Publisher
Richard Bowles

Managing Editor
Stuart Douglas

Content Architect
Shih-Lien Lu (Lead)
Qiong Cai
Patrick Stolt

Program Manager
Stuart Douglas

Technical Editor
David Clark

Technical Illustrators
MPS Limited

Technical and Strategic Reviewers
Andy Anderson
Fatih Hamzaoglu
Serkan Ozdemir
Ningde Xie
Rick Coulson
Rich Uhlig
Henry Stracovsky

Intel Technology Journal

Copyright © 2013 Intel Corporation. All rights reserved.
ISBN 978-1-934053-57-7, ISSN 1535-864X

Intel Technology Journal
Volume 17, Issue 1

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4744. Requests to the Publisher for permission should be addressed to the Publisher, Intel Press, Intel Corporation, 2111 NE 25th Avenue, JF3-330, Hillsboro, OR 97124-5961. E-Mail: intelpress@intel.com.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in professional services. If professional advice or other expert assistance is required, the services of a competent professional person should be sought.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Intel may make changes to specifications, product descriptions, and plans at any time, without notice.

Fictitious names of companies, products, people, characters, and/or data mentioned herein are not intended to represent any real individual, company, product, or event.

Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications. Intel, the Intel logo, Intel Atom, Intel AVX, Intel Battery Life Analyzer, Intel Compiler, Intel Core i3, Intel Core i5, Intel Core i7, Intel DPST, Intel Energy Checker, Intel Mobile Platform SDK, Intel Intelligent Power Node Manager, Intel QuickPath Interconnect, Intel Rapid Memory Power Management (Intel RMPM), Intel VTune Amplifier, and Intel Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks

†Other names and brands may be claimed as the property of others.

This book is printed on acid-free paper. ♾

Publisher: Richard Bowles
Managing Editor: Stuart Douglas

Library of Congress Cataloging in Publication Data:

Printed in China
10 9 8 7 6 5 4 3 2 1

First printing: May 2013

Notices and Disclaimers

ALL INFORMATION PROVIDED WITHIN OR OTHERWISE ASSOCIATED WITH THIS PUBLICATION INCLUDING, INTER ALIA, ALL SOFTWARE CODE, IS PROVIDED "AS IS", AND FOR EDUCATIONAL PURPOSES ONLY. INTEL RETAINS ALL OWNERSHIP INTEREST IN ANY INTELLECTUAL PROPERTY RIGHTS ASSOCIATED WITH THIS INFORMATION AND NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHT IS GRANTED BY THIS PUBLICATION OR AS A RESULT OF YOUR PURCHASE THEREOF. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO THIS INFORMATION INCLUDING, BY WAY OF EXAMPLE AND NOT LIMITATION, LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR THE INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT ANYWHERE IN THE WORLD.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

For more information go to http://www.intel.com/performance

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.
Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

INTEL® TECHNOLOGY JOURNAL
MEMORY RESILIENCY

Articles

Foreword 7

Overview 10

Scaling the Memory Reliability Wall 18

Improving Memory Reliability, Power and Performance Using Mixed-Cell Designs 36

STTRAM Scaling and Retention Failure 54

ECC Techniques for Enabling DRAM Caches with Off-Chip Tag Arrays 76

Improving Error Correction in NAND with Dominant Error Pattern Detection 94

Memory Controller–Level Extensions for GDDR5 Single Device Data Correct Support 102

Towards Proportional Memory Systems 118

Error Analysis and Retention-Aware Error Management for NAND Flash Memory 140

A Case for Nonuniform Fault Tolerance in Emerging Memories 166

An Integrated Simulation Infrastructure for the Entire Memory Hierarchy:
Cache, DRAM, Nonvolatile Memory, and Disk 184

Forward ... A Foreword

Professor Bruce Jacob
University of Maryland

Using present-day architectural design concepts to drive forward the design of next-generation large-scale systems is an attractive approach but is inherently misguided; today's methods simply do not scale to tomorrow's sizes, and no amount of forcing will cram that square peg into a round hole. Nowhere is the mismatch more apparent than in the memory system, because the memory system constitutes a majority of the silicon in a large-scale system. The following points illustrate just a few of the engineering challenges facing us:

- *Power.* Power per node in large-scale systems is on the order of 100 W, a conservative number, roughly half of which is dissipated in the memory system (DIMMs dissipate roughly 20 W when in use, which means 20 W per channel; today's systems often have three or more DRAM channels per CPU socket). Future installations are expected to have hundreds of thousands of nodes. Multiplying the two yields power requirements in the 10–100 MW range per installation, roughly half of which will be dissipated by memory. With electricity costing USD 1 million per megawatt-year, following the current design trend will cost tens of millions of dollars per year per installation, just for the electricity to compute, never mind the cost of cooling.
- *Reliability.* Memory chips comprise the largest number of chips in a typical system. For instance, in a typical drawer one will find on the order of 10 CPU chips, 100 supporting chips (I/O, administrative and monitoring, and other glue), and 1000 DRAM chips. For a medium- to large-scale system (10–100 racks of 10–100 drawers each), this would imply something on the order of 1 million DRAM chips system-wide. Reliability in such a system becomes a significant issue: even with DRAM hard-error FIT rates better than 10, this yields a statistically guaranteed hard-error device failure somewhere in the memory system every few days. Soft errors are generally one to two orders of magnitude more frequent than hard errors, meaning that transient errors will occur at the rate of once every few hours.
- *Volume.* One of the more significant costs of a computing installation is the physical plant, which scales with the physical volume of the computing circuitry needed (that is, how much space it takes up). Simply put: more computing performance requires more computing circuitry, which requires a larger building. Figure out a way to reduce the volume of the circuitry required—in particular, the volume of the memory system required (see previous point)—and you can reduce the size of the physical plant needed. Note that there are a handful of obvious ways to reduce the volume of the memory system, including reducing the total number of bits (not particularly appealing while the number of cores is increasing), reducing feature size (the present approach), or changing to another memory technology with significantly different density characteristics. Note also that, besides reducing the cost of construction, reducing the physical size can also reduce the cost of cooling the computing circuitry (for example, by reducing the number of chillers and air handlers needed), a cost that typically represents half the overall power budget.

It should be clear from this brief look at the challenges facing us that moving forward in large-scale system design will require significant work at the memory-system level. Because the memory system imposes such significant limitations (including performance, power, reliability, and space), we cannot move forward without understanding these limitations and fixing them, which is likely to require a redesign of memory systems in general.

These are challenges of *efficiency and reliability*. One way to look at large-scale installations (supercomputers, and most enterprise-computing systems as well) is that they are the world's highest-performance embedded systems. Most embedded systems are only valuable if they are *efficient* (for example, when they run on a battery charge all day long) and *reliable* (work correctly and require little system maintenance). Like embedded systems and unlike typical general-purpose systems, large-scale

installations tend to run the same software 24x7. Like embedded systems and unlike typical general-purpose systems, users of these installations will go to great lengths to optimize their software and often write their own operating systems for the hardware. And, most importantly considering the focus of the special issue you are reading, like embedded systems, efficiency and reliability in large-scale systems is now (or now has become) the key point. In the design of tomorrow's large-scale systems, people care more about efficient and reliable solutions than high-performance solutions—not because they *want* to, mind you, but because they *have* to. Whereas in the past, performance or capacity sometimes came at a high price in power or reliability, today that is no longer an acceptable tradeoff. The best solutions for tomorrow's systems will be the ones that promise *reliability and efficiency*, even if at a modest cost in performance or capacity.

The articles in this special issue of ITJ address precisely these problems and from precisely this perspective. The articles in the *Low Power Cache/Memory* section trade off cache/memory capacity for reliability and lower power. The articles in the *Error-Correcting Codes* section provide advanced reliability techniques wherein reliability is ensured through redundancy. The *Invited Academic* articles address reliability and lifetime concerns of flash memory. Last, the *Evaluation and Infrastructure* article describes a new simulation framework for accurately evaluating memory-system designs that integrate nonvolatile technologies directly into the memory hierarchy.

OVERVIEW

Contributor

James P. Held
Intel Fellow, Intel Labs,
Director, Microprocessor
and Programming Research

Introduction

Resiliency is an important attribute of a system. It enables a system to continue to function correctly, sometimes in a degraded fashion, in the presence of faults, errors, or other variation. There are many ways to increase the resiliency of a system. A simple way to increase resiliency is by over-designing. That is, we include additional margins in the design specification to account for variations, and cover all possible variations or failure scenarios a system may encounter in its expected lifetime. The added margin enables the system to remain robust and fault-free. Obviously, over-design is not the most efficient way to make a system resilient because not all variations or scenarios will occur at the same time. In order to ensure the correct operation, we must design the system for the worst-case combination of all variations while it may be very rare that all these variations happen at the same time. Moreover, it may also be very difficult to anticipate all possible scenarios or variations a system may encounter in its lifetime at the design phase. Resiliency is a design methodology to manage risks through design tradeoffs and is more efficient than over-design.

Memory is a necessary part of any computing system as it is used to store data as well as programs. The amount of memory used has been increasing for all segments of computing devices to accommodate ever-increasing application usages and data. Memory is not only used at the instruction set architecture level; the amount of memory circuits at the microarchitectural level to enhance performance or power has been increasing as well. For example, the amount of cache on a microprocessor chip has been increasing steadily in the last few decades. With the increased amount of memory circuits in a computing system, the chance of a memory-related failure for a system will also increase. Thus, making the memory subsystem resilient will contribute directly to the overall resiliency of any computing system. The computing community possesses a great wealth of knowledge on techniques for designing resilient memory. With continued scaling and new memory technologies emerging, it is timely to examine recent challenges and research results on memory resiliency.

To provide context for the articles in this issue, in this overview we first discuss different abstract error types and general mitigation strategies for them. When then discuss the memory subsystem and review the different types of memory technologies used in a memory subsystem along with the types of errors specific to each memory technology as it is scaled. Some discussion of how to mitigate these errors is also provided. Finally we introduce articles in this issue

of the *Intel Technology Journal (ITJ)* and explain the related memory errors each of them is addressing.

Types of Memory Errors

It is essential to understand the types of errors that may occur in memory in order to come up with efficient management techniques. We can classify memory bit failures into two broad categories: *persistent* and *non-persistent failures*. Persistent failures, such as stuck bits caused by manufacturing defects, remain at fault once they occur. Persistent bit failures can be detected through testing and they contribute to the majority of bit failures. Persistent failures reduce yield and increase cost.

Non-persistent bit failures are bits that exhibit sporadic failing behavior (soft errors). Many times these bits are marginally functional to begin with and failure can be triggered by environmental changes. Failures resulting from radiation particle strikes are a classic example of this category of failures. Since these failures are non-persistent and occur randomly, they cannot be effectively identified with testing. As a result, these failures do not directly contribute to yield loss and instead they affect a unit's failure-in-time (FIT) rate.

Failure Mitigation Techniques

As mentioned, persistent failures can usually be reliably identified using standard memory testing methods. Testing can be done statically or dynamically. Static testing halts the system's normal operation and puts the memory in a separate testing mode. Dynamic testing allows the system to operate normally but is able to isolate faults at the same time. Failures that cannot be discovered through testing require a mechanism to detect the error. Once failures are detected, mitigation techniques can be employed to correct the faults.

All failure mitigation techniques will incur overhead. In general, the more information we know about the failures, the easier we can mitigate the problem.

Resilient techniques fall into two categories. The first category is effective with testable failures. Methods in this category include sparing and disabling. Sparing is a well-known technique for increasing the yield of memory. It consists of designing with spare rows, columns, and blocks that are switched to replace faulty bits. Disabling is another way to increase the yield. Instead of switching the spares to cover the faulty elements, disabling removes the elements from the active list to allow the system to operate at the degraded mode.

The second category utilizes information redundancy to detect random errors and correct them. This category includes all kinds of error correcting codes (ECCs), and they have been shown to be effective in recovering from non-persistent failures.

The Memory Subsystem and Research Issues

Computing systems have evolved rapidly both in their capability and complexity due to the advancement of semiconductor technology in the last few decades. Every component of a system must advance relatively to each other to keep the system balanced including the memory subsystem.

In the recent decades, the CPU performance has increased at the rate of roughly 50 percent per year, while the speed of main memory has improved at a rate of only 7 percent per year. Several architectural techniques were employed to mitigate the gap between memory and processors, including caching.

Additionally, increasing complexity of applications is putting pressure on the amount of memory needed for a system. We must ensure we can continue to scale memory process technology for greater capacity just as we continue to improve processors through CMOS scaling.

There are many challenges in continuing the scaling of memory. First, as we scale memory cells, the storage node becomes smaller and it is more and more difficult to detect what information is stored. As cells become smaller the distance between cells is shortened as well. Cells tend to disturb each other due to the close proximity. All these lead to the need for better resiliency techniques for memory.

In the next section we will briefly introduce types of memory and then discuss some of the issues facing each type. Finally, we will then discuss tradeoffs must be made among several parameters to meet the requirements for the memory subsystem.

Types of Memory

There are many types of memory available for use in a computing system. These memory types can be classified based on their characteristics and functionality. From the functionality point of view, memory can be classified in two broad categories: RAM (random access memory) and ROM (read only memory). These names are somewhat misleading and we will discuss the differences later. We can also characterize memory types according to their characteristics both physically and logically. A memory type with the physical property of retaining its content without power supply is called *nonvolatile memory*. Memory that loses its content without a power supply is called *volatile memory*. Volatility is really not precisely defined and is commonly used quite loosely. Many nonvolatile memory types are volatile as well. They just have a very long *retention* time, for example in the range of years.

A memory type that should retain its content indefinitely is called *persistent* memory. For example, in a computing system we assume files will retain their contents indefinitely. Memory that is not intended to retain its content in a system is called *non-persistent* memory. For example, main memory in a current computing system is non-persistent.

We now come back to the functionality of memory and how various types of memory are implemented physically. RAM stands for random access memory, which can be read and written by users. There are two common types of RAM: static RAM (SRAM) and dynamic RAM (DRAM). They are different in that DRAM loses its content if it is not periodically "refreshed" by rewriting the bits (thus the name dynamic) while SRAM retains its contents as long as the power supply is on.

A typical SRAM cell is formed by two cross-coupled inverters, each with two transistors locked together and two access transistors. It is usually referred to as the 6-T SRAM because of the six transistors used. There are other memory types built with cross-coupled inverters but with different access port circuit structures. These are usually called register file memory. Conversely, a typical DRAM cell is constructed from a capacitor along with a single access transistor. The capacitor is used to store an electrical charge that determines the data content. We call this type of memory more specifically 1T-1C DRAM.

Figure 1(a) and (b) depict the circuit structure of an SRAM and DRAM cell, respectively. These are single ported memory cells. A port is an access point into the memory content. Ports for SRAM and DRAM cells in Figure 1 are both readable and writeable. As we scale SRAM and lower the supply voltage, the inherent conflict for read and write will surface and cause either write instability or read instability, for example.

As we scale DRAM, the capacitor used to store information becomes smaller as well. It is harder to detect the charge when it becomes too small. Also, when DRAM cell capacitance is small, even a small amount of charge lost due to leakage may cause the data stored to be lost.

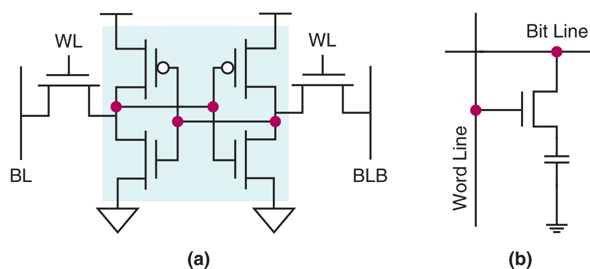


Figure 1: (a) SRAM and (b) DRAM circuits
(Source: Intel Corporation, 2013)

There are other types of RAM with various numbers of transistors and circuit elements depending on how the access ports are constructed and what circuit element is used to store the data. For example, a general type of RAM based on resistance switching is called *resistive RAM* or *R-RAM*. R-RAM uses a variable resistance circuit element instead of a capacitor to store the data content. Figure 2 illustrates the circuit structure of a resistive memory or R-RAM cell. There are different types of R-RAM depending on how the circuit element R is implemented. The article titled "STTRAM Scaling and

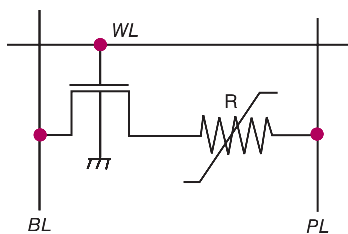


Figure 2: An R-RAM memory cell (Source: Intel Corporation, 2013)

Retention Failure" in this issue will discuss one type of R-RAM that readers can learn more from. STTRAM stands for Spin Transfer Torque Random Access Memory (STTRAM) and it belongs to the "magnetic tunnel junction" family of R-RAM. Another popular type of R-RAM that has attracted much attention is phase change memory (PCM). PCM relies on the fact that some materials have different resistivity when they are in crystalline or amorphous forms (phases). There are other types of R-RAM. They fall into a few families depending on the way their storage elements are built. There is also the R-RAM based on interfacial switching caused by oxygen vacancy drift. All R-RAM memory types are *memristors*, a type of general circuit element proposed by Chua.[1][2]

ROM stands for read-only-memory. It can also be accessed randomly with an address but it is set once, usually at design or manufacturing time, and can only be read thereafter. There is also programmable ROM (PROM), which can be programmed in the field but sometimes with limited programmability frequency. For example fuses can be considered as a one-time programmable ROM. A large group of PROMs is erasable and can be reprogrammed multiple times. Depending on how they are erased they are called EPROM (erased with ultraviolet light) and EEPROM (erased electrically). Since erasing takes a long time for EEPROM, it is inefficient to erase a small chunk first and then program it with new data. A novel design was made to erase a large block of cells at the same time in a "flash" and later program the erased part in smaller chunks thus aggregating the erase time by operating on many bits in parallel. This type of EEPROM is called *flash memory*. Depending on the way circuits for the storage nodes are organized, flash memory can be divided in two types: NOR and NAND flash. NAND flash memory is very popular now due to its density and has wide usage in many consumer products. It has also revolutionized the storage industry with solid-state drives (SSDs). The number of times a memory cell can be programmed and erased is usually called *endurance* in the community. Flash has limited endurance, and when it is used to build SSDs, *wear-leveling* algorithms are used to distribute the number of reprogramming operations to satisfy the endurance limit requirement. The articles "Improving Error Correction in NAND with Dominant Error Pattern Detection" and "Error Analysis and Retention-Aware Error Management for NAND Flash Memory" in this issue discuss different ways to mitigate possible endurance failures.

Tradeoffs of Memory Parameters

As mentioned briefly previously, another way to classify memory is by its device characteristics. One particular characteristic is the ability to retain memory content when power is removed from the circuit element. We call these memory types nonvolatile memory (NVM). For examples, flash memory, EEPROM, and STTRAM are all nonvolatile memory types. DRAM and SRAM are volatile memory types. It turns out the volatility is not a binary parameter. We really cannot say a memory is absolutely nonvolatile because all memory has a limited retention time. Moreover, retention time can be adversely affected by how often a cell is erased and programmed, or

cycled. For example a typical NAND flash memory cell can be erased and reprogrammed in tens of thousands of times only. Cells that are infrequently cycled have longer retention time while frequently cycled cells have shorter retention time on average. Similarly, STTRAM cells can trade off physical size for better retention ability. A larger STTRAM cell will have more magnetic energy stored, which will reduce the probability of having the cell flipped.

Of course each memory type will differ in other characteristics such as access time, density, and power. Some even have different read and write access times. These characteristics are all interrelated. Usually a less dense memory technology will have better access time. For example SRAM is faster than DRAM in general. System architects take advantage of these different characteristics to optimize for performance, power, and cost by using a memory hierarchy. Some levels of the hierarchy may be on the same chip as the processing units. We call these levels *embedded* memory. There is a trend to include more and more levels of memory on the same chip or the same package of the processing unit to improve efficiency because going out of a chip tends to increase power and cost with a corresponding impact on performance. In general, smaller amounts of a faster memory are used as caches for larger amounts of a slower storage; for example, solid state disks (SSDs) based on NAND flash are often used as a cache for rotating disks.

The speed of a memory cell can also be traded off with size for the same memory. An SRAM cell with larger transistors tends to be faster than another SRAM cell that is implemented with smaller transistors, but the larger transistors will lead to a larger array area. The larger physical size also enhances reliability. For example an SRAM cell with larger transistors will have less sensitivity to variations and thus give better tolerance to supply noise. In general we need to address two issues with memory: retention of data content and endurance due to repeated access. There are many parameters that can vary and we must consider the tradeoffs.

The Articles in This Issue

This issue of the *Intel Technology Journal* is grouped into four sections. The first section covers embedded memory. It starts with an interesting article titled “Scaling the Memory Reliability Wall,” which begins with a general review of the causes of memory failures and then argues that any resiliency mechanism adopted by a system must be adaptive to minimize overhead. It then presents two example techniques used to gain power efficiency without sacrificing much performance for designs where reduced power is the main objective. The second article in this section presents an innovation using non-uniform cells to improve reliability, power, and performance. It is based on the fact that larger cells have better reliability but shows that we don’t need to make all cells large; we only need to make sure large cells are used to store information that must be more reliable. The third article presents the opportunities and challenges of using STTRAM, an emerging memory technology, for embedded memory. It describes this relatively new technology, which utilizes spin-transfer

torque (STT) to store information. Since this technology is compatible with a standard CMOS process[3], it has the potential to be the technology to replace SRAM as the future embedded memory technology. However, the retention stability of a STTRAM cell is proportion to the magnetic energy stored. The energy is a function of the material as well as the volume. If the height of the junction also scales with the length and width, then the volume is reduced as well, causing it to be less stable. This article presents a detailed model of the scaling impact on reliability and examines many approaches to address the issues resulting from scaling.

