbit [7]. The error rates observed in these field studies obviously don't reflect the rates observed in accelerated chip tests regularly conducted by DRAM chip manufacturers.

2.3 PC memory errors

In 2011 Nigthingale et al. [5] published a paper, which extends the results gathered by Schroeder et al. with an extensive analysis of hard and soft errors in common PC hardware. Areas of investigation include CPU, HDD as well as main memory error rates. The paper is based on data from the Windows Costumer Experience Improvement Program (CEIP) spanning an observation period of 8 months with a total of around 950.000 observed consumer PCs of mixed brands and quality. The user base consisted of a subset of Windows users that approved regular and automatic submission of data on errors or their machines, it therefore provides an unbiased sample set.
The general lack of ECC protection in PCs, the fact that the reporting process is strongly dependent on system crashes for error detection, as well as the need to filter out data corruption caused by software bugs, put hard constraints on the scope of identifiable DRAM errors in this study. It limited their type to 1-bit errors located in the approximately 30MiB kernel memory space (about 1.5 percent of the average memory size) with a further restriction to the 3MiB kernel core memory for the analysis of error location. These limitations prevented a reliable calculation of the absolute prevalence of DRAM errors and therefore lead to very conservative estimates as many potential errors recorded in the data could not be considered. The results are thus only qualitatively comparable to the error rates observed by Schroeder et al. in professional servers. The results include:

1. A high rate of fatal (OS crash) DRAM errors overall, with a mean chance of first-time failures of around 0.088 percent per year at average use. A rate within the range of the observed CPU and hard drive failures.
2. A high probability of error reoccurrence after the first incident. With an estimated 15–19 percent of reoccurring failures.
3. A high locality, with 79 percent of all secondary failures affecting the same bit.
4. A strong cross-correlation with CPU errors.

These results are roughly comparable to Schroeder et al. and Hwang et al. [5] despite their methodical differences, and show that PCs are significantly affected by memory errors as well. The high locality and the relatively high rate of reoccurring errors again point to sources outside of radiation as the main causes in practical application. The strong cross-correlation with CPU errors is discussed as a possible indicator for common causes such as EMC, heat, dirt or supply voltage fluctuations [5].

[1]Failures in Time = number of failures in 1 billion hours

2.4 Phase Change Memory

DRAM is a technology in active use for more than 40 years. Continuous scaling increased densities by several orders of magnitude, now reaching densities where individual atoms and electrons begin to have significant influence on its operation. For this reason further advancements in DRAM scaling are likely to slow down substantially [4]. The semiconductor industry spent considerable resources to find a replacement for DRAM that could scale well beyond current feature sizes [4]. Resistive memory is a promising replacement technology. PCM (phase-change memory), a particular type of resistive memory, is close to technical maturity[2][4].

PCM saves data in the structural properties of its cells. Each cell consist of a switching element, such as a MOSFET, and a memory element, usually made of chalcogenide glass. This material exhibits the unique characteristic of low electrical resistance when in its crystalline state, and high electrical resistance in its amorphous state. The write process is performed by heating the chalcogenide glass to 1000 °Kelvin with an attached electric heating element and quickly cooling it off again (quenching) [4]. The rate of the quenching process determines the structure of the material and thus the stored value. High cooling rates freeze the material in an amorphous state, which results in high electrical resistance, while lower cooling rates provide enough time for (partial) crystallization, which leads to reduced cell resistance. The corresponding read process involves a straight forward measurement of the cell's resistance by running a low current through the material and measuring the drop in voltage [4].
PCM has some distinct advantages next to its good scalability. It is non-volatile, which makes it viable as a contender for flash memory replacement. It uses little energy, exhibits low latency and high read speeds, partly because there is no need for memory refresh cycles [4]. It also has a high resistance to soft errors caused by radiation, due to the high particle energies needed to change the state of a cell.
One major disadvantage is a strong write limitation of around 100 million [4] write cycles for each bit with recent prototypes[3]. This number is expected to increase with feature size reduction but is still expected to stay low enough to pose a challenge in future RAM applications [4]. The main cause for the write limitation is mechanical stress inside a cell while heated. It leads to a detachment of the heating element and renders the cell unwritable. The last bit written to a cell before failure is therefore "stuck" but is still accessible for readout.

3. COMMON APPROACHES

[2]Semiconductor manufacturer Micron Technologies tentatively offered 1st generation PCM memory between 2010 and 2014 as a lower density, high speed flash alternative for niche applications.
[3]at the 65nm node

The error rates of memory hardware are too high for operation without additional measures if data reliability is of concern. For this reason several error resilience techniques have been developed that are widely used to mitigate the impact of memory faults. The following subsections describe two of the most common techniques: ECC-memory and off-line memory testing.