The second section of this issue also contains three articles on the topic of error correction codes (ECCs). Due to access latency differences of different types of memory, different types of ECC can be employed that are tailored to the memory. The number of check bits required for ECC is related to the correction ability and the information length. Theoretically, for binary coded information the number of checks bits is proportional to $t \times ceiling[\log(n)+1]$ where t is the number of bits the code can correct and n is the number of information bits. It is more efficient, in terms of number of check-bits required, to protect a larger information word with higher number of correctable bits coverage. For example, it only needs 104 extra bits to protect 512 bytes of data with 8-bit correction capability. It will need 576 extra bits (more than 5 times) to protect 16 segments of data each with 32 bytes of data with 4-bit correction capability. These two schemes satisfy certain error coverage given a fixed bit error rate. However, more correction capability with longer data length means more complex decoding logic. One can trade off logic complexity with multiple cycle decoding. Multiple cycle decoding means greater latency overhead due to ECC. Certain memory has longer access latency and adding some extra latency for ECC may not cause any performance issue. The first article in this section provides some new error correcting codes for short latency memories. The main insight of this article is to arrange the generation and decoding matrix in a certain way to cover adjacent errors. The second article of this section is trying to take advantage of the error characteristics and try to tailor the ECC in a way to cover dominate error patterns. In general the more we know about the error behavior and probability the better we can cover the error with more efficient codes. The third article of this section describes needed modifications for a memory controller to provide Chipkill*[4] support for memory technologies that inherently have no RAS support for memory contents protection. *Chipkill* is an ECC technique to cover DRAM memory device failures. Modern systems use DRAM DIMMs to implement their main memory. Each DIMM is made out of several individual DRAM chips. Chipkill can recover a single device failure on a DIMM. Specifically, this article focuses on how to provide single device Chipkill support for GDDR5 memories.

The third and fourth sections of the issue contain four articles from academia. The university authors are recipients of either Intel sponsored research funding or Intel Young Faculty Awards. All of them are doing active research on memory architecture and memory resiliency. They bring different perspectives

on how to address issues related to memory resiliency. The first three articles propose error management techniques for different types of memory. We appreciate their participation in this special issue and look forward to continued collaboration on this research topic. Finally, we conclude with an important article on simulation infrastructure. Any proposed solution to this critical topic must be validated through simulations. The University of Maryland has taken an existing full system simulation infrastructure and extended it to include memory simulation models from cache to disk. It allows researchers to evaluate tradeoffs with good accuracy.

References

[1] L. O. Chua, "Memristor—The Missing Circuit Element," IEEE Transactions on Circuit Theory, CT-18 (5): 507–519, 1971.

[2] L. O. Chau, "Resistance Switching Memories are Memristors," Applied Physics A 102 (4): 765–783, 2011.

[3] M. Hosomi et al., "A Novel Nonvolatile Memory with Spin Torque Transfer Magnetization Switching: SPIN-RAM," IEEE International Electron Devices Meeting, Dec. 2005.

[4] Timothy J. Dell, A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory, IBM Microelectronics Division, 1997.

Author Biography

As director of Microprocessor and Programming Research, Jim Held leads a team conducting research in microarchitecture, parallel computing and programming systems to develop key technologies for future microprocessors and platforms.

Since joining Intel in 1990, Held has served in a variety of positions working on computer supported collaboration technology and Intel Native Signal Processing (NSP) infrastructure. He served as staff principal architect in the Media and Interconnect Technology Lab in IAL and as the Lab Director in CTG, managing the Volume Platforms Lab. As a Senior Principal Engineer in the Microprocessor Technology Lab, he conducted research on extensible processor architecture, multi-core processor architecture and helped develop Intel's virtualization technology strategy. From 2005–2011 he led a virtual team of senior architects conducting Intel Lab's Tera-Scale Computing Research.

Before coming to Intel, Held worked in research and teaching capacities in the Medical School and Department of Computer Science at the University of Minnesota. He is a member of the IEEE Computer Society and the Association for Computer Machinery (ACM).

Held earned a BS in Chemical Engineering in 1972 and an MS (1984) and PhD (1988) in Computer and Information Science, all from the University of Minnesota.

SCALING THE MEMORY RELIABILITY WALL

Contributors

Chris Wilkerson
Intel Labs

Alaa Alameldeen
Intel Labs

Zeshan Chishti
Intel Labs

Technology scaling reduces the size of memory cells, continuing to deliver dramatic improvements in memory density. Increasing memory density, however, also increases susceptibility to known failure types. Furthermore, each new process technology introduces the potential for new, unanticipated failures. In this article, we highlight some of the common failure modes in today's memory technologies as well as uncommon failure modes that may grow in significance on future technology nodes. We describe a number of approaches to efficiently compensate for failure-prone memory, and argue that a key ingredient in resilient systems is the ability to compensate for unanticipated memory failures.

Introduction

Technology continues to scale, driving dramatic improvements in memory density. With decreasing geometries and increasing capacities, ensuring reliable operation of the memory system becomes a greater challenge. The industry's prescription for reliable memory has three components:

"With decreasing geometries and increasing capacities, ensuring reliable operation of the memory system becomes a greater challenge."

- Predicting the locations of failing bits, typically through testing.
- Removing failing bits, typically through the application of some type of redundancy.
- Compensating for unpredictable bit failures.

Memory testing is typically the first step after manufacture of most high volume memory technologies, such as NAND flash, DRAM, and SRAM. Many of the tested memories include small numbers of bad bits. These failures may be clustered due to a marginality in a shared structure such as a sense amp (column failures), or a row. Or these bits may be randomly scattered throughout the array due to random defects in the bit cell. To maximize yield, memory is designed with redundant rows and columns, allowing the repair of clustered bit failures along a row or a column, or even the repair of a few isolated bit cell failures. Finally, after the memory is shipped and assembled in the system, additional unanticipated failures may emerge. These failures may have escaped tests for a number of reasons:

"These failures may have escaped tests for a number of reasons."

- Pattern sensitivity: the bit failures may be due to cell to cell coupling activated only the in context of very specific data patterns not exercised during memory testing.
- Aging: over time, reliable structures (bit cells, sense amps, and so on) may have degraded to the point that previously reliable bits begin to fail.
- Change in conditions: conditions may have changed relative to those anticipated during test.

- Erraticism/variability: the physical mechanism that results in cell failure may only manifest from time to time. As a result, identical testing conditions may vary in their ability to expose failures.

Resiliency

In this article, we examine some of the challenges that occur in embedded CPU memory. Although the bulk of the discussion focuses on CPU caches implemented using SRAM, the techniques we described are broadly applicable to a variety of different memory types. We begin with a discussion of low voltage and its impact on SRAM error rate in CPU caches. Next, we introduce the concept of adaptivity and how it can be exploited to reduce overhead in the context of two different test-based repair mechanisms, bit-fix and word-disable. In the section "Reducing the Need for Tests," we describe MS-ECC, a third technique that exploits adaptivity and reduces the need to rely on memory tests to identify bit errors. The section "Optimizing for the Common Case" introduces this concept and shows this approach can be used to minimize overhead in two different mechanisms, VS-ECC and Hi-ECC. The first of these mechanisms, VS-ECC, is noteworthy for the additional flexibility it allows the system in performing tests. The second, Hi-ECC, minimizes the reliance on testing by constructing an ECC safety net that continuously checks for bit errors during use. The final section summarizes our conclusions.

SRAMs and Vmin

Small signal arrays (SSAs), such as static RAM (SRAM), are probably the most common embedded memory type, due to their compatibility with typical logic manufacturing process. SSAs typically suffer from variations induced during the manufacturing process, making them unreliable at low voltages. Intra-die random dopant fluctuations (or RDFs) play a primary role in cell failure by arbitrarily impacting the number and location of dopant atoms in transistors, resulting in different voltage thresholds (Vths) for matched SRAM devices.[1][2] These variations can cause adjacent devices in a single SRAM cell to have different strengths, reducing the stability of the cell. These defective cells, randomly distributed throughout large memory structures may prevent caches from operating below a minimum voltage often called Vmin (or Vccmin).

"SSAs typically suffer from variations induced during the manufacturing process, making them unreliable at low voltages."

"…Defective cells, randomly distributed throughout large memory structures may prevent caches from operating below a minimum voltage…"

Voltage scaling is one of the most effective ways to reduce the power consumed by a microprocessor since dynamic power is a quadratic function of voltage. Voltage scaling can also effectively reduce static power due to leakage since leakage is an exponential function of voltage. As a result, Vmin is a critical parameter that constrains our ability to reduce a particular design's power consumption. Overcoming Vmin allows designs to operate at lower voltages, improving energy consumption and battery life for handheld and laptop products. Figure 1 shows the probability of failure (Pfail) of an SRAM cell (pfail bit) and a number of multi-bit structures as a function of voltage. As one expects, as the probability of failure for a single bit increases (X-axis), the probability of failure for structures that consist of these bits (Y-axis) also increases. Each line depicts a single structure such as a single byte (8-bit), or a single cache line (512-bit). As the number of bits in a structure increases, so does the probability of at least one of those bits failing.

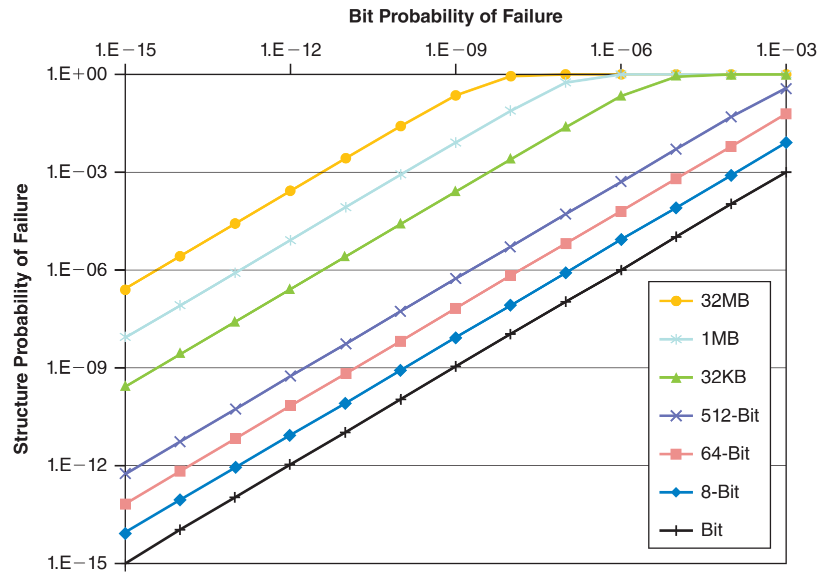


Figure 1: Probability of a failure for different structures as a function of single bit failure probability.
(Source: Intel Corporation 2013)

A number of circuit techniques have been proposed to improve SRAM reliability at low voltages.[3][4][5] These typically involve upsizing devices or employing multiple voltages. Recent work has also identified a number of promising architectural approaches. These typically attack the Vmin problem by augmenting memory with the ability to adapt in the presence of bad bits. One of the key advantages of architectural approaches is that they can be adaptive, incurring additional overhead only when operating conditions demand it.

Adaptivity

"Adaptivity minimizes the overhead of resiliency when conditions preclude the possibility of bit errors."

Adaptivity minimizes the overhead of resiliency when conditions preclude the possibility of bit errors. The system can customize the strength of the repair mechanism to its operating conditions, incurring the highest overhead during the worst-case operating conditions. In previous work[6], we attacked the Vmin problem in SSAs using an adaptive approach, reconfiguring cache resources depending on operating conditions.

We observed that operating modes that require the minimal voltage (and power) may be willing to reduce cache capacity in exchange for reduced voltage. In light of this, we proposed two ways to design a cache to operate at both high and low voltage. These schemes exploit this by incurring the overhead of repairing defects only at low voltage. With minimal overhead, both schemes significantly reduce the Vmin of a cache in low-voltage mode while reducing performance marginally in high-voltage mode. Both schemes achieve a significantly lower overhead compared to ECC-based defect tolerance schemes at low voltage. Both mechanisms trade off cache capacity at low voltages, where performance (and cache capacity) may be less important, to gain the improved reliability

required for low voltage operation. At high voltages where performance is critical, both mechanisms have minimal overhead maximizing the availability of cache resources. We observed a 10-percent performance loss when operating at low voltage when compared to an ideal cache that achieves the same voltage with no overhead. The adaptivity of these techniques avoids the overhead and the performance penalty it applies when operating at high voltage.

To support different operating modes, our adaptive approach repurposed cache resources depending on the operating mode. The mechanisms described in this work rely on memory tests to identify defective portions of the cache and reconfigure the cache to ensure running programs avoid those portions.

"To support different operating modes, our adaptive approach repurposed cache resources depending on the operating mode."

After using memory tests to identify defective portions of the cache, two schemes identify and disable defective portions of the cache at different granularities: individual words or pairs of bits. One scheme, called word-disable, disables 32-bit words that contain defective bits.

Disabling Words

In the word disable scheme, defective words are simply disabled and physical lines in two consecutive cache ways combine to form one logical line where only non-failing words are used. This cuts both the cache size and associativity in half in low-voltage mode. Each line's tag includes a defect map (one bit per word, or 16 bits per 64-byte cache line) that represents which words are defective (0) or valid (1).

The word-disable mechanism isolates defects on a word-level granularity and then disables words containing defective bits. Each cache line's tag keeps a defect map with one bit per word that represents whether the word is defective (1) or valid (0). For each cache set, physical lines in two consecutive ways combine to form one logical line. After disabling defective words, the two physical lines together store the contents of one logical line. This cuts both the cache size and associativity in half. Figure 2 illustrates how this works in more detail.

"The word-disable mechanism isolates defects on a word-level granularity and then disables words containing defective bits."

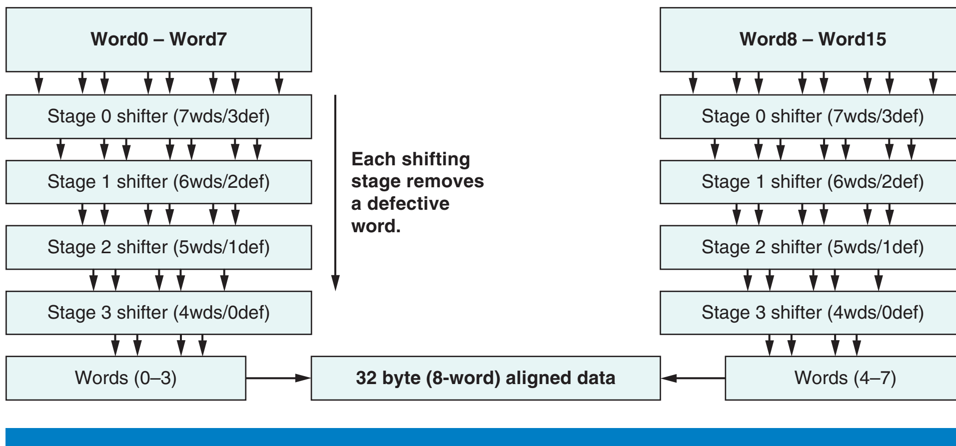


Figure 2: Word disable combines working words from two lines (Source: Chris Wilkerson, et al., 2008[6])

"A single logic cache line is divided into two halves, split between the two physical lines each with a maximum of four defective words."

Assume you have an 8-way cache containing 32-byte lines. For each cache set, eight physical lines correspond to four logical lines. We use a fixed mapping from physical lines to logical lines: Lines in physical ways 0 and 1 combine to form logical line 0, lines in physical ways 2 and 3 combine to form logical line 1, and so on. A single logic cache line is divided into two halves, split between the two physical lines each with a maximum of four defective words. The first physical line in a pair stores the first four valid words, and the second stores the next four valid words of the logical line for a total of eight 32-bit words.

To obtain the 32-byte data in aligned form, we use two four-stage shifters to remove the defective words and aggregate working words as shown in Figure 3. As a result, in low-voltage mode, the capacity of each set is effectively halved to 4-ways.

"Since each level of muxes eliminates a single defective word, we require four levels to eliminate four defective words."

Figure 3 illustrates the logic used to disable and remove a single defective word from a group of words. Starting with the defect map, we extract a 1-hot repair vector identifying the position of a single defective word. In the figure, the vector "0010" identifies the third word from the left as defective. The decoder converts the 1-hot vector into a Mux-control vector containing a string of 0s up to (but not including) the defective position followed by a string of 1s. This has no effect on the words to left of the defect, but each of the words to the right of the defective word shifts to left, thereby "shifting-out" the defective word. Since each level of muxes eliminates a single defective word, we require four levels to eliminate four defective words.

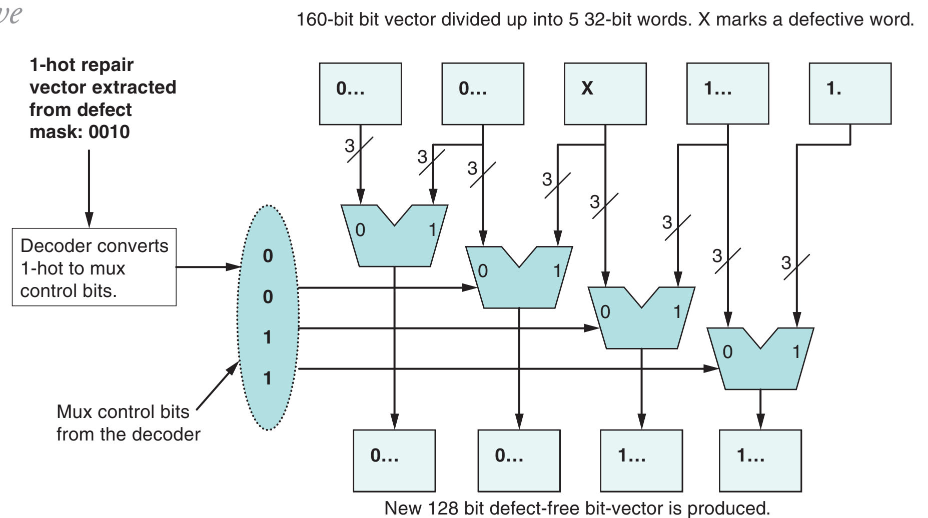


Figure 3: Removing bad words from a cache line
(Source: Chris Wilkerson, et al., 2008[6])

Fixing Bits

The bit-fix mechanism differs from the word-disable mechanism in three respects. First, instead of disabling at word-level granularity, the bit-fix mechanism allows groups of two bits to be disabled. These defective pairs are groups of two bits in which at least one bit is defective. Second, for each defective pair, the bit-fix mechanism maintains a 2-bit patch that can be used to correct the defective pair. Third, the bit-fix mechanism requires no additional storage for repair patterns, instead storing repair patterns in selected cache lines in the data array. This eliminates the need for the additional tag bits required to store the defect map in the word-disable mechanism. In high performance mode when the bit-fix algorithm is unnecessary, the repair pointers must be stored in memory.

"The bit-fix mechanism differs from the word-disable mechanism in three respects."

To illustrate how bit-fix works, consider an 8-way set associative cache with 64-byte lines. The bit-fix scheme organizes the cache into two banks, each containing four ways. The repair patterns for three cache lines fit in a single cache line; therefore, we maintain a single fix-line (a cache line storing repair patterns) for every three cache lines. A fix line is assigned to the bank opposite to the three cache lines that use its repair patterns. This strategy allows a cache line to be fetched in parallel with its repair patterns without increasing the number of cache ports.

Figure 4 contains a high level depiction of how bit-fix works. On a cache hit, both the data line and a fix line are read. In this figure, we fetch the data line from Bank A and the fix line from Bank B. The data line passes through *n* bit shift stages, where *n* represents the number of defective bit pairs. Each stage removes a defective pair, replacing it with the fixed pair. Since the fix line may also contain broken bits, we apply SECDED ECC to correct the repair patterns in the fix line before they are used. After the repair patterns have been fixed, they are used to correct the data line. Repairing a single defective pair consists of three parts. First, SECDED ECC repairs any defective bits in the repair pattern. Second, a defect pointer identifies the defective pair. Third, after the defective pair has been removed, a patch reintroduces the missing correct bits into the cache line.

"Repairing a single defective pair consists of three parts."

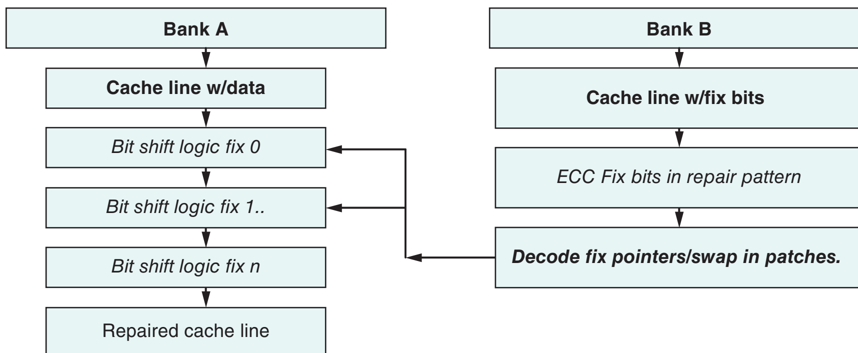


Figure 4: Applying repair patterns in bit-fix
(Source: Chris Wilkerson, et al., 2008[6])

Since bit-fix stores the repair pointers in the cache, all reads and writes to the cache are coupled with a request for repair patterns. Reads rely on the repair patterns to repair broken bits. For writes, repair patterns indicate broken bits, ensuring they are avoided during the write. Patches must be extracted during writes and written into fix lines for use in future reads. When the system operates in modes where bit errors are unlikely, the repair capability provided by bit-fix will be superfluous and the cache capacity should be reclaimed to maximize performance. In these modes, bit-fix requires storage elsewhere in the system (main memory, for example) to hold repair pointers while they are not in use.

Figure 5 compares the probability of failure for a 32-MB cache augmented with the bit-fix and word-disable schemes. The dotted line indicates a hypothetical target for the probability of failure for the whole 32-MB cache; this can also be thought of as yield loss. We've chosen 1/1000 as a target for the comparisons we make in this article, although in an actual design the target will depend on a number of factors including the overall yield target of the product and the likelihood of other structures failing. As shown in Figure 5, bit-fix and word-disable both dramatically improve the ability of cache to tolerate bit errors. A conventional 32-MB cache could meet our hypothetical target of 1/1000 with a bit error rate of about one failure for every 10^{12} bits. In contrast, both word-disable and bit-fix tolerate much higher bit error rates of about one failure for every 1000 bits.

"…Word-disable and bit-fix tolerate much higher bit error rates of about one failure for every 1000 bits."

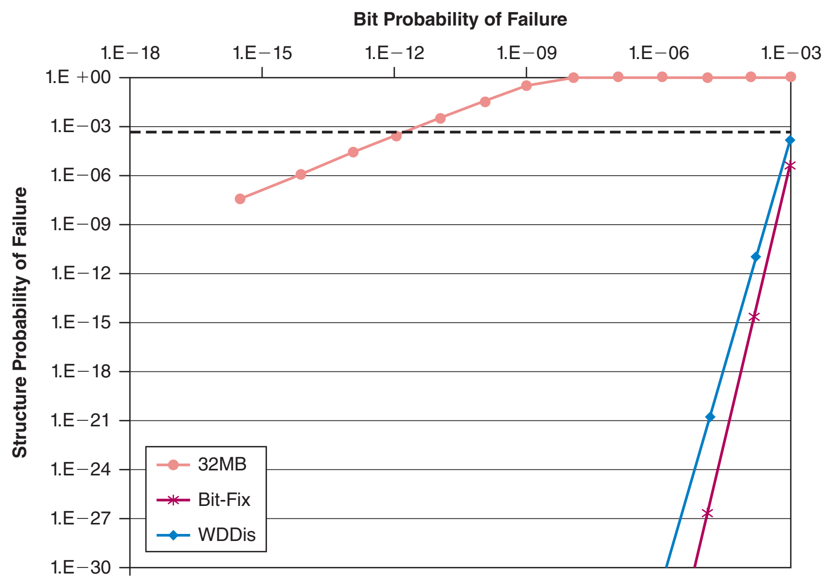


Figure 5: Bit-fix and word-disable compared to baseline
(Source: Intel Corporation, 2013)

Reducing the Need for Tests

The approaches described in previous sections suffer from a reliance on testing to identify defective bits. As technology scales, memory may become

more difficult to test reliably for a number of reasons. First, the time to run memory tests typically grows as a function of the number bits in the memory. Simple tests may grow as a linear function of the memory capacity while more sophisticated tests designed to screen for pattern sensitivity can be much more complex. Second, increasing density may make it harder to isolate bit cells, resulting in more pattern-sensitive failures. Third, memory circuits may become more susceptible to aging-related failures that occur after the part has been tested and shipped. Finally, bit cells implemented in future technologies may be subject to erratic changes in device characteristics that can intermittently change the characteristics of the bit cell even after the product has been shipped.[7]

To address this, we developed Multi-bit Segmented Error Correcting Code (MS-ECC)[8], an adaptive approach that minimizes the role of testing. In place of tests, MS-ECC relies on error correcting codes (ECCs) to identify and correct bit errors after they occur. Encoding and decoding multi-bit error correcting codes can require complex logic and MS-ECC introduces two techniques to reduce this overhead. First, it applied a class of simple but costly error correcting codes called orthogonal Latin square codes (OLSCs). Second, it used segmentation to allow processing of different portions of a cache line in parallel. We describe these in more detail in the following section.

"MS-ECC relies on error correcting codes (ECCs) to identify and correct bit errors after they occur."

Reducing Complexity of Error Correcting Logic

MS-ECC strives to provide an architecture that uses codes to correct several bits per cache line, without incurring the high logic overhead of multi-bit BCH codes. To do this we rely on a simple class of error correcting codes called orthogonal Latin square codes (OLSCs). Although OLSCs require more storage than BCH codes, the use of OLSCs minimizes the cost of the coding/decoding logic. We also employ segmentation, dividing the cache line up into 8-byte segments and providing separate codes for each segment. Both the use of OLSCs and segmentation reduce logic overhead at the cost of increased storage cost for the code itself.

Orthogonal Latin Square Codes

Conventional ECC implementations are based on BCH codes and are tailored to fix one (SECDED) or two errors (DECTED). BCH codes optimize storage overhead (that is, number of check bits) at the cost of logic complexity. The complexity and latency of these codes grow rapidly with the increase in the number of error corrections. To minimize the logic required for multi-bit error correction, MS-ECC needs to use an error correction code whose complexity scales well with the number of error corrections. Hsiao et al.[8] proposed a coding methodology called orthogonal Latin square codes (OLSCs) to correct multi-bit errors. While OLSCs require more check bits than traditional ECCs, they have modular correction hardware, lower logic complexity, As a consequence, OLSCs can be encoded and decoded faster than traditional ECCs implementations using BCH codes.[8]

"BCH codes optimize storage overhead (that is, number of check bits) at the cost of logic complexity."

OLSCs use the general principle of a redundant system based on majority voting, such as triple modular redundancy (TMR). TMR "votes out" any incorrect behavior by having three copies of the design and adopting the results from two out of the three modules as the correct one. Instead of duplicating each data bit three times, an OLSC encodes "orthogonal" groups of bits to form check bits. At decoding time, each data bit generates the final value through a voting process from a group of orthogonally coded data and check bits. Thus, an OLSC does not need to generate a syndrome but can "correct" errors directly from majority voting.

"OLSCs provide a number of benefits for implementing logic, their use requires many more bits to provide a particular level of error correction relative to alternatives such as BCH codes."

Although OLSCs provide a number of benefits for implementing logic, their use requires many more bits to provide a particular level of error correction relative to alternatives such as BCH codes. For conventional BCH codes, correcting a single bit error in a 512-bit cache line requires t log(n) bits; where t is equal to the number of bits to correct, 1 in this case; and n is equal to the number of data bits, 512 in this case. In contrast, an OLSC requires 2 t sqrt(n) bits, growing as the square root of the number of data bits rather than the log. In this case, this means a typical BCH code would require 9 bits and the OLSC would require 2 1 24, or 48 bits to repair a single bit error.

Segmentation

Generating each check bit for an OLSC requires computing a parity for orthogonal groups of sqrt(n) data bits. As a result, we can further simplify the encoding and decoding logic by reducing the size of data word through segmentation. Figure 6 shows an example of multi-bit segmented ECC with eight 64-bit segments in each 512-bit line. On a read hit, as shown in Figure 6(b), we fetch both the data line and the corresponding ECC line. There are separate ECC decoders for each of the eight segments that decode segments in parallel by using information from both the data and ECC ways. The decoded segments are then concatenated to obtain the entire 512-bit line.

Data Bits | Check Bits

First Error Correction

Second Error Correction

H matrix

ENCODE:

C_0 = XOR(d_0, d_1, d_2, d_3)
C_1 = XOR(d_4, d_5, d_6, d_7)
: :

DECODE:

D0 = Majority(d_0, (1)
XOR(C_0, d_1, d_2, d_3), (2)
XOR(C_4, d_4, d_8, d_{12}), (3)
XOR(C_9, d_5, d_{10}, d_{15}), (4)
XOR(C_{12}, d_5, d_{11}, d_{13})), (5)

D1 = Majority(d_1, (1)
XOR(C_0, d_0, d_2, d_3), (2)
.....))

Figure 6: H matrix for a 16-bit OLSC
(Source: Zeshan Chishti, et al., 2009[8])

Segmentation reduces latency through the use of parallel decoders and reduces logic cost by reducing the number of inputs to each check bit parity tree. But these benefits also come with additional costs. In general, smaller segments increase the cost of storing the codes. Figure 7 compares the bit overhead of protecting a 512-bit line with both OLSC and BCH codes. The cost varies depending on size of the segments each code protects. A single code protecting the entire 512-bit word minimizes cost. The smallest segment results in the highest cost. Although it's true that segmentation also increases the level of protection (a code that allows 4-bit errors for every 512-bit line offers less protection than a code that corrects 4-bits for every 32-bit segment), the additional protection segmentation offers is minimal. This is illustrated in more detail in Figure 8, where you can compare the probability of failure of a 32-MB cache with a single SECDED code for each cache line and a second with a SECDED code for each 8-byte segment. Figure 8 also depicts the ability of MS-ECC to tolerate bit errors. MS-ECC succeeds in tolerating very error rates, as high as 1 bit error in every 1000 bits, like the mechanism discussed in the previous section. It achieves this while avoiding the need for extensive tests to identify bit errors but also with a significant cost of 50 percent of the total cache capacity.

"Segmentation reduces latency…"

"In general, smaller segments increase the cost of storing the codes."

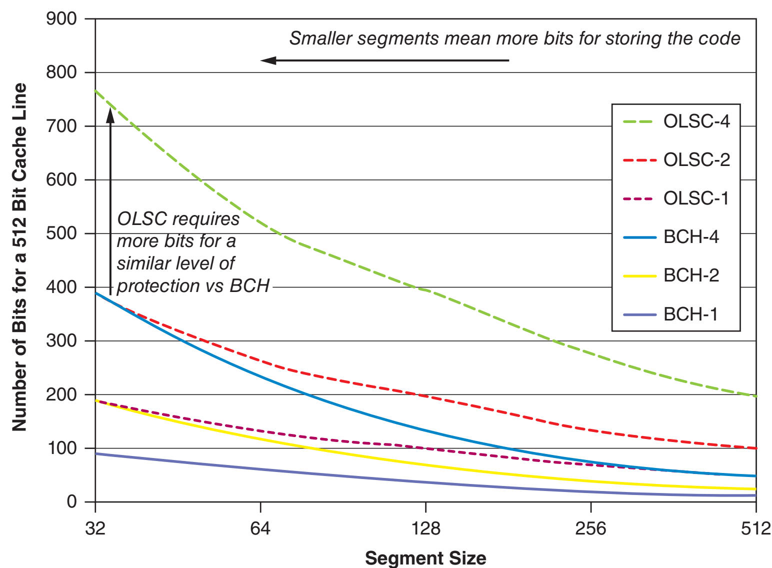


Figure 7: BCH, OLSC, segment size in code choices
(Source: Intel Corporation, 2013)

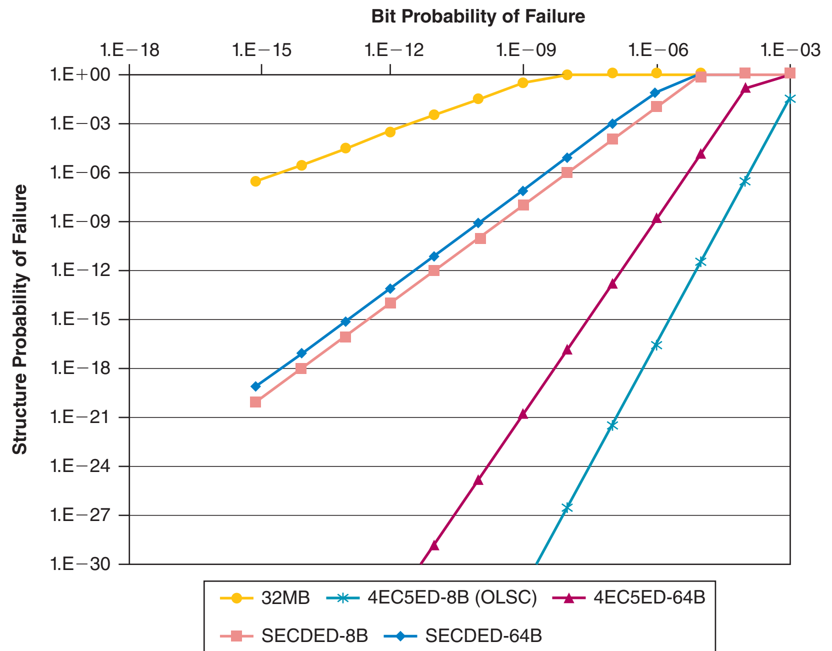


Figure 8: OLSCs and impact of segment size
(Source: Intel Corporation, 2013)

Compensating for Increased Code Storage

"MS-ECC mitigates the increased cost of code storage through the use of adaptivity…"

MS-ECC mitigates the increased cost of code storage through the use of adaptivity similar to what's described in [3]. As in [3], MS-ECC makes the entire cache capacity available at high voltage, but the use of cache resources to store repair information reduces cache capacity at low voltage. To better understand how this approach works, consider an 8-way set associative cache with 64-byte lines.

When using the codes, we divide the eight physical ways in each set amongst data and ECC ways. The ratio of data ways to ECC ways depends on the desired reliability level. If the operating system chooses to improve reliability it could adjust this ratio to increase redundancy at the cost of cache capacity. In this case, we assume each data way comes with a corresponding way storing error correcting codes. We use a fixed mapping to associate data ways with their corresponding ECC ways, as shown in Figure 9(a): physical way 1 stores the ECC for physical way 0, physical way 3 stores the ECC for physical way 2, and so on. Thus, when using the codes, cache capacity and associativity are halved, resulting in a reduction in from eight ways to four ways per cache set. On a write hit to the L2 cache, shown in Figure 9(c), we first use the ECC encoders to obtain the ECC for the data line. Like the ECC decoders, there are separate encoders for each segment that perform ECC encoding in parallel. We then write the new data to the data line and the new ECC to the corresponding ECC line. A similar encoding is performed when a new line is brought into the L2 cache upon a cache miss. We note that when using ECC, each cache access requires both the data and ECC ways to be read. In light of this, the best performance would be achieved if the cache was banked with data and ECC placed in opposite banks, allowing both to be read simultaneously.

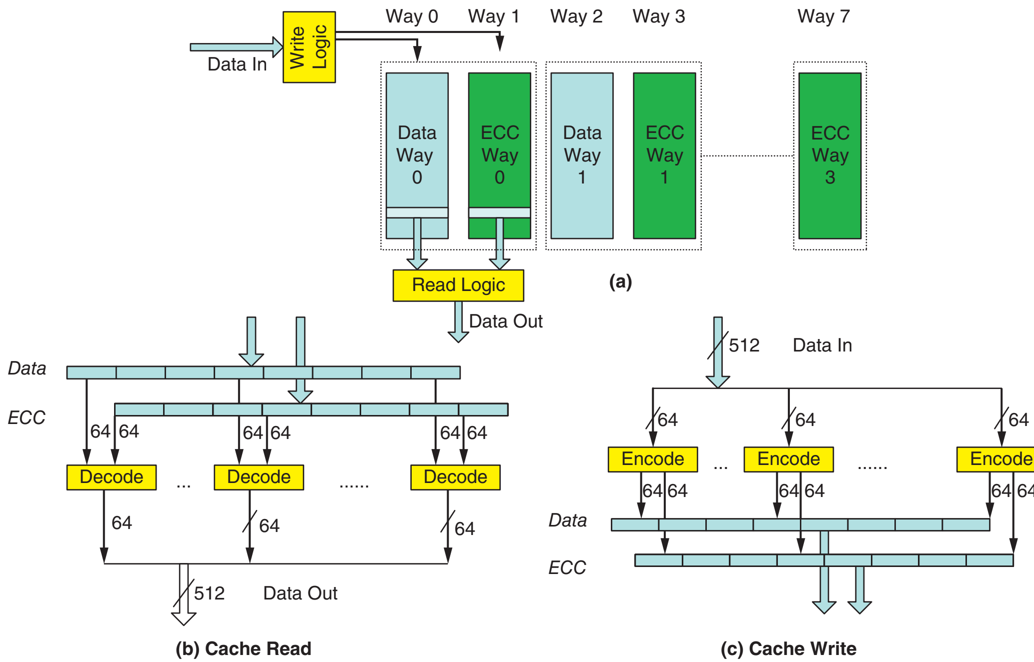


Figure 9: Multi-bit segmented ECC with eight 64-bit segments (Source: Zeshan Chishti, et al., 2009[8])

The MS-ECC approach favors simple, high-overhead codes under the assumption that the high cost of code storage will have little impact when operating at low voltage. This makes sense for systems where memory reliability doesn't present a problem in typical operating conditions.

However, the role of adaptivity is diminished in systems that spend the he bulk of their time operating in modes that require the use of MS-ECC without adaptivity to mitigate the high cost of storing the codes, the overhead of MS-ECC becomes unacceptable. In light of this, we examine other approaches to reduce the overhead of coding.

"…the high cost of code storage will have little impact when operating at low voltage."

"…without adaptivity to mitigate the high cost of storing the codes, the overhead of MS-ECC becomes unacceptable."

Optimizing for the Common Case

The mechanisms reviewed so far represent two extremes: the first two approaches (bit-fix and word-disable) relying solely on testing to identify bit errors, the second (MS-ECC) relying solely on high strength codes. Although memory testing may suffer from a number of challenges, it remains a valuable way of locating bit errors. Ideally, we'd construct a system that could benefit from the information testing provides, while protecting against errors that testing fails to capture. Such a system could rely on memory tests to characterize memory and rely on error correcting codes to compensate for test escapes. VS-ECC, Variable-Strength ECC, proposed by Alameldeen et al.[9] is one such approach.

Optimizing Protection Strength with VS-ECC

VS-ECC combines the use of ECC and memory tests. The architecture exploits the observation that although a few cache lines exhibit multi-bit failures, the vast majority of cache lines contain one or fewer failures. Unlike prior solutions

that use fixed-strength mechanisms to mitigate the impact of cache failures on Vmin, we propose mechanisms that only allocate strong protection to cache lines that need such protection. Figure 10 plots the probability of having one, two, three, and four errors in a 64-byte cache line. Note that the probability of a single bit error exceeds that of a double bit error by about two orders of magnitude. This implies that, in a typical cache, single bit errors should be about one hundred times more common than double bit errors. It follows, therefore, that while many cache lines might require SECDED protection, only a very small subset of the failing cache lines require stronger multi-bit ECC. This suggests a variable-strength technique, VS-ECC, where the check bit budget is allocated judiciously only to the lines that require multi-bit protection.

"…the check bit budget is allocated judiciously only to the lines that require multi-bit protection."

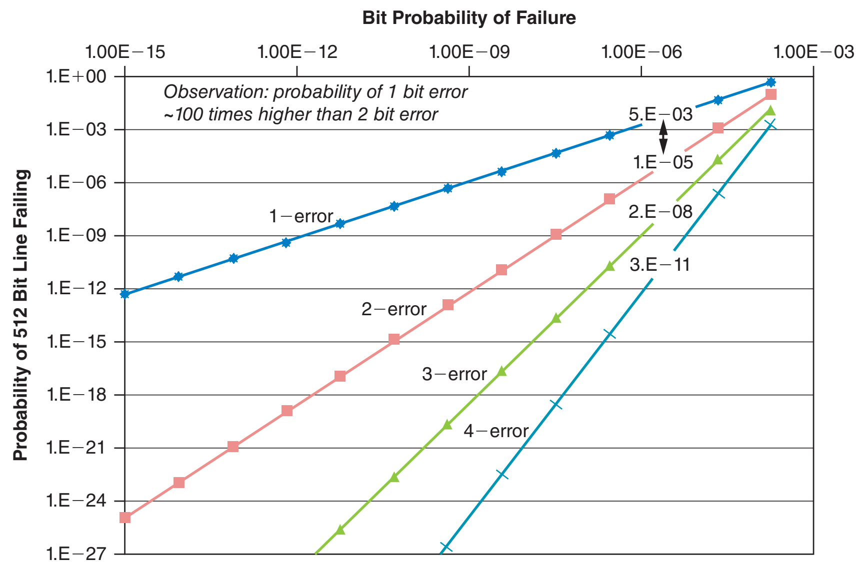


Figure 10: Multi-bit errors are rare relative to single bit errors.
(Source: Intel Corporation, 2013)

To implement a VS-ECC architecture, we begin with a cache that supports SECDED protection for each cache line, then we augment each set with four extended ECC (eECC) fields; these enable the cache to use a strong ECC code for any four of the 16 ways in each cache set. To distinguish lines that use multi-bit correction from lines that use SECDED ECC, we add an extra status bit to each tag, called "Extended ECC bit" or E-bit. If a cache line is classified as having multi-bit failures, the E-bit is set to 1; otherwise it is reset to 0. Each cache access first reads the E-bits with the tag to determine the type of ECC (SECDED or 5EC6ED) protecting the line. Depending on the strength of the code, we forward the cache data to one of two blocks of ECC processing logic: a simple logic block designed for SECDED, or a more complex block designed for multi-bit ECC processing.

Figure 11 shows the efficacy of VS-ECC on a 32-MB cache. It depicts seven different configurations, including three baseline configurations: "32MB," a 32-MB cache with no ECC protection, "SECDED," the same cache with

a SECDED code for each cache line, and finally "5EC6ED," a 32-MB cache with a 5-bit error correcting code protecting each cache line. The curves labeled "VS-ECC-1," "VS-ECC-2," "VS-ECC-3," and "VS-ECC-4," depict four different VS-ECC implementations, with the ability to provide a 5ED6ED code for one, two, three, or four cache lines, respectively. It's worth noting that three of the VS-ECC implementations are virtually indistinguishable from our 5EC6ED baseline, illustrating our earlier point that we need very few 5EC6ED codes to approach the performance of the 5EC6ED baseline.