3.1 ECC-memory

ECC: Error-Correcting Code, describes the use of information redundancy for error detection as well as error correction. The type and scope of this information redundancy can vary wildly with the type of application and the desired error tolerance. ECC for memory error treatment is usually realized through specialized ECC-DRAM hardware. The relative ease of deployment, transparency for the OS and applications, and manageable overhead lead to wide adoption of ECC memory in HPC and server applications while adoption in low cost servers or PCs is still low due to prohibitive hardware costs and differing reliability requirements.

The most simple form of ECC coding is the addition of single parity bits to the protected data. The value of the parity bit is chosen in such a way that the resulting data (payload + parity bit) has an even number of set bits. Flipping a single bit always results in an uneven number of set bits, which points to data corruption, yet the location of the bit flip cannot be resolved. Flipping two or more bits might result in an even number of set bits. This makes half of these higher order faults undetectable for this scheme. Therefore parity bits provide single error detection and no error correction [4].
The most common form of ECC in hardware is SECDED ECC (Single Error Correction, Double Error Detection) usually on 64-bit words realized with Hamming codes [4, 6, 7]. This coding scheme adds 8 additional parity bits saved along with each 64-bit block, causing a data overhead of 12.5 percent. SECDED ECC can reliably detect two independent bit faults and correct a single bit fault in each protected block including faults in the parity bits themselves.
These simple ECC schemes have certain limitations when multi-bit errors occur. Such errors might be caused by radiation (see section 2.1), the accumulation of hard faults (see section 4.3) or the failure of a whole memory chip on a module. This lead to the development of more advanced ECC techniques to counteract these threats. A common solution to prevent hard fault accumulation is page retirement. Here memory pages that have repeatedly been found to contain errors, are marked as bad and removed from the system's memory pool. Mulit-bit errors on a single chip, up to complete chip failures, can be tolerated by advanced ECC techniques such as *Chipkill* or *Advanced ECC* developed by IBM and Sun Microsystems respectively. These solutions apply more complex coding algorithms and utilize additional ECC-data distributed on all memory chips on the same memory module, in principle broadly comparable to RAID technology for hard drives [1, 3, 4]. These higher order fault detection and correction schemes are still relatively uncommon as they require substantially more coding bits and pose much higher computational overhead.

3.2 Memory Testing

Memory testing is a well-established and well-explored approach to memory error mitigation [6]. Its goal is to test the hardware for static memory faults so further action can be taken if the memory is found to be defective. The basic principle is comparatively simple: A testing software writes data into the tested memory locations and then reads the data in those locations for comparison. Any alteration to the written data indicates a permanent fault in the memory hardware. This basic principle works well with simple faults such as 1-Bit stuck-at defects. Many static faults might exhibit more complex, sporadic behavior, including access order, timing and bit pattern correlations, making applicable testing procedures more elaborate. Test algorithms capable of reliable fault discovery have been the focus of research since the 1970s with many papers on the subject published such as by Hayes et al. (1975) or Nair et al. (1978). Modern algorithms developed in these and other research efforts are now able to discover a wide variety of complex, sporadic fault manifestations.
A major disadvantage of memory testing is the need for destructive write access to the tested memory range by most memory testing algorithms. It makes memory testing inapplicable for continuous background operation as memory pages allocated to other applications or the OS itself are inaccessible for testing [6]. This constraint limits most memory testing applications to off line use in system down time[4] and therefore severely restricts feasible testing rates. One notable exception in wide use is ECC-scrubbing, an extension to normal ECC operation. Here the memory controller continuously checks all ECC codes in the whole address range when idle, to reduce the time to error discovery in comparison to simple on-access checking. This method, just as standard ECC, is most useful for soft error detection due to the strictly passive testing algorithm.

4. NEW APPROACHES

The growing demand in reliable computing, the rising DRAM chip counts in high performance computers as well as new insights on memory soft error rates and characteristics fuel ongoing research on new methods of memory error treatment. This section provides an introduction to three error detection, prevention and tolerance techniques developed in recent research efforts. RAMpage in subsection 4.1 provides an approach for error detection, mitigating the drawbacks of memory testing. Subsection 4.2 discusses the possibility of error prevention with a special purpose operating system. Subsection 4.3 concludes with the introduction of dynamically replicated memory as a technology to tolerate accumulated hardware faults in phase-change memory.