"…three of the VS-ECC implementations are virtually indistinguishable from our 5EC6ED baseline…"

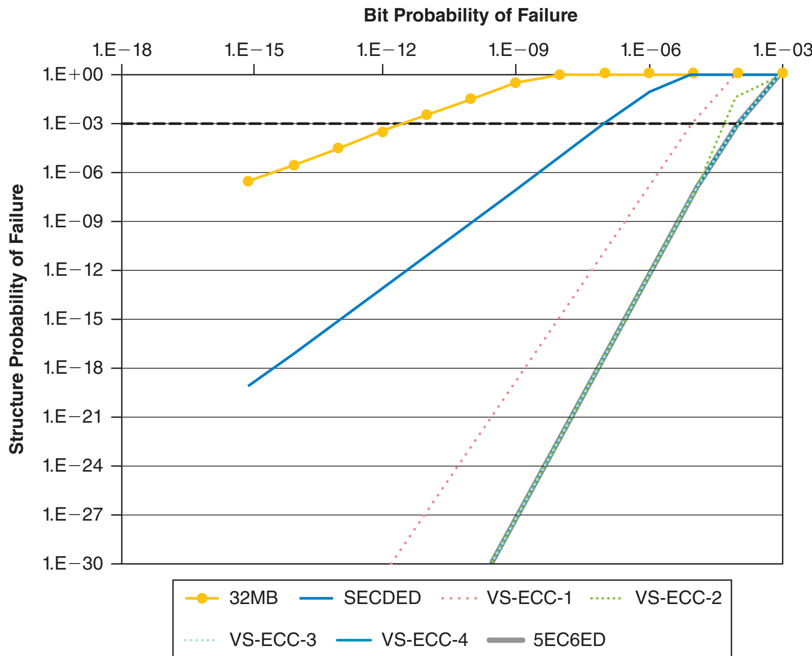


Figure 11: Efficacy of VS-ECC codes
(Source: Intel Corporation, 2013)

VS-ECC and Runtime Testing

The variable application of ECC protection in VS-ECC allows us to harvest the benefit of very strong ECC codes while minimizing overhead. Realizing this benefit, however, relies on prior knowledge of which lines require additional error correction. Since VS-ECC customizes the strength of the ECC depending on the presence or absence of bit errors, it requires tests to identify cache lines that contain bit errors. The strength of the VS-ECC approach, however, is that these VS-ECC tests can potentially be run while the system is active. Typically, memory tests must be run while the system is placed in marginal state, without voltage or timing guard-bands, to increase the likelihood of exposing bit errors. In conventional system designs, removing these guard-bands places any data stored in memory at risk. VS-ECC, however, can be used to provide stronger protection to the parts of the memory that contain live program data, while the rest of memory is tested. After each test phase, we move live data from the active to the inactive portion, activate the latter, and start testing the previously active region.

"…VS-ECC tests can potentially be run while the system is active."

Since VS-ECC allows the system to take advantage of spare cache cycles to test the cache, it can reduce the impact of testing on the user. This, in turn, opens up the possibility of using more complex, time-consuming tests. In addition, it facilitates intermittent testing that would help identify failures due to aging that may emerge after shipping a product.

"The primary benefit of VS-ECC is the additional flexibility it allows in testing…"

The primary benefit of VS-ECC is the additional flexibility it allows in testing, including runtime testing and the ability to compensate for testing failures through the universal application of minimal ECC. The primary drawback of the VS-ECC approach is that apportioning the ECC strength relies on the ability of tests to identify locations of bit errors. Although VS-ECC ultimately relies on testing, the flexibility it provides can make tests cheaper and less intrusive. We could improve on VS-ECC, however, by further minimizing its reliance on testing.

Error Correcting Codes as a Safety Net

One way to minimize a systems vulnerability to testing escapes is to construct a "safety net" for memory using ECC. If the overhead of the codes could be reduced sufficiently we could always rely on the ECC to identify and repair bit errors regardless of when they occurred. Hi-ECC[10] describes an approach to building such a system.

"…Hi-ECC takes the opposite approach of MS-ECC."

Hi-ECC relies on a low cost multi-bit error correcting code to identify bit errors. To reduce the storage costs of the code, Hi-ECC takes the opposite approach of MS-ECC. Instead of reducing logic costs through the use of large codes, Hi-ECC minimizes storage costs through the use of larger segments and dense but logically complex BCH codes. Typically, these tradeoffs would dramatically increase logic complexity; however, Hi-ECC avoids this by taking an approach similar to that of VS-ECC and optimizing for the common case.

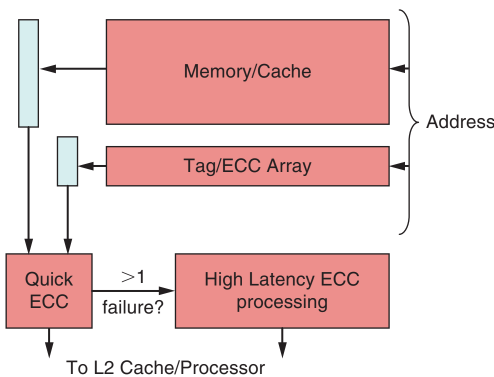


Figure 12: Quick ECC identifies error-free or easy-to-correct data and forwards it to the CPU; other data is forwarded to a high-latency ECC processing block
(Source: Chris Wilkerson, et al., 2010[10])

Recall that VS-ECC minimized overhead by providing two mechanisms to handle errors, a low-cost mechanism for the common case with few or no errors, and a high-cost mechanism for multi-bit errors. In the case of VS-ECC the two mechanisms were different strength codes, low-cost SECDED codes for the majority of the lines in the cache, high-cost 5EC6ED codes for lines that were identified as being prone to failure. Like VS-ECC, the Hi-ECC approach offers two mechanisms to handle errors. In contrast, however, the two mechanisms provided in Hi-ECC don't differ in the level of protection they offer but in the latency they incur and the resulting performance impact.

A key insight offered in Hi-ECC is that the ECC "check" (checking for bit errors) can be separated from the ECC "correct" (correcting the bit errors). The vast majority of the data is error free and only needs to be checked. Hi-ECC exploits this through the use of high-speed ECC checking logic (Quick ECC). Quick ECC (Figure 12) forwards lines that are either error-free or easily corrected to the processor with minimal additional latency. The remaining lines, which require additional processing, are sent to slow but relatively cheap ECC correcting logic.

Hi-ECC minimizes the cost of the ECC correction at the expense of increased latency. In fact, correcting a multi-bit error in Hi-ECC may take tens or hundreds of cycles. We argue that since bit errors are rare, the high latency of the correcting logic will have little impact on performance. To ensure this and to avoid pathologic cases such as repeated accesses to particularly error-prone lines, Hi-ECC removes bit errors as they are identified by ECC either through disabling the cache lines that contain them or removing the offending bits through techniques like bit-fix.

"Hi-ECC minimizes the cost of the ECC correction at the expense of increased latency."

Conclusion

In this article, we've highlighted some emerging challenges in memory resiliency. We've described five mechanisms to handle very high error rates. Three mechanisms minimized the performance overhead of correcting errors through adaptivity, incurring overhead when the performance cost of the overhead was minimized. The other two, VS-ECC and Hi-ECC, minimize overhead by providing two mechanisms to handle errors, a low-cost mechanism for the common case with few or no errors, and a high-cost mechanism for multi-bit errors. A*daptivity* and *optimizing for the common case* are both approaches that can help us design more resilient memory systems while minimizing the overhead.

"Adaptivity *and* optimizing for the common case *are both approaches that can help us design more resilient memory systems while minimizing the overhead."*

Although testing continues to play an important role in managing memory reliability, future technologies may introduce new failure mechanisms, which must be handled in new innovative ways. In light of this uncertainty, a memory resiliency architecture that scales with advancing technology must not rely on the physics of today's failures but must be flexible enough to adjust to the unanticipated failures of future technologies. The approaches taken in VS-ECC and Hi-ECC are noteworthy for the way they integrate multiple resiliency techniques. VS-ECC integrates testing and codes, while Hi-ECC supplements codes with additional repair. Future work in this area will extend this integration, combining error correcting codes, testing, and supplemental repair. Testing is likely to be a continuous process, combining information collected from ECC during use as well as from intermittent tests run throughout the lifetime of the system.

References

[1] A. J. Bhavnagarwala, X. Tang, and J. D. Meindl, "The Impact of intrinsic device fluctuations on CMOS SRAM Cell stability," IEEE Journal of Solid-state Circuits, Vol. 40, No. 9, pp. 1804–1814, September, 2005.

[2] S. Mukhopadhyay, H. Mahmoodi, and K. Roy, "Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 24, No. 12, pp. 1859–1880, December, 2005.

[3] Jaydeep Kulkarni and Kaushik Roy, "Ultra-low Voltage Process Variation Tolerant Schmitt Trigger based SRAM Design," IEEE Transactions on VLSI Systems, 2011.

[4] L. Chang et al., "An 8T-SRAM for Variability Tolerance and Low-Voltage Operation in High-Performance Caches," IEEE Journal of Solid-State Circuits, Vol. 43, No. 4, April, 2008.

[5] R. E. Aly, M. I. Faisal, and M. A. Bayoumi, "Novel 7T SRAM cell for low power cache design," IEEE SOC Conference, pp. 171–174, September, 2005.

[6] Chris Wilkerson, et al., "Trading off Cache Capacity for Reliability to Enable Low Voltage Operation," International Symposium on Computer Architecture, pp. 203–214, June 2008.

[7] M. Agostinelli et al., "Erratic fluctuations of SRAM cache Vmin at the 90nm process technology node," IEDM Technical Digest, pp. 655–658, Dec. 2005.

[8] Zeshan Chishti et al., "Improving Cache Lifetime Reliability at Ultra-low Voltages," Intl. Symp. on Microarchitecture, pp. 89–99, Dec. 2009.

[9] Alaa Alameldeen et al., "Energy-efficient cache design using variable-strength error-correcting codes," International Symposium on Computer Architecture, pp. 461–472, June 2011.

[10] Chris Wilkerson et al., "Reducing Cache Power with Low Cost, Multi-bit Error-Correcting Codes," International Symposium on Computer Architecture, pp. 83–93, June 2010.

Author Biograhies

Chris Wilkerson received the master's degree from Carnegie Mellon University, Pittsburgh, Pennsylvania, in 1996. He is currently a research scientist at Intel Labs, Hillsboro, Oregon. He has authored or coauthored a number of papers published on a number of microarchitectural topics including value prediction, branch prediction, cache organization, and runahead and advanced speculative execution. His current research interests include microarchitectural mechanisms to enable low-power operation for microprocessors.

Alaa R. Alameldeen received BSc and MSc degrees from Alexandria University, Egypt, in 1996 and 1999, respectively, and the MSc and PhD degrees from the University of Wisconsin-Madison, in 2000 and 2006, respectively, all in computer science. He is a research scientist at Intel Labs, Hillsboro, Oregon. His current research focuses on energy-efficient memory and cache design.

Zeshan Chishti received his BSc (Hons) degree in Electrical Engineering from the University of Engineering and Technology, Lahore, Pakistan, in 2001, and the PhD degree in Computer Engineering from Purdue University in 2007. He is a research scientist at Intel Labs, Hillsboro, Oregon. His current research interests include energy-efficient processors and memory systems and cache hierarchy design for chip multiprocessors.

IMPROVING MEMORY RELIABILITY, POWER AND PERFORMANCE USING MIXED-CELL DESIGNS

Contributors

Alaa R. Alameldeen
Intel Labs

Nam Sung Kim
University of Wisconsin-Madison

Samira M. Khan
Intel Labs and Carnegie Mellon University

Hamid Reza Ghasemi
University of Wisconsin-Madison

Chris Wilkerson
Intel Labs

Jaydeep Kulkarni
Intel Labs

Daniel A. Jiménez
Texas A&M University

Many enterprise and mobile systems attempt to maximize energy-efficient performance, dynamically trading off performance and power to have the best performance while keeping power within specified limits. Cache and memory system behavior plays a large role in this tradeoff, since power optimizations may jeopardize memory cell reliability.

In this article, we show that mixed-cell memory designs could play a key role in achieving the right balance between performance, power, and reliability for single-core and multi-core systems. In such designs, part of the memory structure is built with cells that are more robust and failure-resistant, while the rest is designed using traditional cells. Robust cells ensure resiliency under low-voltage conditions to protect the most vulnerable data, while the rest of the memory structure could be used to store redundant data to improve performance. We demonstrate this concept using two specific examples: (1) A cache system that only turns on the robust portion at low-voltage, achieving good reliability and power savings while providing high-voltage performance improvements; (2) A cache system that uses the whole cache (including the non-robust portion) at low voltage, achieving good performance and reliability while not exceeding power limits. While the specific examples we explore in this article are cache-related, the same concept could be used throughout the entire memory hierarchy to improve memory resiliency without sacrificing performance or energy efficiency.

Introduction

Power is a key design constraint for modern multiprocessors used across market segments, from mobile systems to servers. In mobile systems, thermal design power (TDP) plays a key role in determining the form factor of the mobile device, and therefore optimizing processor power is critical. Likewise, data centers are built with fixed power and cooling capabilities, and improving processor performance within a given power budget yields direct economic benefits by increasing the compute capability supported by a fixed investment in data center infrastructure.

"Increasing core counts, in the context of fixed power budgets, is a key challenge for future systems."

To address these power constraints, new processor generations have provided improvements in core performance and efficiency and have also increased the number of cores on a die. Today, state-of-the-art server processors may contain tens of cores, and even mobile products, including tablets and smart phones, have more than one core. Increasing core counts, in the context of fixed power budgets, is a key challenge for future systems.

"In today's TDP-limited systems, the voltage of active cores has to decrease as the number of active cores increases."

In today's TDP-limited systems, the voltage of active cores has to decrease as the number of active cores increases.[6] Conversely, as cores become inactive,

the voltage of the remaining cores is raised to maximize performance. Changing the voltage in response to changes in core activity allows the power budget of these systems to remain constant regardless of the number of active cores.

Voltage reduction, however, comes at the cost of dramatically reducing reliability for memory cells that operate at a low voltage. To circumvent this problem, prior work has explored using separate voltages for the core logic and caches. This captures most of the power benefits by reducing the core voltage, while ensuring reliable cache operation at a higher voltage. However, separate voltage domains greatly increase design complexity.[13] This added complexity can be avoided by building memories with robust cells better suited for low voltage operation (using larger cells with upsized transistors or more transistors). Unfortunately, robust cells significantly increase power and area for a memory structure.

"Voltage reduction, however, comes at the cost of dramatically reducing reliability for memory cells that operate at a low voltage."

The high overhead of cell upsizing has led architects to propose mixed (heterogeneous) cell cache architectures, consisting of traditional cells and robust cells[4][5][8], with the goal of minimizing the use of expensive, robust memory cells, while continuing to harvest their low voltage benefits. Mixed-cell cache architectures achieve this by implementing a small portion of the cache with robust cells that can operate reliably at low voltage, and the remainder with non-robust cells. When operating at a high voltage, both portions would be used to maximize cache capacity and performance. When operating at low voltage, the failure-prone non-robust cells would be turned off, reducing cache capacity by up to 75 percent.[4][5] Conversely, the non-robust cells can be turned on but are only used to store noncritical data.[8]

We advocate using mixed (heterogeneous) cell cache architectures to build reliable and scalable memory structures. Memory structures do not need to be uniformly reliable. With careful design mechanisms, a robust (reliable) portion can be used to store critical data, while the non-robust portion can be power-gated or used for noncritical data.

"We advocate using mixed (heterogeneous) cell cache architectures to build reliable and scalable memory structures."

In the remainder of this article, we demonstrate how mixed-cell architectures help achieve memory resiliency at low voltage. We highlight two examples for cache hierarchies designed with mixed cells:

- In the first design[5], a last-level cache is designed with a fraction of all cells built with robust cells, while the rest are built using standard cells that are power-gated at low voltage. Such a system helps maintain low-voltage cache reliability while allowing the whole cache to be active at high voltage/frequency to maximize performance.
- In the second design[8], both robust and non-robust cells are enabled at low voltage, but special logic needs to be implemented to ensure critical data (that is, the only copy in the system) is stored in robust cells. Such a design helps maximize performance for a multi-core system where all cores could be active only at low voltage.

Background

Achieving the highest possible density is a main design goal for different memory technologies. SRAM bit cells, for example, generally employ minimum-geometry transistors, which are susceptible to systematic as well as random process variations such as random dopant fluctuations (RDF) and line edge roughness (LER). Process variations produce V_T (threshold voltage) mismatch between neighboring transistors, resulting in asymmetric bit cell characteristics, and making bit cells susceptible to failure at low voltage. DRAM cells are also designed with minimum-sized transistors in a given process technology, making some cells less reliable when power-saving optimizations are used (such as lower refresh frequency). We'll use SRAM caches as the main topic of discussion in this article, but similar tradeoffs could also apply to other memory technologies.

"…large memory structures in the core, such as caches, become unreliable at low voltage."

With bit cells susceptible to failure, large memory structures in the core, such as caches, become unreliable at low voltage. This limits voltage and frequency scaling for the cores, which must operate at a minimum voltage (Vmin) to ensure reliable operation. Reducing cache Vmin has become an area of active research. Prior work in this area fits into two broad categories: circuit solutions and architectural solutions.

Circuit Solutions

Circuit techniques generally aim to reduce Vmin by improving the bit cell. One approach is to reduce the voltage for the core logic and use a separate (higher) voltage for caches. Unfortunately, a partitioned power supply increases power grid routing complexity, reduces on-die decoupling capacitance, increases susceptibility to voltage droops, and may require level shifters that add latency to signals that cross voltage domains. [13] Most commercial processors use multiple voltages generated off-chip by high-efficiency off-chip voltage regulators (~95-percent efficiency). As the number of cores increases, providing multiple voltages for each core becomes increasingly impractical. A four-core system with separate voltages for the core and its private L1/L2 caches would require three voltage domains per core (a total of 12 power supplies), in addition to those needed for other system components.

"…upsizing devices can dramatically reduce variations and improve Vmin."

Another way to improve bit cell Vmin involves upsizing its constituent devices. Threshold voltage (V_T) variation depends inversely on the transistor gate area.[9] Consequently, upsizing devices can dramatically reduce variations and improve Vmin. Zhou et al.[22] designed and optimized six different 6T SRAM cells (C1-C6 cells), and analyzed the failure probabilities of the cells due to process variations in a 32 nm technology. These analyses demonstrated that increasing a cell's size can reduce its failure probability by orders of magnitude.

Unfortunately, the Vmin benefits of upsizing a typical 6T bit cell diminish as device size increases. Figure 1 compares the Vmin for four different caches implemented in a 65 nm technology. Each cache is implemented using one

of four different 6T cells. [11] We set Vmin at the point when the cache failure probability is 1/1000.[20] The figure depicts the probability (y-axis) that the cache will contain a single failing bit as a function of voltage (x-axis). A 4-MB cache constructed with a minimum-sized 6T cell, 4M-min, exhibits very high failure rates (~30 percent) even at high voltages (>900 mV). The 4M-2X implementation of a 4-MB cache doubles the device sizes in each memory cell, increasing cell area by 33 percent. 4M-4X quadruples the size of the devices, doubling the size of the cell. The 4M-8X implementation uses the most robust cell with devices that are eight times as large and a 233 percent larger cell size than 4M-min. Increasing cell sizes initially yields dramatic improvements over minimum-sized cells (note the 275 mV improvement moving from 4M-min to 4M-2X). But further size increases yield smaller benefits, 60 mV and 55 mV for the 4M-4X and 4M-8X, respectively.

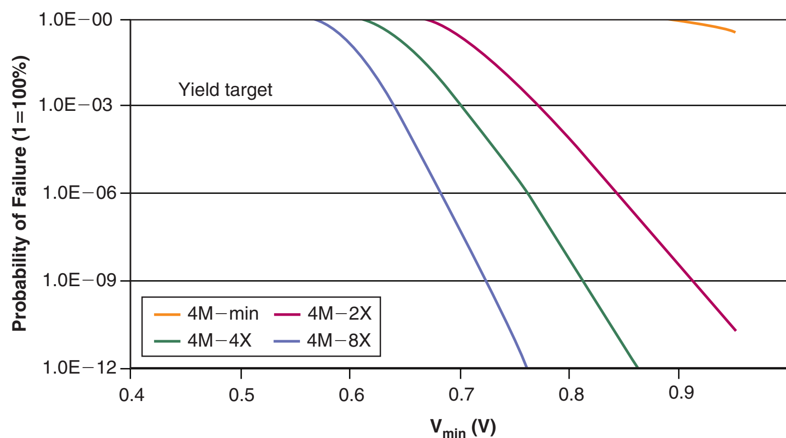


Figure 1: Vmin improvements with bit cell upsizing
(Source: Khan et al., 2013[8])

Cell upsizing also causes increases in static and dynamic power. Static power (leakage) varies linearly as a function of transistor dimensions, therefore increasing with larger cells. Larger cells also add switching capacitance on the word lines (WL) and bit lines (BL) increasing dynamic power. Upsizing from the minimum cell to the 2X cell yields a substantial benefit since the reduction in Vmin (275 mV) more than compensates for the additional power introduced by larger devices. Further upsizing, however, increases power since the costs of larger devices outweigh the savings from voltage reductions (–60 mV, –55 mV).[8]

"Cell upsizing also causes increases in static and dynamic power."

Architectural Solutions

Another approach to reducing Vmin uses failure-prone cells with smaller devices, but augments the memory array with the capability to repair bit failures. Prior work introduced many repair mechanisms that depend on memory tests to identify bad bits.[17][18][20] Relying on memory tests limits the applicability of these approaches when memory tests are expensive or failures are erratic.[1] Other repair mechanisms rely on coding techniques, such as

error-correcting codes (ECC), to autonomously identify and repair defective bits.[3][10]

Fundamentally, each of these approaches trades off the repair mechanism overhead for the ability to compensate for defective bits. For memory designs with very high failure rates, this tradeoff may be unattractive.

To address the high overhead of operating at low voltage, Wilkerson et al.[20] improve Vmin by storing error-correction patterns in cache resources, trading off cache capacity for low voltage operation. Chishti et al.[3] identify the limitations of testing-based implementations and propose to provide error correction capability using orthogonal Latin square codes. Chakraborty et al.[2] also trade off cache capacity for lower voltage. A multi-copy cache stores two copies of each clean datum and three copies of each dirty datum to allow detection and correction of corrupted bits, respectively.

"More recently, designs with mixed (heterogeneous) cell designs have been proposed to achieve low voltage with modest area cost."

More recently, designs with mixed (heterogeneous) cell designs have been proposed to achieve low voltage with modest area cost. Dreslinski et al.[4] propose to combine the low voltage benefits of robust upsized cells and the cost benefits of smaller cells by building caches with a mixture of cell types. Cache lines consisting of robust cells operate at low voltage, while a separate power supply provides a higher voltage to less robust cells. By moving recently accessed data to the low voltage cache lines, Dreslinksi et al. service the majority of requests using low voltage cache lines, and reduce active power in the L1 cache.

While using mixed cell architectures could help achieve reliable low voltage operation, it is important to ensure that such design has a minimal impact on high-voltage performance (for a single-core system) or low-voltage performance (for a multi-core system). In the next two sections, we highlight two mixed-cell cache architectures we explored in our prior work. The first architecture[5] is tailored towards high-performance systems, where high-voltage performance is critical but we need to maintain reliability at low voltage using robust cells. The second architecture[8] targets multi-core TDP-limited systems, where the highest performing point is when all cores are active at low voltage, so low-voltage performance is critical.

A Mixed-Cell Architecture for High-Performance Systems

A typical last-level cache (LLC) consists of hundreds or thousands of SRAM sub-arrays. We proposed an architecture for a single-core system that uses multiple cell sizes in a single LLC.[5] When high performance is needed, the processor runs at high voltage/frequency states where even small (non-robust) cells can operate reliably. As supply voltage is lowered, the failure rate of small cells increases exponentially, so we disable ways or sets one after another beginning with those consisting of the smallest SRAM cells. Ways or sets implemented with large cells remain active

(and reliable) at lower supply voltage, providing the needed LLC capacity. To avoid failures, a uniform implementation of the cache using only robust cells would significantly decrease cache capacity for the same area, therefore hurting high-voltage performance. Alternatively, we propose a heterogeneous-cell architecture to avoid low-voltage failures without hurting high-voltage performance.

"…we propose a heterogeneous-cell architecture to avoid low-voltage failures without hurting high-voltage performance."

LLC Implementation Using Heterogeneous Cell Sizes to Support Low Vmin

Consider a four-way set-associative cache. Figure 2 illustrates an example of building a four-way set-associative LLC, where each group of sub-arrays is associated with a cache way and has a different cell size. In this illustration, the total number of sub-arrays is divided into four groups where each group represents a particular way with a particular cell size; the sub-arrays with larger cells become taller since the cell size increases in the horizontal direction.[22] In this example, the processor and LLC are operating at 0.7 V. Thus, the LLC sections corresponding to ways three and four are disabled at 0.7 V since they are comprised of small cells, many of which will fail at such a voltage.

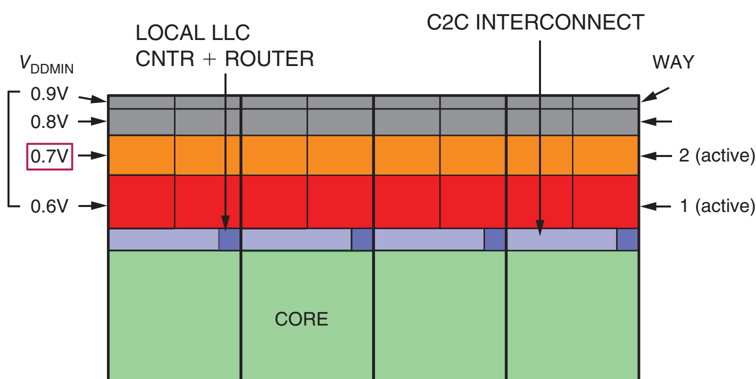


Figure 2: An example four-way LLC, where each way uses a different cell size and is active at a different voltage. The processor runs at 0.7 V, so only ways 1 and 2 are active
(Source: Ghasemi et al., 2011[5])

Consider an 8-MB LLC comprised of C5 and C3 cells[22] where each cell size provides 4-MB capacity. The cell failure probability of these cells is presented in Table 1. In this particular architecture, we can reduce the total cell area by 15 percent, that is, the total LLC area by 13 percent considering SRAM array efficiency equal to 85 percent.[26] When the voltage (frequency) is higher than 0.8 V (1.6 GHz), the processor is able to use the full 8-MB LLC capacity. If the voltage (frequency) is below 0.8 V (1.6 GHz), the 4-MB section of the LLC consisting of the smaller C3 cells will be disabled. However, the 4-MB section consisting of the larger C5 cells will operate reliably in the whole voltage range from 0.7 V to 0.9 V.

	C1	C2	C3	C4	C5	C6
Relative Cell Size	1.00	1.12	1.23	1.35	1.46	1.58
Pfail at 0.90 V	3.2×10^{-7}	2.5×10^{-9}	7.0×10^{-11}	4.5×10^{-12}	5.1×10^{-13}	1.2×10^{-13}
Pfail at 0.85 V	5.4×10^{-7}	1.0×10^{-8}	3.1×10^{-10}	1.6×10^{-11}	3.8×10^{-12}	1.0×10^{-12}
Pfail at 0.80 V	1.0×10^{-6}	3.0×10^{-8}	1.5×10^{-9}	7.6×10^{-11}	2.9×10^{-11}	9.0×10^{-12}
Pfail at 0.75V	2.0×10^{-6}	8.1×10^{-8}	7.4×10^{-9}	4.1×10^{-10}	2.2×10^{-10}	7.9×10^{-11}
Pfail at 0.70V	4.1×10^{-6}	2.1×10^{-7}	3.7×10^{-8}	2.2×10^{-9}	1.6×10^{-9}	7.0×10^{-10}

Table 1: Cell Size and Voltage vs. Probability of Cell Failure at Voltages from 0.9 V to 0.7 V (Source: Ghasemi et al., 2011[5])

To minimize the LLC area further, we can design a more heterogeneous LLC composed of C5, C4, C3, and C2 cells (Table 1) where each cell size gives 2 MB of capacity (for compactness of notation we refer to this as a 2-MB/2-MB /2-MB/2-MB C5/C4/C3/C2 LLC). As the voltage is decreased from 0.9 V to 0.8 V, to 0.75 V, and to 0.7 V, the LLC capacity is reduced from 8 MB to 6 MB, to 4 MB, and to 2 MB. In this architecture, the full 8-MB capacity operates reliably at 0.9 V. As the voltage decreases to 0.8 V, 0.75 V, and 0.7 V, each 2-MB section consisting of C2, C3, and C4 cells will, respectively, be disabled in turn. Within their range of valid operating voltages the resulting cache failure probability of each of the 6-MB, 4-MB, and 2-MB sections of the LLC is acceptable. Using this architecture, we can reduce the total area dedicated to SRAM cells by 18 percent, and therefore, the total LLC area by 16 percent if we assume 85-percent array efficiency. We also explored two other LLC architectures: (1) a 4-MB/2-MB/2-MB LLC consisting of C2/C3/C4 cells, and (2) a 2-MB/2-MB/4-MB LLC consisting of C1, C2, and C4 cells. These two additional LLC architectures satisfy the yield target for the given voltage range, 0.7–0.9 V as long as the proper section of the LLC is shut down for each voltage down-transition. Figure 3 shows the total LLC cell area and the operating voltage range of each section for four different LLC architectures relative to the baseline 8-MB one.

"Using this architecture, we can reduce the total area dedicated to SRAM cells by 18 percent…"

LLC Arch.	Capacity, V_{DDMIN} and relative area associated w/each cell type in LLC				Rel. tot. area
Baseline	C6:8M:0.7V				1.00
A	C5:2M:0.7V	C4:2M:0.75V	C3:2M:0.8V	C2:2M:0.9V	0.81
B	C5:4M:0.7V	C3:4:0.8V			0.85
C	C5:2M:0.7V	C4:2M:0.75V	C3:4:0.8V		0.83
D	C5:4M:0.7V	C3:2M:0.8V	C2:2M:0.9V		0.83

Figure 3: Total LLC cell area for different LLC configurations relative to the baseline. In each colored box whose area is proportional to the total cell area for a given cell size X:Y: Z represents cell size capacity and minimum operating voltage (Source: Ghasemi et al., 2011[5])

Microarchitecture Techniques for LLC Way Shutdown

In our example in Figure 2, as supply voltage decreases, one LLC way after another will be disabled in ascending order of cell size; a cell size is associated with an LLC way. When a voltage/frequency down-transition is triggered by dynamic voltage and frequency scaling (DVFS), an LLC way that cannot operate reliably at the new voltage is shut down. In such a case, the dirty LLC lines in the LLC way must be written back to main memory. The mechanism for shutting down a subset of LLC is already available in commercial multi-core processors to reduce leakage power consumption.[15]

Once the DVFS controller decides to decrease the operating voltage/frequency of the processor, each local LLC controller shown in Figure 2 examines each line in the way that is being shut down. If the line is dirty, it is either (a) written back to the memory controller queue (cache to memory or C2M) or (b) moved to another way after evicting a least recently used clean line in the same set (cache to cache or C2C). The next line is then examined after the status bit of the dirty line is set to the "invalid" state. This process is repeated until all lines are examined in the way that needs to be shut down. Note that a way shutdown process using option (a) may increase the traffic between on-chip cores and off-chip memory (and thus power consumption). On the other hand, the LLC can still service read/write requests to minimize the performance impact associated with the shutdown operations.

Performance and Power Impact

Mixed-cell LLC architectures may impact performance and power both positively and negatively. First, the leakage power remains significant due to the use of larger cells. However, our heterogeneous LLC architectures can reduce a substantial amount of the LLC leakage power since some LLC ways are automatically disabled at low voltage/frequency operating states. Second, the heterogeneous LLC architectures require significantly less die area for the same capacity (Figure 3) compared to a cache with all-robust cells. This freed-up die area can, in turn, be used to increase the LLC capacity, providing higher peak performance at the highest voltage/frequency state.

"…heterogeneous LLC architectures can reduce a substantial amount of the LLC leakage power…"

On the downside, two factors contribute to increasing memory traffic and higher power consumption. First, the flushing operations required before reducing voltage/frequency and disabling LLC ways increases memory traffic. Second, the reduced LLC capacity at low voltage causes more misses and therefore more memory traffic. These effects reduce overall performance and increase memory system power consumption. However, one should note first that workloads that need high performance would spend a substantial fraction of their runtime at the high voltage/frequency states. Furthermore, the interval of voltage/frequency changes is often longer than 10 milliseconds in a commercial operating system, mainly due to the performance penalty associated with PLL re-locking time (tens of microseconds) for changing

frequency.[16] This makes the overall performance impact of the flushing operations quite small; on average, the performance degradation is less than 0.5 percent even with the 1ms voltage/frequency change interval when combined with the C2C scheme.

Evaluation Summary

We evaluated our architecture using Simics[12] augmented with GEMS[14] running four commercial workloads and two memory-intensive SPEC[19] workloads. A more detailed analysis of our results is presented in.[5] Our proposed LLC architecture reduces the LLC total cell area by 15–20 percent without impacting high-voltage performance, compared to an all-robust LLC. Comparing the same cache area as an all-robust LLC, our architecture provides a higher cache capacity, leading to an average 15 percent higher peak performance. The performance impact of the proposed architecture is negligible when various voltage/frequency states are explored by DVFS as a function of changing performance and power demands. The proposed LLC architecture reduces their leakage power due to way disabling at low voltage. Overall energy consumption is reduced by 5–10 percent even though extra energy consumption is required to support the slightly longer runtimes and more frequent accesses to the LLC and off-chip memory.

"Our proposed LLC architecture reduces the LLC total cell area by 15–20 percent without impacting high-voltage performance…"

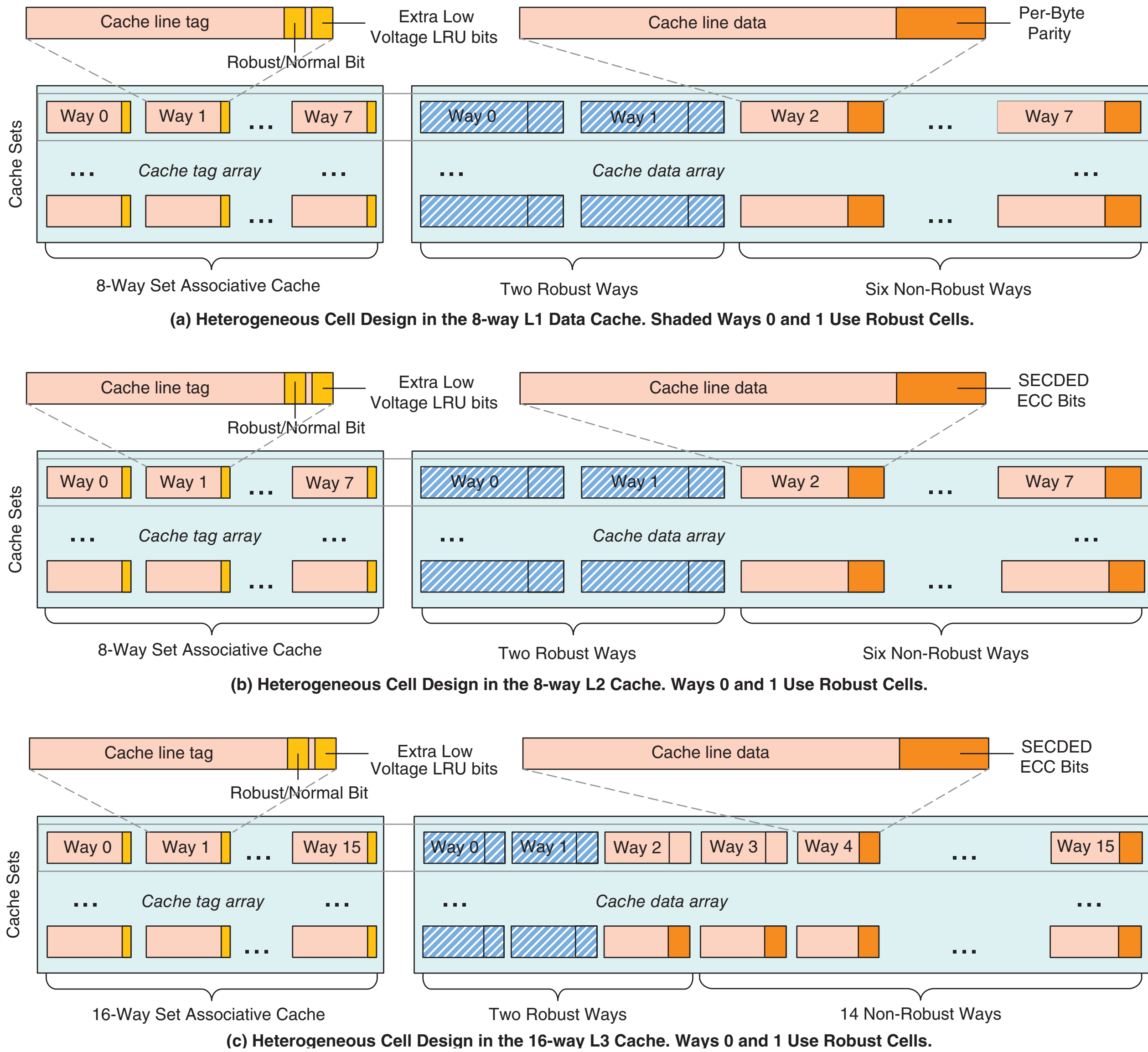
A Mixed-Cell Architecture for Multi-Core Systems

The motivation for our second mixed-cell cache architecture[8] is to enable the whole cache at low voltage, and therefore avoid a higher cache miss rate and improve low-voltage performance. This is needed for TDP-limited multi-core architectures where all cores can only be active at low voltage. To achieve this goal, we need to protect modified lines by storing them in robust cells, while using the remainder of the cache for clean lines. We use simple error detection and correction mechanisms to detect errors in clean lines, allocate write misses to robust lines, and read misses to clean lines. On a subsequent write to a clean line, we examined three alternatives to ensure modified data is not lost.

"…we need to protect modified lines by storing them in robust cells…"

Cache Hierarchy with Mixed-Cell Support

Figure 4 shows all three levels of our cache hierarchy with support for robust cells. Our baseline cache hierarchy uses a 32-KB 8-way L1 cache, 256-KB 8-way L2 cache, and a 4-MB 16-way LLC (L3). For each level in the cache hierarchy, we implement two ways with robust cells, while the remaining ways use standard (non-robust) cells. This adds an area overhead of 25 percent (L1 and L2) and 12.5 percent (L3) for the cache data array. We add a status bit associated with each tag indicating whether the associated line is a robust way or a non-robust way. We don't necessarily need this extra bit if the robust ways are fixed to two specific ways (Way 0 and Way 1 in Figure 4). We also add an extra LRU bit since we implement a different replacement algorithm in the low-voltage mode.



(a) Heterogeneous Cell Design in the 8-way L1 Data Cache. Shaded Ways 0 and 1 Use Robust Cells.

(b) Heterogeneous Cell Design in the 8-way L2 Cache. Ways 0 and 1 Use Robust Cells.

(c) Heterogeneous Cell Design in the 16-way L3 Cache. Ways 0 and 1 Use Robust Cells.

Figure 4: A Mixed-cell cache hierarchy: L1 cache uses parity, while the L2 and L3 use SECDED ECC (Source: Khan et al., 2013[8])

Each cache level has a different requirement for error detection and correction. Since the L1 cache is byte-accessible and extremely latency sensitive, we use a parity bit for each byte in the L1, similar to many Intel® Atom™ and Intel Core™ processors. We use simple SECDED ECC for each line in the L2 and L3 caches. We provide this protection for both robust and non-robust lines to account for soft errors as well as voltage-dependent failures. In general, detectable errors in clean data are recoverable from the next cache level or from memory. However, detectable errors in dirty lines may not be recoverable. To minimize detectable unrecoverable errors (DUEs), we handle modified data differently from unmodified data.

"To minimize detectable unrecoverable errors (DUEs), we handle modified data differently from unmodified data."

If an error is detected in a clean line, it is treated like a cache miss and is obtained from the next cache/memory level. For modified data, however, we must ensure a very low probability of failure, which we achieve through the use of robust cells. This is particularly true in the L1, where parity is unable to correct bit errors and the increased robustness of the cell allows us to minimize the likelihood of bit errors.

To simplify our L1 cache implementation, we handle all accesses to failing lines as cache misses. Since the number of such lines is small, this has little impact on performance. For the L2 and L3 caches, SECDED ECC corrects most errors. Errors that are detected but not corrected (for example, lines with two errors) are handled as cache misses and obtained from the next cache level or from memory. L2 and L3 lines that incur double-bit errors can be disabled to avoid undetectable errors, that is, silent data corruption (SDC), when soft errors hit the same line. Our analysis shows that the probability of failures in robust cells is extremely low at the voltages we consider. For example, we find that 99.9 percent of the L3 caches will suffer failures in less than 1 percent of all lines at low voltage.[8]

Our mixed-cell cache handles writes differently from reads. We need to satisfy the condition of storing modified data only in robust ways. To achieve this objective, we modify the cache replacement policy to handle write misses differently from read misses, and also need to handle subsequent writes to non-robust lines, as we explain in the next two subsections.

Changes to Cache Replacement Policy

We assume the baseline caches implement a least recently used (LRU) replacement policy to simplify our explanation, though the proposed mechanism could be applied to other replacement policies. In our mixed-cell cache architecture, we allocate write misses only to robust ways and read misses to non-robust ways.

"…we allocate write misses only to robust ways and read misses to non-robust ways."

The flowchart in Figure 5 demonstrates changes we made to the cache replacement policy. On a read miss, we choose a replacement victim, NR_LRU, only from non-robust ways based on LRU bits. On a write miss, we choose a victim, GLOBAL_LRU, which is the LRU line among all ways of the set (both robust and non-robust). If the victim line is robust, we trigger a writeback for modified data and allocate the new line in its place. If the chosen victim is in a non-robust way, we choose the LRU line from the two robust ways (RB_LRU), trigger a writeback for modified data to convert the RB_LRU line to a clean line, move the RB_LRU line to use the GLOBAL_LRU line's storage, and allocate the new line to the RB_LRU line.

An alternative implementation we investigated was to implement LRU for two disjoint groups of lines for each cache set: robust lines and non-robust lines. However, some benchmarks, where writes represent a significant fraction of all misses, suffered significant performance losses when write-allocates were limited to choose a victim only from robust ways. We still observed significant

performance losses even when the number of L3 robust ways increased from two to four or eight. This motivated our modified algorithm that chooses a GLOBAL_LRU victim on a write miss. While some benchmarks are affected due to limiting victim selection for read misses to only non-robust ways, the performance losses are small since each cache set has many more non-robust lines than robust lines (6 out of 8 for the L1 and L2 caches, and 14 out of 16 for the L3 cache).

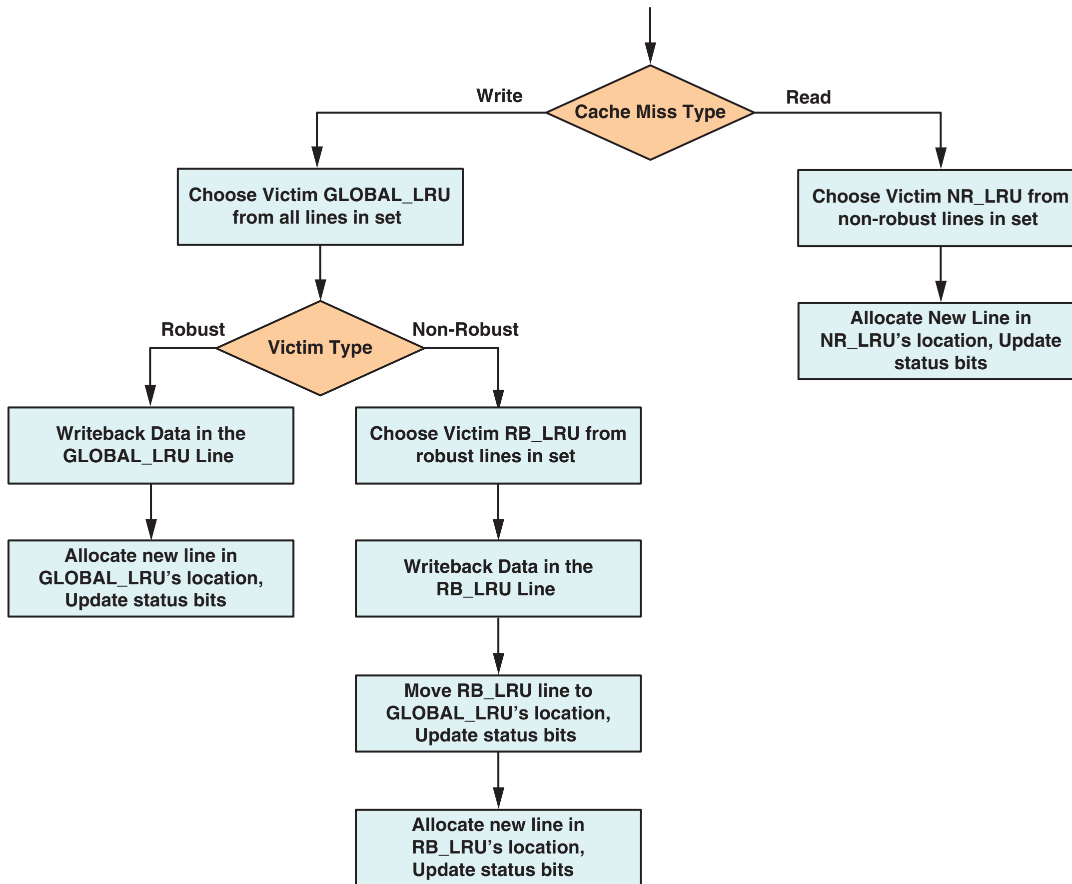


Figure 5: Changes to cache replacement policy
(Source: Alameldeen et al., 2013)

Handling Writes to Non-Robust Lines

Our mixed-cell cache architecture needs to prevent DUE and SDC for modified data. It is straightforward to implement this for lines allocated on a write miss, since the cache replacement algorithm would allocate them to robust cells. However, for lines that were allocated to non-robust ways on a read miss, we explore different alternatives to prevent failures.