4.1 RAMpage

Classic off-line memory testing has a major methodical drawback: Due to its need for destructive write access, a system being tested needs to be shut down from normal operation, which makes it unavailable for a significant period of time. This may pose a severe conflict of interest in the administration of many low-cost server applications or consumer PCs and ultimately lead to infrequent or irregular memory tests to preserve the availability of the systems. Yet a high frequency is of paramount importance for reliable system operation, especially if memory testing is to be employed

[4]Section 4.1 describes a possible solution to this problem

instead of ECC hardware. In 2013 Schirmeier et al. proposed a method to alleviate this problem. They designed a Linux memory testing system, RAMpage, which can continuously check memory in the background of normal system operation.
RAMpage tries to fulfill two goals: Continuous memory testing with little impact on the overall system performance and the implementation of an on-the-fly page retirement system. This allows for fully autonomous graceful degradation management implemented entirely in software, a feature currently only available with hardware ECC solutions [6].

4.1.1 Structure

The main challenges for continuous memory testing are the peculiarities of memory allocation in a modern operating system. Linux, like many modern operating systems, uses virtual memory as a means to control physical memory use in fine granularity. This poses a problem when page request are issued, as requested and thereupon allocated virtual memory pages are not guaranteed to cover the desired address space in physical memory. The Linux memory system doesn't even guarantee that the allocated virtual memory is actually backed by physical memory [6]. The address space of the physically allocated page frames is therefore unknown to the requesting user space application.
Concurrent use of the hardware by the OS and other applications pose a second problem to on-line memory testing. To test the whole physical memory space, RAMpage needs access to all page frames, including those already in use by other applications or even the kernel itself. This demands an applicable solution to somehow safely claim the necessary page frames, move their contents to other locations and finally release them after testing for reuse. RAMpage solves these problems with a 2-tier approach as shown in figure 4.

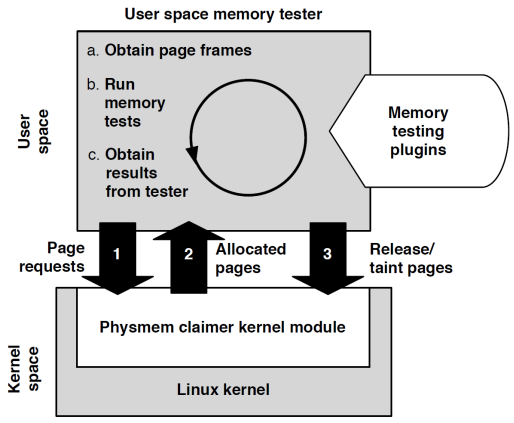

Figure 4: RAMpage overall structure [6]. The User Space Memory Tester manages all testing. It is supported by the Physmem Claimer kernel module that executes the page requests.

The user space memory tester is the main application. It administers the test schedule, starts execution and issues necessary page requests. It also provides page tainting/release functionality. It implements ports of all eight test procedures available in Memtest86+ with a focus on low page allocation requirements, to keep impact as low as possible on the host system. Each tested page frame needs to be allocated and pruned of data, a functionality a user space application cannot provide. A second kernel space module called the *Physmem* claimer kernel module is therefore utilized for page frame allocation [6].

4.1.2 Page Frame Allocation

Physmem uses three separate techniques page frame allocation. The first approach is allocation by means of the standard Linux page allocator (*buddy* allocator). The success rate of the *buddy* allocator varies strongly with the amount of free memory and the purpose the used memory is allocated for [6]. In the likely event of an allocation failure *Physmem* uses two fall-back methods to try to liberate the page frames before allocating them again with the *buddy* allocator.
The first fall-back method is provided by *hwpoison*. This is a Linux framework intended for memory error recovery. It adds a function to "poison" memory page frames, which precludes them from further allocation. Secondly it provides a memory "shaking" function. This function liberates single page frames in a non-destructive manner as it migrates their contents to other locations. The second fall-back method is derived from Linux's memory hotplugging system, intended for runtime removal of whole RAM-modules. It too provides a function for non-destructive memory liberation, albeit in coarse 4MiB chunks on a machine with 4kiB page frames [6]. If the requested page frames were successfully claimed, they are marked as non-cacheable, to prevent testing of the CPU-cache instead of the main memory and their locations are passed to the user-space tester which then maps them to its virtual address space and starts the testing procedures. If page frames have been found to be faulty in testing, the memory tester marks and therefore removes them from the system's memory pool with the poisoning functionality of the *hwpoison* subsystem [6].

4.1.3 Evaluation

The evaluation, both on real hardware with and without faulty memory and simulation with a series of automated tests, utilizing the Fail* framework, show RAMpage to be effective [6]. Only small parts of the memory space, around 4.8 percent are unclaimable and therefore unavailable for testing. These are primarily the page frames that contain the MMU page tables allocated to the kernel. Overall the simulation of 1-Bit stuck-at defects in Fail* ,using 262,016 experiments in a 2,047 MiB address space, resulted in a 94.7 percent detection rate. The missing 5.3 percent are accountable to unclaimable page frames and faults resulting in immediate system or RAMpage crashes. Figure 5 provides an overview of the distribution of undetected memory faults. Furthermore the system performance impact as well as energy usage was tested on server hardware and found to be low. Figure 6 shows the small impact RAMpage has on system performance.