"Our mixed-cell cache architecture needs to prevent DUE and SDC for modified data."

Writeback

We handle the write to a non-robust line like we would for a write-through cache. We store modified data in the same non-robust line, but convert it to a clean line by writing back the data immediately to the next cache level.

This writeback traffic causes significant network congestion, as well as power and latency overhead. A write to the L1 cache can trigger cascading writes all the way to memory if the L2 and L3 caches allocated the same line to non-robust ways.

Swap

We observe that a write to a cache line is often followed by more writes to the same cache line. To reduce writeback traffic, we handle a write to a non-robust line by swapping with the LRU way of robust lines in the set, RB_LRU. The RB_LRU line triggers a writeback to convert to a clean line. The RB_LRU line is then swapped with the written line. The status and LRU bits are also swapped between the two cache tags. This approach reduces traffic as it is more likely to write to the most recently written line than it is to write to the LRU robust line. We model this mechanism's overhead by blocking access to the cache for three cycles (L1) or six cycles (L2 and L3 that have 32-byte accesses) to account for using the cache read and write ports to perform the swap.

Duplication

To avoid writeback traffic and the additional swap latency, we explore trading off capacity to save this overhead. In this mechanism, we assign each two consecutive non-robust lines as "partner lines" similar to. [21] For example, in Figure 5's L1 cache, the line in way 2 is a partner line to that in way 3, the line in way 4 is a partner line to that in way 5, and the line in way 6 is a partner line to that of way 7. When a write occurs to a non-robust line, we evict its partner line and write the data to both lines, using two extra cycles. We modify the replacement algorithm so that the partner line is always invalid and not a candidate for replacement. This duplication causes losing some cache capacity, but avoids writeback traffic and swap overhead. When writing to a duplicate line, we perform the write to both the original line and its partner. When reading from a duplicate line, we check parity (L1) or ECC (L2/L3), and trigger a read from the partner line if an error is detected.

Evaluation Summary

We evaluated a power-constrained system with the ability to operate one, two, and four cores within the same power budget using CMP$im[7] and SPEC benchmarks.[19] A more detailed analysis of our results is presented in.[8] To support four active cores, our hypothetical system used a mixed-cell cache architecture to operate at 590 mV. In this mode, the 75 percent capacity loss experienced by our baseline mixed-cell cache architecture resulted in a 12 percent performance loss. Our proposal delivers a 9.5 percent performance benefit relative to a non-mixed cell baseline using only robust memories, which is similar to the performance improvement for our mechanism in the section "LLC Implementation Using Heterogeneous Cell Sizes to Support Low Vmin." However, our design avoids significant reductions in cache size at low voltage, improving multi-core performance by up to 17 percent on average and saving 50 percent of the L1 dynamic power compared to using only robust cells. While the writeback mechanism incurs significant overheads, both swap and duplication achieve significant performance improvements and power reductions.

"…our design avoids significant reductions in cache size at low voltage…"

"…improving multi-core performance by up to 17 percent on average and saving 50 percent of the L1 dynamic power…"

Conclusions

In this article, we showed that mixed-cell memory designs could play a key role in achieving the right balance between performance, power, and reliability for single-core and multi-core systems. For mixed-cell designs, part of the memory structure is built with cells that are more robust and failure-resistant, while the rest is designed using traditional cells. Robust cells ensure resiliency under low-voltage conditions to protect the most vulnerable data, while the rest of the memory structure could be used to store redundant data to improve performance.

We showed how this concept works using two specific examples. First, our heterogeneous LLC system only turns on the robust portion at low voltage, while using the whole cache at high voltage. This mechanism achieves significant power savings at low voltage and significant performance improvements at high voltage compared to a uniformly robust cache design. Second, our multi-core mixed cell architecture uses the whole cache (including the non-robust portion) at low voltage while ensuring modified data is not lost. This mechanism enables a multi-core system where all cores are active in a TDP-limited design and achieves significant performance improvements and power savings at low voltage. The same concept could be used throughout the entire memory hierarchy to improve memory resiliency without sacrificing performance or energy efficiency.

"This mechanism enables a multi-core system where all cores are active in a TDP-limited design…"

Acknowledgements

We are very grateful to Aamer Jaleel who helped us with his simulator, CMP$im. Work at the University of Wisconsin was supported by an NSF grant (CCF-1016262), a Fall Competition Multidisciplinary Research Award from the University of Wisconsin-Madison Graduate School, and NSF CAREER Award (CCF-0953603).

References

[1] M. Agostinelli, et al., "Erratic fluctuations of SRAM cache Vmin at the 90 nm process technology node," IEDM Technical Digest, pp. 655–658, Dec. 2005.

[2] Arup Chakraborty, et al., "E < MC²: Less Energy through Multi-Copy Cache," Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), pp. 237–246, Oct. 2010.

[3] Zeshan Chishti, et al., "Improving Cache Lifetime Reliability at Ultra-low Voltages," International Symposium on Microarchitecture, pp. 89–99, Dec. 2009.

[4] Ronald G. Dreslinski, et al., "Reconfigurable Energy Efficient Near Threshold Cache Architectures," International Symposium on Microarchitecture, pp. 459–470, Dec. 2008.

[5] Hamid Reza Ghasemi, Stark Draper, and Nam Sung Kim, "Low-Voltage On-Chip Cache Architecture using Heterogeneous Cell Sizes for Multi-Core Processors," International Symposium on High-Performance Computer Architecture, pp. 38–49, Feb. 2011.

[6] Intel Corporation, "Intel® Turbo Boost Technology in Intel® Core™ Microarchitecture (Nehalem) Based Processors," White Paper, 2008.

[7] Aamer Jaleel, et al., "CMP$im: A Pin-Based On-The-Fly Multi-Core Cache Simulator," 4th Workshop on Modeling, Benchmarking and Simulation, Beijing, China, June 2008.

[8] Samira M. Khan, et al., "Improving Multi-Core Performance Using Mixed-Cell Cache Architecture," International Symposium on High-Performance Computer Architecture (HPCA), February 2013.

[9] Muhammad M. Khellah, et al., "Read and Write Circuit Assist Techniques for Improving Vccmin of Dense 6T SRAM Cell," Conf. on Int. Circuit Design and Technology, pp. 185–189, June 2008.

[10] Jangwoo Kim, et al., "Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding," International Symposium on Microarchitecture, pp. 197–209, Dec. 2007.

[11] Jaydeep Kulkarni and Kaushik Roy, "Ultra-low Voltage Process Variation Tolerant Schmitt Trigger based SRAM Design," IEEE Transactions on VLSI Systems, 2011.

[12] P. Magnusson et al., "Simics: A full system simulation platform," IEEE Computer, 35(2): 50–58, Feb. 2002.

[13] Wai-Kei Mak and Jr-Wei Chen, "Voltage Island Generation under Performance Requirement for SoC Designs," Asia and South Pacific Design Automation Conference, Jan. 2007.

[14] M. Martin et al., "Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset," Computer Architecture News, 33(4): 92–99, Sep. 2005.

[15] A. Naveh et al., "Power and thermal management in the Intel® Core Duo Processor," Intel Technology Journal, 10(2): 109–122, May 2006.

[16] Jaehyun Park, et al., "Accurate Modeling and Calculation of Delay and Energy Overheads of Dynamic Voltage Scaling in Modern High-Performance Microprocessors," International Symposium on Low-Power Electronics and Design (ISLPED), pp. 419–424, Aug. 2010.

[17] David Roberts, Nam Sung Kim and Trevor Mudge, “On-Chip Cache Device Scaling Limits and Effective Fault Repair Techniques in Future Nanoscale Technology,” Digital System Design Architectures, Methods and Tools, pp. 570–578, Aug. 2007.

[18] Stanley Schuster, “Multiple Word/Bit Line Redundancy for Semiconductor Memories,” IEEE Journal of Solid-State Circuits, vol. 13, no. 5, pp. 698–703, Oct. 1978.

[19] SPEC CPU2006 Benchmarks, http://www.spec.org/cpu2006/

[20] Chris Wilkerson, et al., “Trading off Cache Capacity for Reliability to Enable Low Voltage Operation,” International Symposium on Computer Architecture, pp. 203–214, June 2008.

[21] Wei Zhang, et al., “ICR: In-Cache Replication for Enhancing Data Cache Reliability,” International Conference on Dependable Systems and Networks, pp. 291–300, June 2003.

[22] S.-T. Zhou et al., “Minimizing total area of low-voltage SRAM arrays through joint optimization of cell size, redundancy, and ECC,” Proc. IEEE International Symposium on Computer Design, pp. 112–117, Oct. 2010.

Author Biographies

Alaa R. Alameldeen received BSc and MSc degrees from Alexandria University, Egypt, in 1996 and 1999, respectively, and MSc and PhD degrees from the University of Wisconsin–Madison, in 2000 and 2006, respectively, all in computer science. He is a research scientist at Intel Labs, Hillsboro, Oregon. His current research focuses on energy-efficient memory and cache design.

Nam Sung Kim is an assistant professor at the University of Wisconsin–Madison. He has been conducting interdisciplinary research that cuts across device, circuit, and architecture for power-efficient computing. His research has been supported by National Science Foundation (NSF), Semiconductor Research Corporation (SRC), Defense Advanced Research Project Agency (DARPA), AMD, IBM, Samsung, Microsoft, and Korean Ministry of Education, Science, and Technology. Prior to joining the University of Wisconsin–Madison, he was a senior research scientist at Intel from 2004 to 2008, where he conducted research in power-efficient digital circuits and processor architecture. He also has served several prominent international conferences as a technical program committee member. Nam Sung Kim has been the recipient of IEEE Design Automation Conference (DAC) Student Design Contest Award in 2001, Intel Fellowship in 2002, and IEEE International Conference on Microarchitecture (MICRO) Best Paper Award in 2003, NSF CAREER Award in 2010, and IBM Faculty Award in 2011 and 2012. His current research interest is designing robust, low-power computing systems in nanoscale technology. He is an IEEE senior member

and holds a PhD in Computer Science and Engineering from the University of Michigan–Ann Arbor, and both an MS and a BS in Electrical Engineering from Korea Advanced Institute of Science and Technology.

Samira M. Khan is a post-doctoral researcher associated with Intel Labs and Carnegie Mellon University. She received her PhD from the University of Texas at San Antonio. Her research focuses on new microarchitectural designs that can make future microprocessors fast and energy efficient. She received her BSc degree from Bangladesh University of Engineering and Technology.

Hamid Reza Ghasemi is a PhD candidate at the Computer Sciences Department of University of Wisconsin–Madison. He received a BS and MS degree from the Electrical and Computer Engineering Department of University of Tehran. His research interests include computer architecture, power efficient high-performance processing, and low power architecture.

Chris Wilkerson received a master's degree from Carnegie Mellon University in 1996. He is currently a research scientist at Intel Labs, Hillsboro, Oregon. He has authored or coauthored a number of papers published on microarchitectural topics including value prediction, branch prediction, cache organization, and runahead and advanced speculative execution. His current research interests include microarchitectural mechanisms to enable low-power operation for microprocessors.

Jaydeep Kulkarni received a BE degree from the University of Pune, India, in 2002, an M. Tech. degree from the Indian Institute of Science (IISc), Bangalore, India, in 2004, and a PhD from Purdue University, in 2009, all in electrical engineering. During 2004–2005, he was with Cypress Semiconductors, Bangalore, India, where he was involved with low-power SRAM design. He is currently with the Circuit Research Lab, Intel Corporation, Hillsboro, Oregon, working on embedded memories, low-voltage circuits, and power management circuits. Dr. Kulkarni was the recipient of the Best M. Tech Student Award from IISc Bangalore in 2004, two SRC Inventor Recognition Awards, the 2008 ISLPED Design Contest Award, the 2008 Intel Foundation PhD Fellowship Award, and the Best Paper in Session Award at 2008 SRC TECHCON.

Daniel A. Jiménez is an Associate Professor in the Department of Computer Science and Engineering at Texas A&M University. Previously he was a full Professor and Chair of the Department of Computer Science at The University of Texas at San Antonio and Associate Professor in the Department of Computer Science at Rutgers University. His research focuses on microarchitecture and low-level compiler optimizations. He introduced and developed the perceptron branch predictor which has inspired the design of two implemented microarchitectures: the AMD "Bobcat" core and the Oracle SPARC T4. Daniel earned his BS (1992) and MS (1994) in Computer Science at The University of Texas at San Antonio and his PhD (2002) in Computer Sciences at The University of Texas at Austin. From

2002 through 2007, Daniel was an Assistant Professor in the Department of Computer Science at Rutgers. In 2005 Daniel took sabbatical leave at the Technical University of Catalonia (UPC) in Barcelona, Catalonia, Spain. In 2008 he was promoted to Associate Professor with tenure at Rutgers. He returned to his native Texas to take a position at UT San Antonio. He recently returned from a second sabbatical leave in Spain at the Barcelona Supercomputing Center and was General Chair of the 2011 IEEE HPCA conference. He is an NSF CAREER award recipient and ACM Distinguished Scientist.

STTRAM Scaling and Retention Failure

Contributors

Helia Naeimi
Intel Labs, Intel Corporation

Charles Augustine
Intel Labs, Intel Corporation

Arijit Raychowdhury
Georgia Institute of Technology

Shih-Lien Lu
Intel Labs, Intel Corporation

James Tschanz
Intel Labs, Intel Corporation

"This article will show that the main limitation on STTRAM dimensional scaling will be posed by retention time failure."

Ever larger on-die memory arrays for future processors in CMOS logic technology drive the need for dense and scalable embedded memory alternatives beyond SRAM and eDRAM. Recent advances in nonvolatile spin transfer torque (STT) RAM technology, which stores data by the spin orientation of a soft ferromagnetic material and shows current induced switching, have created interest for its use as embedded memory. Any attractive memory technology would be a viable solution if it could scale well for a few generations. We study the STTRAM scaling roadmap for last level cache (LLC) and how much dimensional scaling is feasible with this technology. This article will show that the main limitation on STTRAM dimensional scaling will be posed by retention time failure. When an STTRAM cell is scaled, the thermal stability factor (D) scales down linearly with the area, and causes unreliability due to retention failure. Today manufacturing techniques can fabricate nonvolatile STTRAM cells (with D ≥ 60 kT). Researchers are actively working at two fronts to pave the STTRAM scaling path for a few generations: 1) Novel manufacturing techniques that can facilitate fabrication of non-volatile STTRAM cells (with D ≥ 60 kT), 2) Architecture solutions that can relax the non-volatility condition and drop the required lower bound of 60 kT.

In this article, we focus on the solutions in the second category, that is, relaxing the nonvolatility condition to allow lower bound on D. Although there have been an extensive number of publications on dramatically relaxing the D, we believe these solutions alone can lower the bound on the thermal stability down one more generation before they become too costly. Beyond one more generation scaling, the dimensional scaling would depend on new manufacturing techniques to fabricate STTRAM cells with high thermal stability at scaled dimensions.

Introduction

As the number of cores in a chip continues to increase linearly, the demanded on-die memory capacity is expected to grow linearly as well. The limited power envelope of the future embedded and general-purpose systems makes emerging memory technologies, like spin transfer torque (STT) RAM technology, with zero-static power very attractive solutions in the cache hierarchy. STTRAM stores data by the spin orientation of a soft ferromagnetic material and shows current induced switching. When the spin-polarized current passes through a mono-domain ferromagnet, the ferromagnet absorbs some of the angular momentum of the electrons. It creates a torque that causes a flip in the direction of magnetization in the ferromagnet. This is used in magnetic tunneling junction (MTJ) based STTRAM cells where a thin insulator (MgO) is sandwiched between a fixed ferromagnetic layer (polarizer) and the free

layer (storage node). Depending on the direction of the current flow, the magnetization of the free layer is switched to a parallel (P: low resistance state) or antiparallel (AP: high resistance state) state.

STTRAM has the potential to replace the conventional on-die memory because of its zero standby power, high density, competitive read performance, and CMOS compatibility. Alongside the above attractive features, a viable new technology has to demonstrate a clear path to move to smaller technology nodes as the underlying CMOS technology scales down. STTRAM technology scaling is challenging with the nonvolatility condition. As the STTRAM cell scales down, assuming a volumetric dependence of thermal stability, the thermal stability factor (Δ) scales down linearly with the area. Thermal stability factor is the core feature of STTRAM cell. It identifies how much stability the MTJ has against thermal noise, which directly impacts the data retention time. The higher the thermal stability factor is, the longer the data is retained in the cell. A nonvolatile STTRAM cell (retaining data 10+ years) requires $\Delta \geq 60$ kT. Although cells with $\Delta \geq 60$ kT are fabricated today, fabricating nonvolatile STTRAM cells with practical write performance is not guaranteed at deeply scaled dimensions. The scaling of STTRAM cells every generation with $\Delta \geq 60$ and practical write performance requires novel manufacturing approaches or materials.

Using STTRAM in the last level cache (LLC) brings up an interesting opportunity. The data in LLC does not require 10+ years of retention time; hence the condition on high Δ can be relaxed. The purpose of this article is to provide a realistic assessment of relaxing the thermal stability in STTRAM cells in LLC. The notion of scaling Δ to support dimensional scaling and also lower WRITE current has been published quite extensively[1][2][3][4][5][6] and seems to provide a scaling path beyond what technology can provide. Our aim in this article is to understand the realistic limits of Δ, and how many generations of scaling will such change in the nonvolatility condition provide. To formulate this problem and allow the readers to follow our method of evaluation, we provide a detailed description of our numerical and analytical simulators, which have been built bottom-up. In conclusion we will show that, contrary to what has been previously published, with a true assessment with realistic process parameter variations, relaxing nonvolatility condition by the means of architectural solutions do assist dimensional scaling (and the scaling of Δ), but the advantage is limited.

"Using STTRAM in the last level cache (LLC) brings up an interesting opportunity. The data in LLC does not require 10+ years of retention time; hence the condition on high Δ can be relaxed."

"Our aim in this article is to understand the realistic limits of Δ, and how many generations of scaling will such change in the nonvolatility condition provide."

In the rest of this article, we start by reviewing STTRAM through coupled transport and LLG solvers in the section "Basics of STTRAM with Magnetic Tunneling Junction (MTJ)." This section will continue with read, write, and retention failure of STTRAM. Then we review our study of retention failure as the technology scales well as architectural solutions to facilitate scaling Δ in the section "Retention Failure and Scaling." Then we complete our retention failure study by including device variations and the challenges of retention time testing in the section "Process Variation and Retention Time Testing," followed by an article summary.

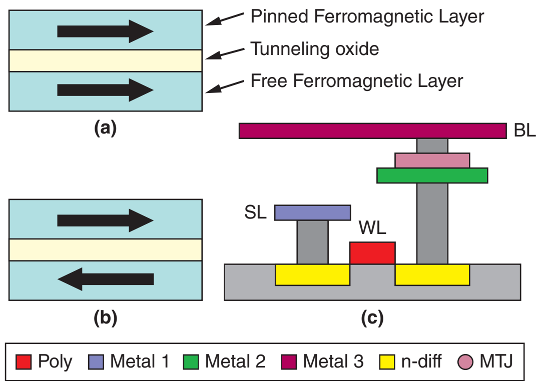


Figure 1: An MTJ stack with (a) parallel spin configuration (b) antiparallel spin configuration and (c) integration with the CMOS access transistor in the metal stack, showing Bit Line (BL), Source Line (SL) and Word Line (WL)
(Source: Intel Corporation, 2013)

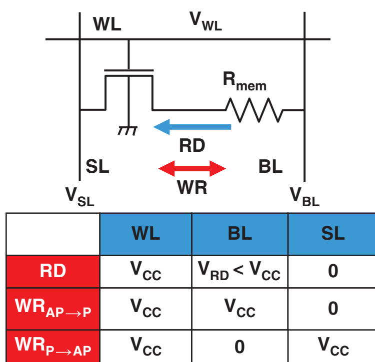
Basics of STTRAM with Magnetic Tunneling Junction (MTJ)

Here we provide a detailed description of the bitcell, its READ and WRITE circuits, and the principle technology/material parameters that need to be considered. The background material will also provide someone unfamiliar with the operation of an STTRAM cell a basic understanding of the cell behavior and the rigorousness required to perform a reasonable assessment of the bitcell.