The evaluated implementation of RAMpage proves to mostly fulfill its goals. RAMpage provides the whole Memtest86+ memory test set for continuous background operation in Linux. The low impact on system performance and the high test coverage make it an applicable solution when certain minimum goals of memory reliability and graceful degrada-

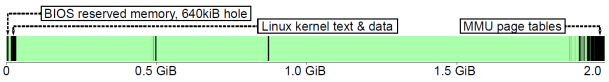

Figure 5: Undetected errors by RAMpage , mainly caused by unsuccessful page claiming [6]. Simulated single-bit stuck-at faults injected with Fail*. Around 5.3 percent of the faults could not be located.

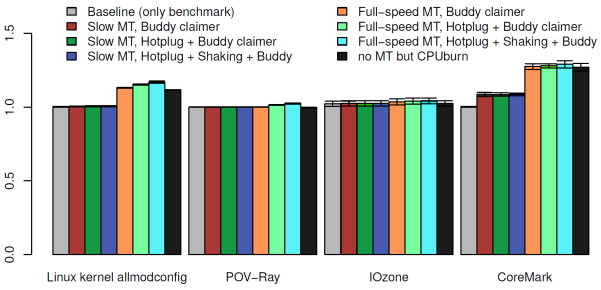

Figure 6: System performance degradation with use of RAMpage [6]. All results normalized to execution time without RAMpage. Slow MT equals a full memory test within 24h, Full-speed MT completed a full test within 30 minutes.

tion management are desired while ECC hardware is unavailable.

4.2 Error-Resistant Operating Systems

Several methods to achieve varying levels of memory error resistance that target the memory itself have been discussed thus far. Yet even with the combined application of all these methods a completely error-free system operation is unlikely to achieve [2]. Furthermore the results of Schroeder et al. [7, 3] and Nightingale et al. [5] point to possible sources of defect outside of the memory itself, possibly weakening the effectiveness of error many correction measures. The question therefore to be answered is, if the most important as well as most vulnerable software on a computer, the operating system itself [7, 5], can be designed in a way that contributes to error resistance. This section gives an introduction to research on error resistant operating systems, especially the results of Ferreira et al. (2012) [1] , which cover the HPC operating systems "Kitten" and Cray Linux Environment.

4.2.1 Introduction

In 2012 Ferreira et al. [1] published a study on operating system vulnerability. Its goal is to outline the structure and subsystems of the Cray Linux Environment (CLE) and Kitten, two special-purpose operating systems for HTC applications, to investigate their vulnerability to common memory failures and to evaluate optional error hardening approaches on both systems. A practical implementation of such error hardening techniques was not part of the study.
The Cray Linux Environment is a scalable, Linux-based operating system, developed for the Cray's XT line of supercomputers [**?**]. It includes a number of extensions and optimizations to adapt Linux for HTPC applications such as modified memory management and reduced jitter.
Kitten, developed by Sandia National Laboratories, is too a special-purpose operating system for supercomputers. It is also based on Linux, but with a highly modified and reduced code base. The goal of Kitten is to provide an operating system with the least possible amount of latency and jitter. The chosen approach is a reduction of OS kernel functionality to the bare minimum needed in HPC applications, while moving all other functionality into user-space. Kitten also supports the extension of its functionality through on-demand loading of guest operating systems.

4.2.2 Complexity

Ferreira et al. estimate the relative complexity of both operating systems through comparison of the relative the sizes of their codebases. This metric is used as a rough measure to estimate the difficulty of managing and implementing error mitigation techniques. CLE (Linux) and Kitten differ strongly in this regard. Linux has a current codebase of 10 million source lines of code (SLOC) with close to 350K SLOC in the x86 core kernel. In contrast Kitten's codebase consists of only 246K SLOC with just 30K SLOC in the core kernel. This is more than an order of magnitude less than Linux and points to a much simpler overall system structure.

4.2.3 Memory Footprint

A major influence on memory error resistance of any software is the amount of memory it uses. The less memory a software uses, the less often it is affected by memory errors as they are less likely to appear in data within its memory space [5]. That is to say it provides a smaller target. The most critical part of an operating system is its kernel [4, 5, 3]. A small kernel memory footprint of an operating system therefore supports error resistance.

A comparison of memory layouts in Figure 7 shows the primary difference between both operating systems. Linux uses a unified page pool with dynamic assignment of pages to all roles while Kitten partitions memory into two distinct regions. One region exclusively for kernel memory, with a fixed upper limit of 64MiB, and a second region used for user-space applications, which covers the rest of the memory space. The Linux kernel in CLE can theoretically grow to cover all of the available memory, but in practice the average memory requirements running a typical HPC application have been observed to be around 900MiB [4]. This is still more than $14x$ the worst-case requirements of Kitten.

4.2.4 Subsystems

An examination of all the major kernel subsystems allows a discussion of practical approaches to kernel hardening. Ipek et al. discuss the following subsystems:

1. Page table memory – Corruption considered fatal in both OS. Kitten employs deterministic mapping between physical and virtual address space. This allows for simple reconstruction in case of corruption as a hardening measure. Hardening CLE's page table involves complex redundancy.

2. Physical memory management – Corruption considered fatal in both OS. Hardening with software ECC or redundancy.

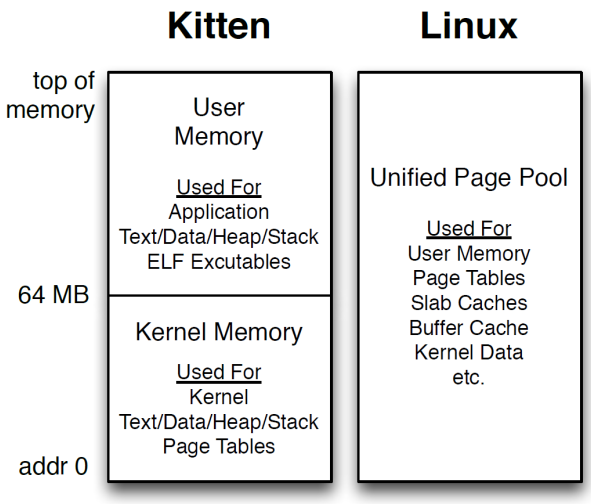

Figure 7: Comparison of the physical memory layout in Kitten and Linux [1]. Kitten partitions the memory with a fixed upper limit on kernel memory space of 64MiB.

3. Dynamic kernel memory - Hardening by protection of all allocation headers, possibly with ECC-like codes or redundant copies.
4. Address space and tasks – More complex in CLE. No concrete proposal for hardening.
5. Kernel entry and exit – Identical in both OS. Proposed implementation of kernel space checksumming on each kernel exit and entry. Much lower overhead of this approach due to smaller and contiguous kernel memory in Kitten.

These subsystems cover the vast majority of Kitten's kernel memory footprint and are also considered to be the minimal baseline functionality needed for HPC system operation [4]. All of these subsystems are considerably bigger and more complex in CLE as Linux provides more functionality in its kernel. CLE also includes a higher number of kernel subsystems than Kitten, all of wich are potentially susceptible to memory faults. This requires more elaborate hardening techniques if memory error resistance comparable to Kitten is to be achieved [4].

4.2.5 Evaluation

The analysis of the special-purpose operating systems Kitten and CLE / Linux shows that the Kitten lightweight operating system is most likely less susceptible to memory errors and an easier target to harden against such errors. It has a much smaller memory footprint, a simpler system structure and a highly deterministic system state, therefore providing a much smaller target area for memory faults. Nevertheless, hardening critical kernel functions, such as the memory page table, the memory management as well as the kernel entry and exit interfaces, to achieve increased error robustness, is possible in both operating systems.
The results of this theoretical investigation of OS memory fault resistance is supported by a recent study published by Hoffman et al. [2], who provide a comprehensive comparison of silent data corruption rates in hardened and unhardened special-purpose, real-time operating systems. Hoffman et al. show that a lean OS design and the implementation of hardening techniques indeed have a major effect on memory error resilience. In conclusion the design choices made in Kitten, driven by the goal of low overhead, latency and jitter, also support the implementation of effective fault resistance.

4.3 Dynamically Replicated Memory

PCM memory's primary cause of memory faults is caused by the write-limitation of its cells. It results in a steady accumulation of broken memory cells over the lifetime of the device. This fault characteristic makes PCM quickly unusable without the employment of additional techniques to mitigate the endurance problem. Many common solutions, such as those applied in DRAM or flash memory, are unfit for use in PCM, which poses a challenge to its operation.
This section presents *Dynamically Replicated Memory* (DRM), proposed by Ipek et al. in 2010 [4], as a method specially designed for the particular fault model and performance characteristics of PCM. It allows for graceful degradation of PCM systems and therefore extended system operation[5].