It is a known fact that when a spin-polarized current passes through a mono-domain ferromagnet, it attempts to polarize the current in its preferred direction of magnetic moment. As the ferromagnet absorbs some of the angular momentum of the electrons (spin transfer effect or STT), it creates a torque that causes a flip in the direction of magnetization in the ferromagnet. The application of spin transfer effect in a memory device was enabled with a structure termed magnetic tunneling junction (MTJ). MTJ consists of two ferromagnetic layers separated by a tunneling barrier (such as Al_2O_3 or MgO) as shown in Figure 1(a) and Figure 1(b). The first ferromagnetic layer is fixed in magnetization and acts as a polarizer and the second layer orientation can be altered with the help of spin transfer torque and is referred to as a free layer. The resistance difference in this structure is termed tunneling magneto resistance ($TMR = [R_{AP}-R_P]/R_P$) due to the tunneling phenomenon. Switching of Al_2O_3-based MTJ was experimentally shown in 2004 and it was replaced by MgO-based MTJ to support better readability with TMR as high as ~200 percent at room temperature.[19] TMR improvement in MgO-based MTJs can be attributed to coherent spin-polarized tunneling.

MTJ stacks can be integrated with an access transistor to realize a bitcell structure 1T-1MTJ as shown in Figure 1(c). Figure 2 illustrates the 1T-1MTJ memory bitcell schematic and the voltage polarities for read (RD) and write (WR). The bitcell is read by precharging the BL to V_{RD} and allowing it to decay through the cell. A reference BL, which is simultaneously drained using a reference cell, acts as the sense amplifier reference. Both the reference and the accessed BLs are clamped using a PMOS current source, so that a constant differential is maintained at the sense amplifier input even for very long access times. On the other hand a bidirectional writing scheme is used for writing into the bitcell. For writing "1", BL is charged to V_{WR} and SL is connected to V_{SS}. For opposite direction switching, SL is biased at V_{WR} and BL at V_{SS}. Figure 3 shows the characteristics of MTJ during read and write operations. Write voltages for P-to-AP ($V_{WR}^{P\rightarrow AP}$) and AP-to-P ($V_{WR}^{AP\rightarrow P}$) switching and read voltage (V_{RD}) are indicated in the diagram.

To simulate transport and magnetization dynamics in multilayer magnetic tunnel junctions (MTJs) we have developed a self-consistent simulation framework based on non-equilibrium Green's function (NEGF) formalism and Landau-Lifshitz-Gilbert (LLG).[25][26] NEGF can numerically estimate the spin current flowing through the MTJ structure and LLG can



	WL	BL	SL
RD	V_{CC}	$V_{RD} < V_{CC}$	0
$WR_{AP\rightarrow P}$	V_{CC}	V_{CC}	0
$WR_{P\rightarrow AP}$	V_{CC}	0	V_{CC}

Figure 2: Circuit schematic and table showing RD and WR directions. For WR, the VCC will be limited by the VMAX of the technology node
(Source: Intel Corporation, 2013)

comprehend the dynamics of the free layer ferromagnet under spin current/spin torque. NEGF and LLG are coupled together for a self-consistent simulation framework[28][31] and are experimentally calibrated with material parameters reported in [27].

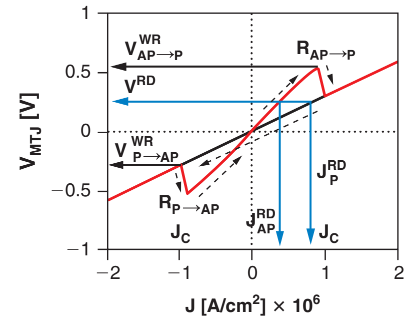


Figure 3: V-J characteristics of the MTJ. For symmetric JC, the preferred RD direction is identical to the WR direction for AP→P transition. RD is performed at a constant voltage VRD
(Source: Intel Corporation, 2013)

STTRAM Failure Analysis

Due to aggressive scaling of STTRAM cell structures, process variations and thermal disturbances can negatively impact the memory performance. The process variation and thermal disturbance causes three main failure modes: 1) Read failure, 2) Write failure, and 3) Retention failure. In our analysis of these failure modes we considered the following variation parameters.

In this analysis we have considered variations both in the access transistor and in the MTJ. Variations in the MTJ can be due to five major parameters, which are:

(a) systematic lithographic variation of critical dimensions of feature size F with $\sigma = 10$ percent

(b) normally distributed localized fluctuation of magnetic anisotropy, $K (K = M_S H_k)$[22]

(c) thermally activated initial angle of precession $[P(\theta) \approx 2(K/k_B T) \exp(-K\sin^2\theta/k_B T)]$

(d) thermal component of internal energy with $\sigma \sim k_B T$

(e) variation in MgO thickness

For the access transistor we have considered variations in threshold voltage V_{th}. Using the response surface technique[30], we formulate read (RD) and write (WR) failure probabilities, as well as retention failure rate for the bitcell and for a memory array.

Read Failure (RD Failure)

During reading bitcell value is identified using the reference current I_{REF}. When $I_{RD} > I_{REF}$, the bitcell state is P and when $I_{RD} < I_{REF}$, the state is AP. Read failures in a bitcell can be either due to distinguishability of states (P and AP) or due to read-disturb. In the first failure mechanism the states are sense limited where I_{RD}[AP state] $> I_{REF}$ or I_{RD}[P-state] $< I_{REF}$, or they can be time limited where not enough bit-line (BL) differential has been developed in the given read cycle time TRD. In the second failure mechanism (read-disturb), we accidentally write into the cell while reading the cell due to larger read current $I_{RD} > I_C$ ($I_C = J_C A$). Since WR is bidirectional and RD is single directional, only one type RD disturb (either P-to-AP or AP-to-P) can be present during bitcell reading.

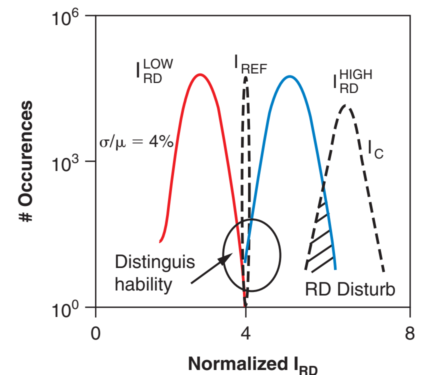


Figure 4: PDF of RD current for a default Monte-Carlo run (TMR = 100 percent; RA = 100Ω – um²) showing the failure modes
(Source: Intel Corporation, 2013)

Figure 4 illustrates the probability density function (PDF) of I_{RD} variation from a typical Monte-Carlo run considering variations described above. The figure

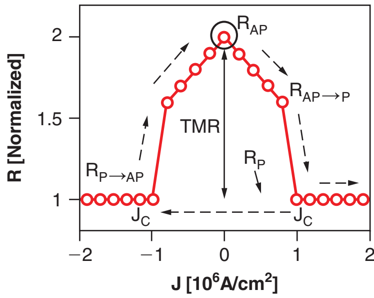


Figure 5: Normalized R_{MTJ} as a function of injected current density ($J_C = 1.1 \times 10^6$ A/cm^2)
(Source: Intel Corporation, 2013)

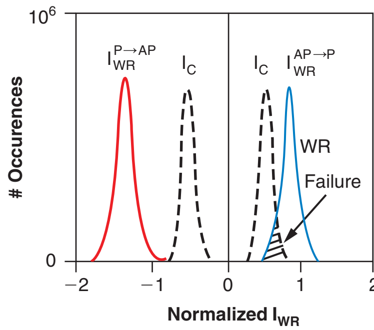


Figure 6: PDF of WR current for a default Monte-Carlo run ($J_C = 10^6$A/cm^2; $R_{AP}A = 50\Omega - um^2$) showing the WR failure modes
(Source: Intel Corporation, 2013)

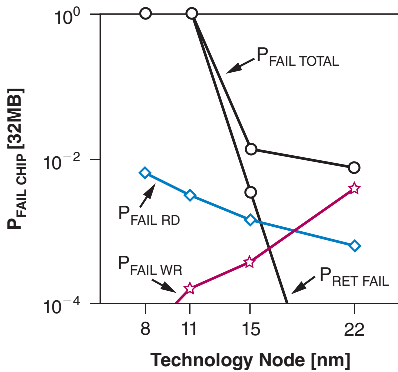


Figure 7: The trends of the type of failure
(Source: Intel Corporation, 2013)

shows the different modes of RD failure, namely distinguishability of states and RD disturb ($I_{RD}(P) > I_C$).

Write Failure (WR Failure)

Figure 5 shows the switching of MTJ under both positive (AP→P switching) and negative currents (P→AP switching) and associated resistance changes. Write failure in a bitcell is the inability to write a specific value into cells. It can be either due to higher J_C or due to lower access transistor current for a target write time TWR. As a result, either $I_{WRAP\to P} < I_{CAP\to P}$ or $I_{WRP\to AP} < I_{CP\to AP}$ or both can happen. Figure 6 shows the I_{WR} variation in a typical Monte-Carlo run and WR failure is present when $I_{WRAP\to P}$ is lower than $I_{CAP\to P}$.

Retention Failure

Retention failure occurs when the content of an idle cell flips due to the thermal noise. The thermal activation model of STTRAM suggests that a bit flip has a Poisson distribution with time characteristic of τe^{Δ}. Therefore the probability that a bit flips *n* times in the unit of time *t* is:

$$P_{flip}(n) = \frac{\lambda^n e^{-\lambda}}{n!} \quad (1)$$

where , $\lambda = \frac{t}{\tau e^{\Delta}}$ and τ is 1ns. The probability that a cell fails during time t is the sum of the probability that it switches an odd number of time. Since the first switching probability (P_{flip} (1)) is the dominating factor, the accumulated probability of odd number of switching is very close to the accumulated probability of the total number of switching. So we approximate the retention failure probability with the probability of total number of flips. Following the Poisson distribution this probability is:

$$P_{ret\text{-}fail} \cong \Sigma_{n=1}^{\infty} P_{flip}(n) = 1 - \exp({}^{-t}/_{\exp(\Delta)}) \quad (2)$$

The retention failure rate is exponentially dependent on the thermal stability factor (Δ). As the STTRAM cell and the MTJ device scales down, assuming a volumetric dependence of Δ, the thermal stability factor Δ scales down linearly with the area, causing exponentially increase in the retention failure rate.

Figure 7 shows all the failure rates as the technology scales, for a 32-MB cache. You can see that the retention failure will be the dominating source of error as the technology scales. This graph shows that retention failure can stop STTRAM scaling, and it needs to be addressed before any other issue. In the following sections we review the effect of retention failure on scaling in more detail, and propose solutions.

Retention Failure and Scaling

The stochastic process of retention failure resembles the well-known failure behavior of soft errors induced by cosmic rays. Soft error mitigation has been studied for decades, and the outcome of these studies can very well be applied to STTRAM as well or at least has to be taken into consideration to find a solution for STTRAM retention failure. The stochastic nature of error occurrence both in the soft errors and in STTRAM retention failure can flip the bit content with no warning or detectable signal. Therefore to ensure the data integrity by first, detect whether an error has happened and second, correct it; we need to incorporate some information redundancy in the array. The most efficient way of including the redundant information is to use error correcting codes (ECC). Each cache line is encoded separately with a number of code bits. Assuming a cache line has 64 bytes (plus *c* bits of code), the failure probability of a line with an ECC correcting *e* errors is a binomial cumulative distribution:

“The stochastic process of retention failure resembles the well-known failure behavior of soft errors induced by cosmic rays.”

$$P_{\text{line-fail}} = 1 - \Sigma_{i=0}^{e} P_{\text{fail}}^{\ i}(1 - P_{\text{fail}})^{512+c-i} \quad (3)$$

This is the probability that more than *e* errors happen in a line and it either goes undetected, or it is corrected to an incorrect value. Both of these two cases fall under the Silent Data Corruption or SDC. As the name suggests these are the types of errors that corrupt the data without being detected. When designing a reliable system, an SDC budget is allocated to each part of the system. Each part of the system has to be designed in a way that stays below its SDC budget. The LLC array is allocated an SDC budget, which will be divided equally among all the cache lines. To ensure the reliability of an array, each cache line has to follow its SDC budget. Below we explain the expected reliability failure of an array and how it is measured.

The SDC is measured in the number of failures in 1 million hours, which is called FIT rate. The target SDC FIT rate of a system is picked based on the sensitivity of the application. The SDC FIT rate of a CPU in today’s supercomputer is expected to be around 10 (that is, 10 failures in 1 million hours).[33] Since most of the CPU area is occupied by the LLC, the LLC will be the main contributor to the CPU SDC FIT rate. So we assume the LLC SDC FIT rate to be close to 10 as well. To ensure this SDC FIT rate for the array, the cache line SDC has to be bounded by $10/N$, where N is the number of cache lines. Obviously the bound becomes tighter as the cache capacity grows. Figure 8 shows the SDC FIT rate bound of a 64-byte cache line in LLC, assuming linear cache capacity scaling. Any reliability measure for a cache line has to ensure that the SDC FIT rate of a line falls below this curve. Therefore the value of $P_{\text{fail-line}}$ from Equation 4 has to stay below the red curve of Figure 8 to guarantee the 10 SDC FIT rate for the whole array structure. In the next section we evaluate how ECC can guarantee this bound and what are the associated costs.

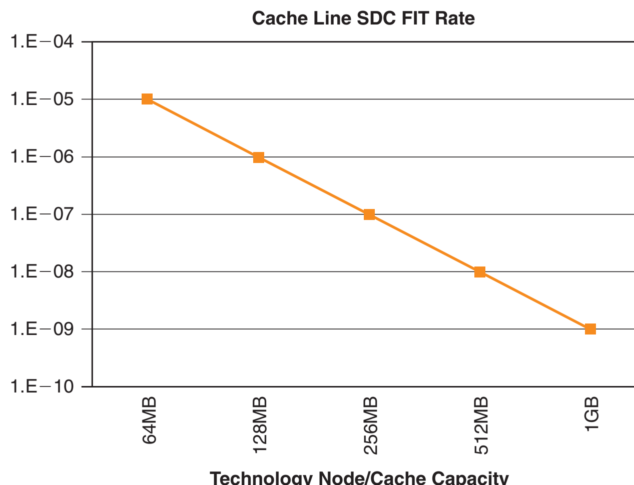


Figure 8: Maximum cache line SDC FIT rate with the technology and capacity scaling
(Source: Intel Corporation, 2013)

ECC and Scrubbing

The cost and complexity of error detection and correction circuitry of ECC increase rapidly for larger ECCs. The higher the bit failure rate is, the stronger code and more code bits are required. A large number of ECC bits take up array size and detecting and correcting strong ECCs with a large number of bits is a costly process from a performance and power point of view, which adds unwanted latency and power usage to access the array. In order to balance the number of ECC bits, one well-known technique is *scrubbing*.[8] During the scrubbing process, each cache line is read periodically and checked for errors. If there is an error, the line is corrected and written back. If the scrubbing process is done at the right frequency, it prevents the accumulation of error bits in a line; hence with the right scrubbing rate in place, we can achieve the target SDC rate with smaller ECC.

We did our analysis on BCH codes (Bose and Ray-Chaudhuri code) that are widely used in the memory architectures.[32] The detection and correction latency of BCH code grows with the error correcting capability of the codes. The access latency to an array is very critical, so we bound the ECC detecting latency to 1 clock cycle (1 ns). This bound includes ECCs with up to 5-error correcting capabilities.[10] So in this article we look at the codes up to 5-error correcting, which includes: SECDED (single error correction and double error detection), DECTED (double error correction and triple error detection), 3EC4ED, 4EC5ED, and 5EC6ED codes. For a 64-byte cache line, a code with e error correction and $e + 1$ error detection capability requires $10e + 1$ ECC bits.

Besides the ECC costs, there are also costs associated with scrubbing as well. Too frequent scrubbing consumes power and performance. During the scrubbing process, lines are read one by one. After each line is read it is checked using the ECC bits. If an error is detected then it is corrected and written back to the array. If no error is detected, then there is no need to rewrite the correct line. We use the timing model in [23] where the read access is 3 ns, and it is constant with the technology scaling. The ECC checking and correction takes one cycle, and write access takes 10 ns. Correction and writing back happens rarely, since an error occurs rarely. Therefore the scrubbing time is mainly reading and checking the lines. By using the read buffer and pipelining the reading and checking process, the checking latency is also hidden during scrubbing, and the main portion of the scrubbing is the time to read all the lines one by one.

We bound the scrubbing performance overhead by 5 percent. With 5 percent performance overhead limit and read latency of 3ns, we ran scrubbing experiment on SPEC CPU 2006 benchmark[34] that resulted the following scrubbing rates of 11.5 Hz, 5.8 Hz, 1.9 Hz, 0.7 Hz , and 0.25 Hz for 64-MB (at 32 nm) array, 128-MB (at 22 nm) array, 256-MB (at 15 nm) array, 512-MB (at 11 nm) array, and 1-GB (at 8 nm) array respectively. The last two scrub rates are extrapolated from the simulation data. Figures 9, 10, and 11 show the detail performance overhead of scrubbing for 64-Mb single core, 128-MB double core, and 256-MB quad core environment, respectively.

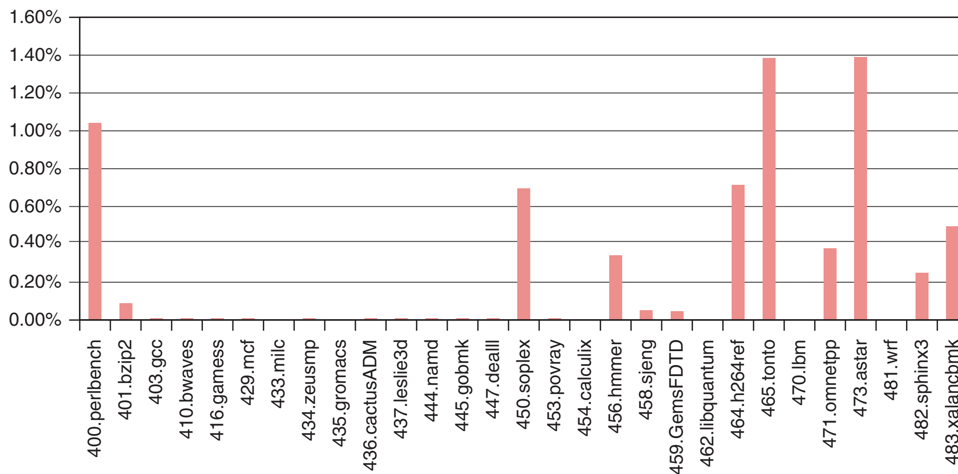


Figure 9: Scrubbing overheads of 64-MB cache with single core processor at 11.5 Hz (Source: Intel Corporation, 2013)

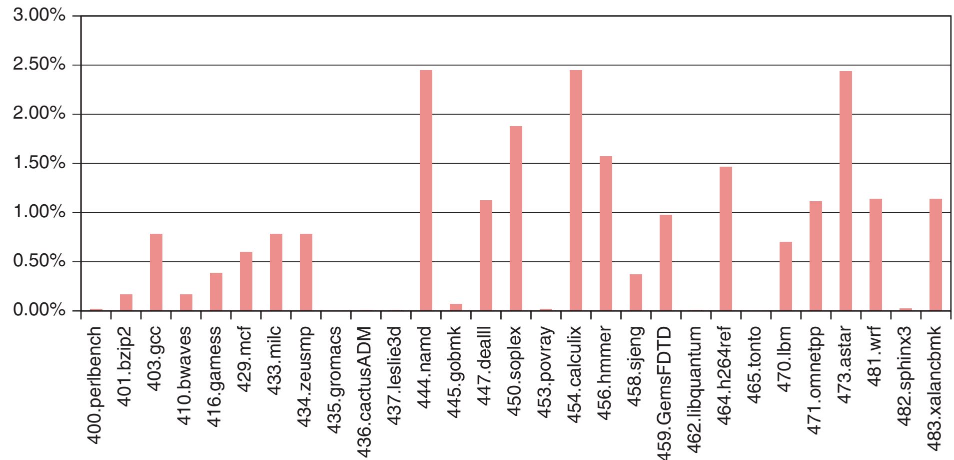


Figure 10: Scrubbing overheads of 128-MB cache with dual core processor at 5.8 Hz (Source: Intel Corporation, 2013)

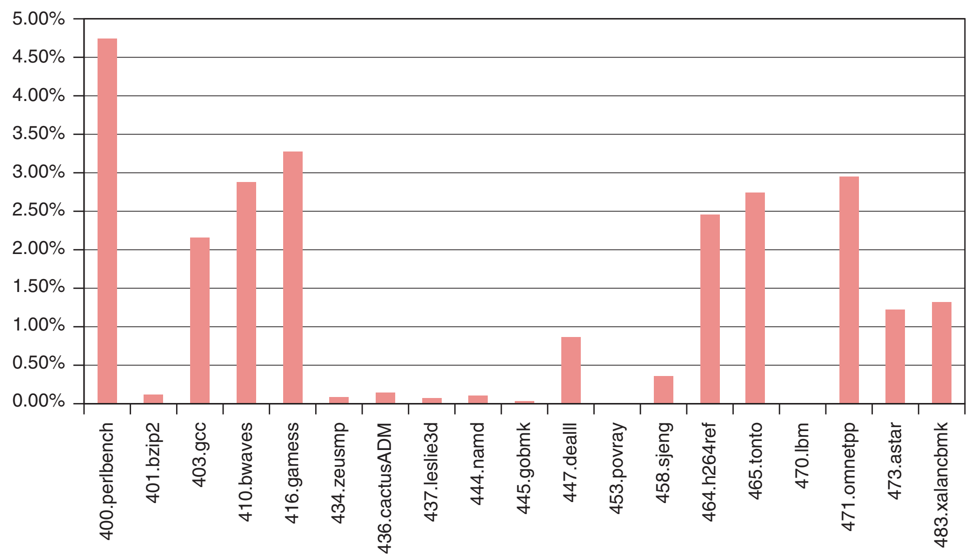


Figure 11: Scrubbing overheads of 256-MB cache with quad core processor at 1.9 Hz (Source: Intel Corporation, 2013)

Based on the derived scrubbing rates, the new lower bound on the value of Δ is presented in Table 1. The Δ value in Table 1 is generated in such a way that it follows the SDC rate of Figure 8.

Here we implemented a simple scrubbing algorithm. It simply blocks the cache with the scrubbing frequency for the period of time that takes to read all the cache lines. Of course, performing smarter scrubbing algorithm and parallelizing the process can potentially facilitates higher scrubbing rates and hence smaller lower bounds on Δ.