4.3.1 Approach

Possible countermeasures to cope with limited write-endurance fall into two categories: A) Mechanisms that prevent or delay hard memory faults. B) Techniques that allow for continued operation even when parts of the memory are already affected by faults ("graceful degradation").
Techniques of both categories have been the focus of numerous studies, usually as a part of research efforts targeting flash-memory – a technology with write-limitations similar to those in PCM. For this reason many applicable solutions that fall into category A already exist for PCM today. The most obvious countermeasure is the use of wear-leveling, just as with flash-memory, to spread write access evenly on all cells. This can be accomplished with several established wear-leveling algorithms. Secondly there's the possibility to use caching or hybrid hardware to reduce the amount of write cycles [4].
Yet even when applying the all possible techniques to delay cell wear, faults might manifest themselves early in the device's lifetime. The problem is the strong variation in the individual write-endurance of PCM cells, a characteristic expected to get worse with future technology nodes [4]. Depending on the amount of variation the time-to-failure between two neighboring cells in future PCM might differ by as much as an order of magnitude [4]. So if a useful lifetime it to be gotten out of a PCM device, a error mitigation technique must be employed that is able to cope with the fact that many individual memory cells might fail long before the expected average lifetime of the device has been reached. Flash technology has a similar problem. The solution applied for flash is advanced ECC technology, usually based on Reed-Solomon codes, that is capable to correct several tens of failures in large 16KiB blocks of data [4]. This approach creates substantial overhead, as the contents of large blocks of data need to be calculated and rewritten with every write access. In flash memory, where write access is executed in a block oriented manner anyway and at much

[5] albeit at reduced performance and capacity

lower speeds, this poses no problem. But for PCM connected at memory bus speeds, such a solution would not be viable. SECDED ECC with page retirement is a lightweight solution with low overhead in wide use with DRAM. With this method memory pages frames that have been found to contain a faulty bit are decomissioned ("retired") by the ECC controller. This approach is coarse and therefore wasteful as memory page sizes are commonly in the range of 4KiB and more, depending on the architecture. PCM is likely to quickly develop faults in most of the memory pages, which leads to rapid decommissioning of substantial parts of the memory, even though only one in over 4 thousand bits in each of the retired pages is actually faulty [4].

Ipek et al. propose a new approach that is capable to mitigate these problems: *Dynamically Replicated Memory*. The idea is to reuse memory pages that contain a small number of hard faults[6], and to pair them with other faulty but compatible memory pages. The goals of this method are an extension of useful memory lifetime through graceful degradation, better than obtainable with SECDED ECC or similar methods, low overhead and transparency for the operating system [4].

4.3.2 Components

DRM consists of several components that work in unison. The most important part is the PCM memory with the memory controller. The controller manages all access to the physical memory. It checks the memory for write errors, manages read access to replicated pages and provides a virtualization layer for the operating system [4].
The virtualization layer is realized through the use of a so called "real table". This internal page table, located in the memory, maps each address of the physical address space, as seen by the operating system, to a single or a redundant pair of hardware memory pages (called "real address space"). This method achieves transparency of the replication for the operating system. The OS is therefore able to issue read and write requests without any knowledge of the dynamic replication managed by the PCM controller in the background. Figure 8 shows the relationships between real table, page tables and memory pages [4].

The second component is a high priority kernel thread in the operating system. Its task is to run a greedy, low-overhead, approximate, dynamic pairing algorithm that pairs compatible pages together, so they can be reused by the memory controller. The kernel keeps an internal frame pool for this purpose[7], so there's always a high chance to quickly find two compatible memory pages when needed. Within a compatible pair of pages, any defective data block (= 1 Byte) in one page has a corresponding faultless data block in the other page. Figure 9 presents an exemplifies page compatibility. The communication between the controller and the OS is realized through timed interrupts and a 128-entry ring buffer kept in memory [4].

4.3.3 Write Access and Fault Detection

The memory controller issues a checker read after each write to the memory to test if the write succeeded. This read poses a comparatively small overhead as read access in PCM is very fast compared to the expensive write procedure [4]. If the write is found to be unsuccessful, the memory controller marks that part of the page frame as defective. DRM implements parity bits for each Byte of memory for this purpose. This provides fault marking with fine granularity and a storage overhead of only 12.5 percent – the same overhead as with SECDED ECC. When the location has been marked as defective, the controller tries to repeat the write on the backup page. If this write also fails, possibly because this was the first fault on the page and no backup page exists yet, the controller proceeds to fault recovery [4].

Fault recovery initiates the dynamic replication of memory pages. First the controller removes a new pair of replicated memory pages from the ring buffer, by mapping their locations in the real table and deleting their location in the buffer. The content of the newly defective memory page(s) is then copied to the new location, including the data that

[6]The paper proposes up to 160
[7]e.g. a pool of 10.000 memory pages

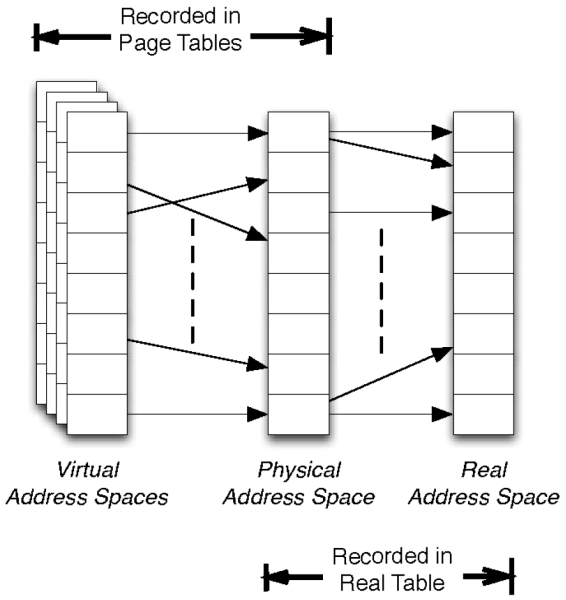

Figure 8: DRM Physical Address Space[4]. DRM adds another layer of virtualization between the virtual address space of the OS and the hardware called physical address space.

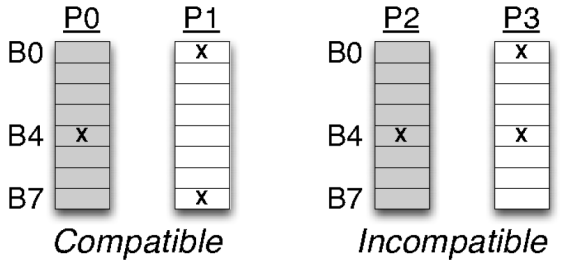

Figure 9: Example for compatible and incompatible page frames[4]. Compatible page frames complement each other.

failed to be written. Now the controller writes the address(es) of the defective memory page(s) into the ring buffer and wakes the OS kernel thread with an interrupt. The OS kernel thread moves all faulty pages from the buffer into its internal pairing pool, runs the pairing algorithm and refills the buffer with new page pairs [4].

4.3.4 Read Access

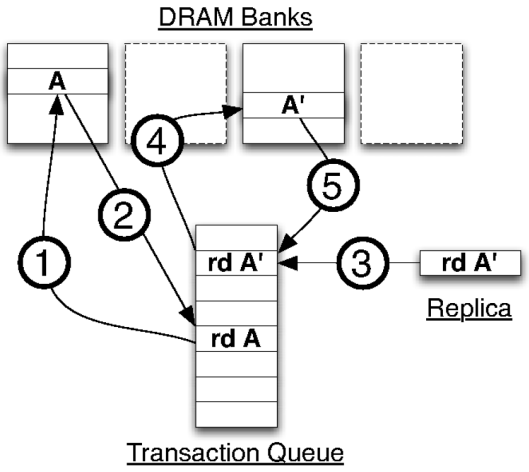

Figure 10: Lazy memory access scheduling in DRM [4].

Read access in DRM can happen with two distinct scheduling methods – "eager" and "lazy". "Eager" provides lower latency through parallel read requests to both replicas, at the cost of more overhead and higher energy usage. "Lazy" scheduling is the standard approach with lower overhead but higher latency in the case of a failed read attempt. Figure 10 shows the read procedure with "lazy" access scheduling. A typical read request in DRM with this scheduling consists of these distinct steps [4]:

1. Scheduling and execution of a read request to location A (primary copy – usually the page with fewer defective blocks).
2. Data returns – The controller checks the parity to resolve if the block in location A has been marked as defective.
3. A fault has been detected – The controller issues a read request to the replica.
4. The read request executes.
5. The read request completes – The controller returns the data.

4.3.5 Evaluation

DRM has been evaluated using a modified version of the SESC simulator. The simulated hardware includes a x86 CPU and a PCM memory subsystem modeled after DDR2-800 SDRAM. The simulation assumes perfect wear leveling, and a PCM fault model with hard bit failures that are uncorrelated in their local distribution. Defective cells are thought

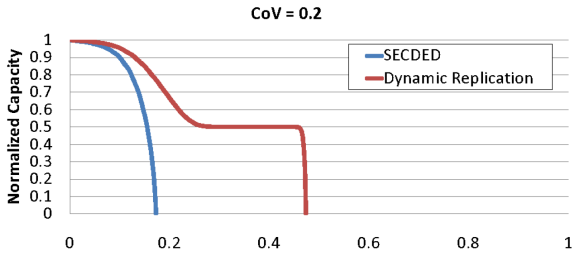

Figure 11: Endurance of SECDED ECC with page retirement compared to dynamic replication with medium variation of individual cell write-limitations [4]. Results normalized to ideal PCM lifetime with no variation.

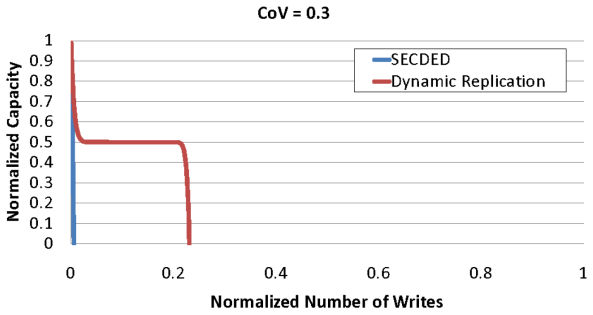

Figure 12: Endurance of SECDED ECC with page retirement compared to dynamic replication with high variation of individual cell write-limitations [4]. Results normalized to ideal PCM lifetime with no variation.

to stay readable and the individual cell lifetimes are varied with a normal distribution. This fault model is similar to the observed fault distribution and behavior of PCM prototypes [4]. The simulation includes three values for the coefficient of variation (CoV) of the lifetime distribution. Low (0.1), medium (0.3) and high (0.3) to model three possible levels of cell lifetime variation. Figure 11 shows the degradation of memory capacity over time using SECDED ECC with page retirement, compared to dynamic replication at a CoV of 0.2. Figure 12 shows the same comparison with a CoV of 0.3. The evaluation shows that DRM prolongs the endurance of PCM memory successfully, as this method is able to reuse page frames with up to 160 defective blocks, whereas ECC page retirement decommissions a page already after the first correctable fault.
The benefits of DRM compared to ECC, defining half of the initial capacity as the critical value for usable lifetime, range between a *1.25x* improvement at the low CoV of 0.1 and up to a *40x* improvement at a Cov of 0.3. The benefits are more pronounced with a higher variation of cell lifetimes such as those expected with shrinking feature sizes [4]. Ipek et al. further argue that DRM or similar techniques, complimented by wear-leveling, will become essential for the successful operation of PCM as main memory.

5. CONCLUSIONS

In this paper I present an introduction to the fault model of memory errors and their prevalence in practice. The results from big field studies conducted by Schroeder et al. and Nightingale et al. show that DRAM errors are several orders of magnitude more common than expected from data received in accelerated lab testing [7, 5]. The studies also show a very high percentage of hard faults, a result not expected with fault models that see radiation as the primary cause of memory errors. I further argue that new memory technologies, such as PCM, introduce additional problems with hard faults caused by cell write-limitation, which demand new approaches for error mitigation [4].

I also introduce several recent approaches for memory fault mitigation. The first two techniques are software solutions: RAMpage, an efficient memory error testing and page frame retirement tool, capable of continuous background memory testing in Linux – And Kitten, an exemplary special-purpose operating system with very reduced memory requirements, mostly static and simple memory allocation structures and simple code paths, which is evidently very suited for future error hardening techniques. And finally a hardware solution: Dynamically Replicated Memory, a method to provide low-overhead, finely-grained, graceful memory degradation in future PCM systems – several times more powerful than current SECDED-ECC page retirement techniques.

6. REFERENCES

[1] Kurt B. Ferreira, Kevin Pedretti, Ron Brightwell, Patrick G. Bridges, David Fiala, and Frank Mueller. Evaluating operating system vulnerability to memory errors. In *Proceedings of the 2Nd International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '12, pages 11:1–11:8, 2012.

[2] Martin Hoffmann, Christoph Borchert, Christian Dietrich, Horst Schirmeier, Rüdiger Kapitza, Olaf Spinczyk, and Daniel Lohmann. Effectiveness of Fault Detection Mechanisms in Static and Dynamic Operating System Designs. In IEEE Computer Society, editor, *Proceedings of the 17th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC '14)*, 2014.

[3] Andy A. Hwang, Ioan Stefanovici, and Bianca Schroeder. Cosmic rays dont strike twice: Understanding the nature. In *of DRAM Errors and the Implications for System Design, ASPLOS*, 2012.

[4] Engin Ipek, Jeremy Condit, Edmund B. Nightingale, Doug Burger, and Thomas Moscibroda. Dynamically replicated memory: Building reliable systems from nanoscale resistive memories. *SIGPLAN Not.*

[5] Edmund B. Nightingale, John R. Douceur, and Vince Orgovan. Cycles, cells and platters: An empirical analysisof hardware failures on a million consumer pcs. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 343–356, 2011.

[6] Horst Schirmeier, Ingo Korb, Olaf Spinczyk, and Michael Engel. Efficient online memory error assessment and circumvention for linux with rampage. *IJCCBS*, pages 227–247, 2013.

[7] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. Dram errors in the wild: A large-scale field study. In *SIGMETRICS*, 2009.

[8] Charlie Slayman and Mike Silverman. Soft errors in modern memory technology. *CMSE Conference 2011*, September 2011.

[9] J. F. Ziegler and W. A. Lanford. *The Effect of Cosmic Rays on Computer Memories*. 1979.