ECC type	64 MB	128 MB	256 MB	512 MB	1 GB
No ECC	60.0	60.0	60.0	60.0	60.0
SECDED	47.3	48.1	48.7	49.4	50.1
DECTED	39.8	40.5	41.2	41.9	42.5
3EC4ED	35.9	36.6	37.3	38.0	38.7
4EC5ED	33.6	34.2	35.0	35.6	36.3
5EC6ED	32.0	32.6	33.3	34.0	34.7

Table 1: Highest thermal stability scaling available with ECC, as the technology and capacity scales
(Source: Intel Corporation, 2013)

In the next section we evaluate the power overheads of ECC and scrubbing. We investigate the costs of adding ECC bits and scrubbing and the benefit of lower write power as the result of scaled thermal stability.

Power Analysis

Here we first review our power model, and then explain the scrubbing costs. The write power scales with the technology and the thermal stability.[23] The read power at the first order is independent of the thermal stability factor and stays constant in this analysis. We assume 3-to-1 read/write ratio for typical operation. We assume 64-MB cache at 32nm and linear capacity increase at every generation. We assume maximum LLC bandwidth usage in our analysis. The array structure and power model follow the model in [23].

The goal of this power analysis is to show how much power saving is gained by lower write power and how much power is consumed by ECC and scrubbing process. Figure 12 summarizes the power analysis. The power numbers for each array size and technology node are relative to the base case, which is at 32nm with Δ = 60 kT and no scrubbing. For each array size and technology node we evaluated 5 different ECC strengths. The lower bounds on the Δ of each of the ECCs and array sizes are available from Table 1. The relative power consumption trend for each ECC and technology node is illustrated with one bar in Figure 12.

The graph shows the scaled thermal stability reported in Table 1 that facilitate close to one more dimensional scaling can save up to 5% in power by using ECC and scrubbing.

“The graph shows the scaled thermal stability reported in Table 1 that facilitate close to one more dimensional scaling can save up to 5% in power by using ECC and scrubbing.”

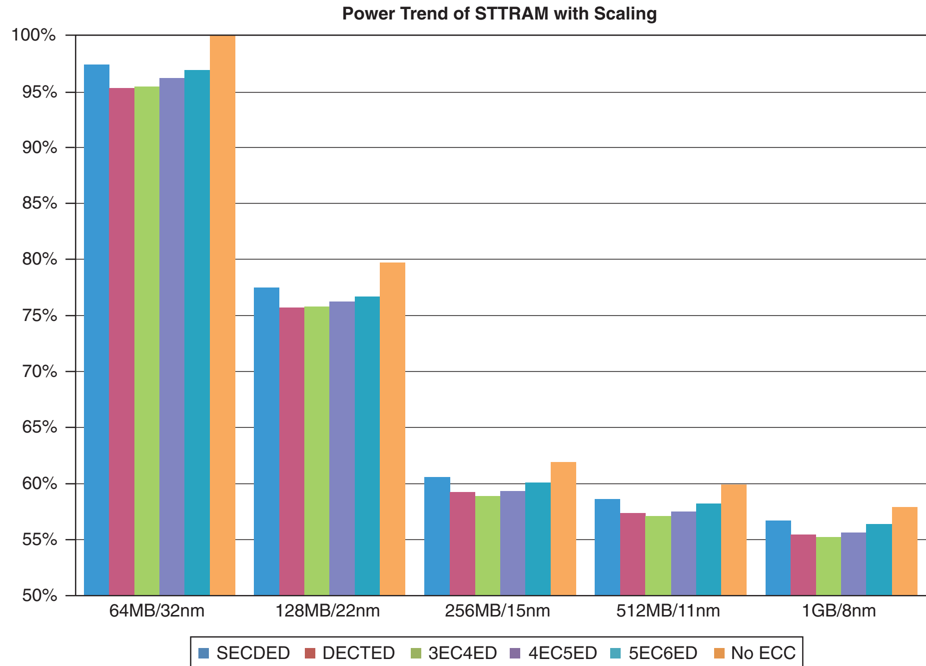


Figure 12: Relative power consumption of STTRAM array with 5 different ECCs and no ECC
(Source: Intel Corporation, 2013)

DRAM-Style Refresh vs. ECC and Scrubbing

The idea of scaling Δ to support lower power consumption has been widely applied. Then to address the increased retention failure due to the scaled Δ, there are number of publications proposing DRAM-style refresh.[1][2][6][5] DRAM-style refresh though popular and low cost, cannot provide any reliability for STTRAM technology. Below we explain the details, starting by reviewing the DRAM or eDRAM retention failure mechanism and how the refresh process can boost the DRAM or eDRAM retention time, but not the STTRAM cell retention time.

"…DRAM-style refresh though popular and low cost, cannot provide any reliability for STTRAM technology."

The data in DRAM cells is represented by the amount of charge on the DRAM cell capacitor. As soon as the data is written into the DRAM cell, the cell capacitor starts discharging gradually, and hence losing its state value slowly. Figure 13 shows the discharge process and the retention failure as the function of time for different cell capacitors. Note that the failure probability is a sharp function. When the capacitor charge gradually reaches the set threshold, the probability of a failure increases sharply (the figure shows the success probability). Hence any time before the charge drops to the set threshold the DRAM cell value can be read and the lost charge of the capacitor can be restored. Figure 14 shows how the refresh process restores the charge periodically and hence the retention failure is kept low by refreshing.

"In STTRAM the retention failure is a stochastic process. A bit flip happens due to the thermal noise and it happens almost instantly."

In STTRAM the retention failure is a stochastic process. A bit flip happens due to the thermal noise and it happens almost instantly. So there is no gradual

degradation process similar to DRAM capacitor. Reading and writing back like DRAM-style refresh will not improve reliability. It just reads corrupted bit values and rewrites corrupted values.

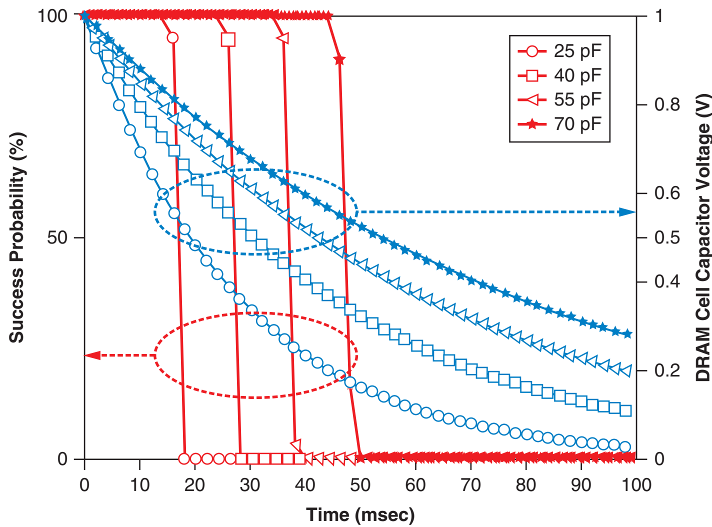


Figure 13: This graph shows the discharge process of DRAM cell capacitor and the retention failure probability of a DRAM cell. We assumed V_{th} is 40% of VDD and 10% random process variation (Source: Intel Corporation, 2013)

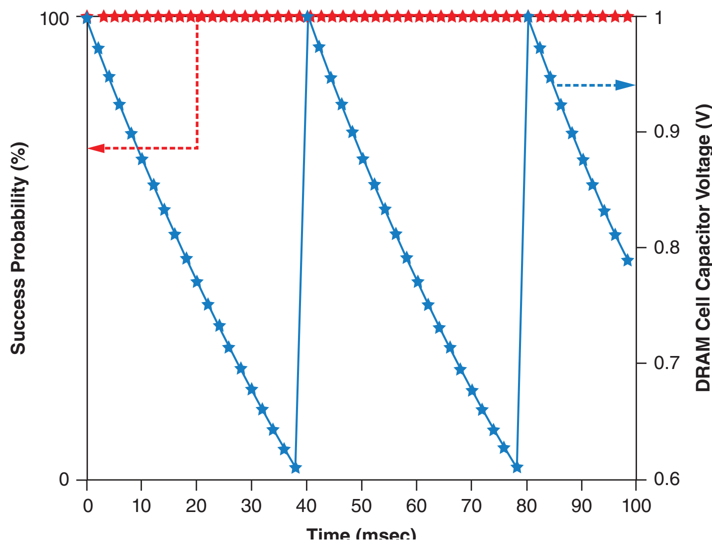


Figure 14: This graph shows the discharge process and the retention failure with refresh every 40ms. The cell capacitor is 40fF. (Source: Intel Corporation, 2013)

"Since there is no way to predict or sense whether the bit is going to fail in STTRAM, we need to add some way of redundancy to protect the information, like ECC…"

Since there is no way to predict or sense whether the bit is going to fail in STTRAM, we need to add some way of redundancy to protect the information, like ECC, and in order to prevent error accumulation in the codewords, every line has to be scrubbed frequently, as it was explained earlier in this section.

Process Variation and Retention Time Testing

The above analysis on thermal stability scaling is done with zero variation. Later in this section we evaluate the effect of scaling with process variation. Since the MTJ fabrication process is compatible with CMOS, we expect that STTRAM will have the same variation as the CMOS process. We make conservative assumption and assume the cell features have a normal distribution with a standard deviation of $\sigma = 10$ percent μ.[35]

Here we evaluate three commonly used approaches to address process variation. The most common approach is designing with guardband. In this approach the system is overdesigned or made to underperform to make the weak bits functional. The other commonly used approach is to detect bad bits and isolate them. The detect-and-isolate technique can use spare bits or ECC to replace the isolated bits. The last approach is to design for the average case, by adjusting the design values such that the average value matches the nominal value. Below we evaluate the cost and reliability of these three approaches for STTRAM.

"The last approach is to design for the average case, by adjusting the design values such that the average value matches the nominal value."

Designing with Guardband

Guardbanding is the most common solution to address variation. We start with a simple review of how it works. Assume a device parameter is required to be above a certain threshold t, and it has variation σ with normal distribution. If the nominal feature value is targeted for t, then 50 percent of the cells have lower than the threshold value, and are considered broken. However if the nominal feature is targeted for a value higher than t, for example $t + 3\sigma$, only 0.1 percent of the cells are bad and if it is targeted for $t + 6\sigma$, then only one cell in every one billion cells is bad. So guardbanding the feature value t with 6σ improves the BER (bit error rate) to 1 per billion. Even with this low BER, the chip yield will be very low (58 percent for 64MB array). For more practical chip yield, e.g. 99.99 percent, the nominal Δ should be 103.6 kT for 64 MB array. Using the guardbanding approach, the whole chip has to be tested to detect any bad bits with low Δ, and if a bad bit is detected the chip will be discarded. Detecting cells with low Δ is not an easy process (details in the section "Retention Time Testing").

Detect-and-Isolate

Unlike guardbanding which discards a chip with any detected bad bits, the detect-and-isolate approach can tolerate limited number of bad bits (about 0.1 percent [11]), and require small area and performance costs. Consider the

STTRAM example from the previous section, with a $\sigma = 10$ percent, and $\Delta \geq 60$ kT. Since detect-and-isolate can tolerate limited BER, in this case 0.1 percent, we have to guardband Δ by 4σ to get to the low BER of 0.1 percent. This requires nominal $\Delta = 80.4$ kT,

Similar to guardbanding, detect-and-isolate requires retention time testing to identify the bad bits.

Designing for the Average Case

Both of the previous approaches depend on retention time testing and since it is a costly process, here we look at another solution that does not require testing and detecting bits with low Δ.

It is important to note that the variation caused weak bits in MTJ fails differently than the weak bits in SRAM or capacitance-based cells. The capacitance-based weak cells tend to behave as permanently broken or intermittently broken bits. The weak MTJ cells with lower thermal stability follow the same stochastic bit flip process as the strong cells, except with higher probability, so even the weak bits work most of the time. With this understanding of the retention failure process, we can keep the weak bits and just add more ECC the same way that we added ECC earlier to address retention failure with no variation, in the section "ECC and Scrubbing."

We calculated the retention failure of an array *in the average*. Then we calculate the effective Δ based on this average value; that is, if there were no variation and all the cells have the same thermal stability factor, how much would have been the value of Δ, so that the array has the same failure rate as the array with the variation. First we calculate the average failure probability as:

$$P_{fail_avg} = \int_{0}^{+\infty} f(\Delta) \times P_{ret\text{-}fail}(\Delta) \quad (4)$$

Where $P_{ret\text{-}fail}(\Delta)$ is the retention failure of STTRAM cell with thermal stability factor Δ from Equation 3, and $f(\Delta)$ is the normal distribution. We used trapezoid numerical solution to solve the above equation.

Then we calculate the value of $\Delta_{effective}$ such that:

$$P_{fail\text{-}avg} = 1 - \exp(-t/\exp(\Delta_{effective})) \quad (5)$$

Table 2 summarizes the minimum nominal delta required for each of the above approaches with the presence of 10 percent variation. For example, using DECTED on a 64-MB array requires nominal Δ of 54.3 kT. We can see that the architectural solutions the ECC plus scrubbing can support scaling the thermal stability factor down to 44 for 1-GB array size. Depending on the application any one of the above solutions could be used by using the nominal thermal stability value from Table 2. Since the first two solutions depend on the retention time testing, for the sake of completeness we review the cost of performing retention time testing next.

Solutions		64 MB	128 MB	256 MB	512 MB	1 GB
Design for average	NO ECC	113.5	113.5	113.5	113.5	113.5
	SECDED	73.9	75.9	78.0	80.0	82.1
	DECTED	54.3	55.7	57.2	58.6	60.0
	3EC4ED	46.6	47.8	49.1	50.3	51.6
	4EC5ED	42.4	43.5	44.7	45.8	47.0
	5EC6ED	39.6	40.7	41.8	42.9	44.0
Guardbanding (99.99% yield)		103.6	104.2	104.7	105.3	105.9
Detect-and-isolate		80.4	80.4	80.4	80.4	80.4

Table 2: The thermals stability scaling available with the process variation (Source: Intel Corporation)

Retention Time Testing

MTJ characteristic is determined by two figures of merit, the thermal stability factor (Δ) and the intrinsic switching current (Jc_0). Retention time testing therefore boils down to determining the value of these two features. The key model to obtain these values is based on the thermal activation model:

$$P_{sw} = 1 - \exp\left(-t \Big/ \exp\left(\Delta \left(1 - \frac{I_c}{I_{c0}} \right) \right) \right) \tag{6}$$

This function and the functions derived from it are commonly used to fit experimental data in order to obtain the values of Ic_0 and Δ.[12][13][14][15] Since this model is a stochastic model, all the retention time test approaches require a large number of experimental data to obtain statistically significant result (for example, 1e5 to 1e7 experiments per data point[12]). Additionally, the simple thermal activation model is most accurate in the low switching current and long pulse width (100 ns[12][13][14][15][16]). The combined large number of experimental data with long pulse width makes the retention test time very long.

The retention test can be set up in various ways depending on how to use the thermal activation model. Here we look at one of such approaches from [12]. Other approaches are similar.

Starting from the thermal activation model and with

$$\frac{t_p}{\tau_0 \exp\left(\Delta \left(1 - \frac{Ic}{Ic_0} \right) \right)} << 1 \tag{7}$$

performing a Taylor expansion, results in the following expression:

$$\text{In}\left(P_{SW}\left(\frac{Ic}{Ic_0} \right) \right) = \text{In}\left(\frac{t_p}{\tau_0} \right) - \Delta\left(1 - \frac{Ic}{Ic_0} \right) \tag{8}$$

As we mentioned, the thermal activation model is most accurate for long pulse width and small switching current. Experimental data from [12] and [13] suggests switching pulse width of t_p = 100ns and switching current ratio of $Ic/Ic_0 \leq 0.80$ are reasonable values, which results in $P_{sw} \leq 1E{-}3$. For this small switching probability, in order to have a statistically significant result we need more than 1E+5 experimental data for each data point on the curve.[17] Testing the hypothesis of expected probability of 1E-3 with ±1 percent error margin and 99 percent confidence requires 5E+5 number of experiments.[17] We need multiple points on the curve to be able to fit the curve accurately. Assuming we need 10 points, if multiplied by 100ns pulse width, it takes about half a second to test one cell (we neglected the one-nanosecond read time after each 100-nanosecond write pulse). Assuming that the cells of a line can be tested in parallel, then testing a 64-MB array, line-by-line, takes more than five days. This is obviously not practical. We need to parallelize the test process with multiple built-in self-test (BIST) structures embedded in the array. We show below that parallelizing the test process can improve the test time to 16 minutes.

Test Algorithm and Structure

Figure 15 demonstrates the BIST structure to test a subarray. For simplicity it only shows one line of the subarray. The test pattern is first scanned in to the test flip-flips (FF). Then we follow the test process of Figure 16. First the test pattern is written to the line under the test. Then a weak current, Ic, is applied for t_p (that is, 100 ns). Then the value of the line is read. If there is a mismatch between the value of the FFs and the read value, then the error counter will increase. The error counter will help identify the switching probability. If the total number of failures during testing a line for an Ic/Ic_0 value is N_{fail}, dividing N_{fail} by the total number of experiments, N, is the estimated switching probability of that cell

$$P_{sw}\left(\frac{Ic}{Ic_0}\right) \approx \frac{N_{fail}}{N} \tag{9}$$

The value of the counters will be scanned out after N rounds of disturb and read. The inner loop rotates N times. Then this process is repeated for M times, the number of required points on the curve for curve-fitting. Once all the M values of Ic/Ic_0 are tested, the pairs of

$$\left(P_{sw}\left(\frac{Ic}{Ic_0}\right), Ic/Ic_0\right) \tag{10}$$

are used for curve fitting to Equation 8 and identifying Δ and Ic_0. This process will be repeated for each line in the subarray (and other subarrays that share the same tester). Assuming each subarray has one thousand lines, and every other subarray has one BIST. The total test time is:

$$\text{Number of Lines} \times N \times M \times t_p = 2000 \times 5E + 5 \times 10 \times 100_{ns} \approx 16_{min} \tag{11}$$

Although this is still very long test time, we cannot increase the parallelism due to the limit on the amount of current that the system can draw. In [12] the switching current is about 0.23mA. In a 64-MB array, writing 512 bits per line and testing 500 lines in parallel draws 60 A. This is about the upper bound of the current that the array can draw.

The retention time testing is mainly challenging due to its lengthy process. We should expect 16 minutes to test an array of size 64MB. Due to current limitations it is not possible to test more than 500 subarrays in parallel. Therefore the test time grows as the cache size grows.

Based on the above analysis, testing a 128-MB, 256-MB, 512-MB, and 1-GB array takes half an hour, one, two, and three hours, respectively. So any variation tolerance solutions that rely on retention time testing will suffer from very long test time.

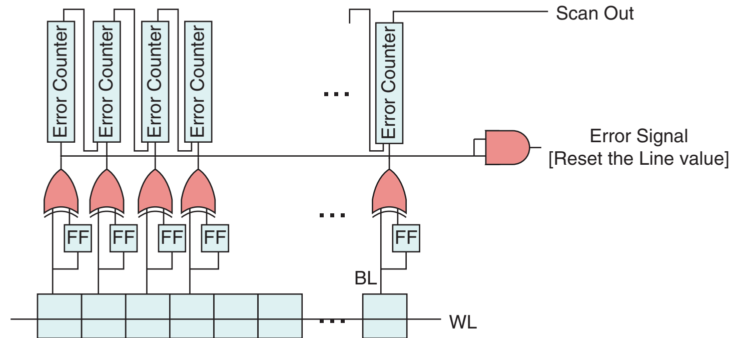


Figure 15: Test structure for retention time testing of an array (Source: Intel Corporation, 2013)

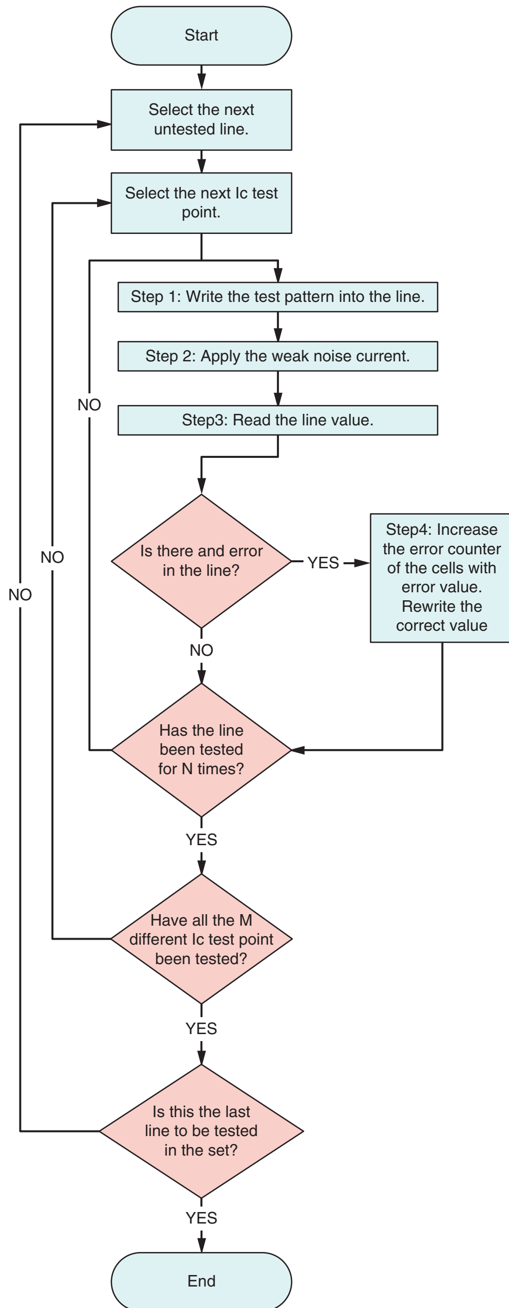


Figure 16: Retention time testing algorithm (Source: Intel Corporation, 2013)

Summary

STTRAM has many attractive features to be considered for replacing the capacitance-based cache or main memory technology. In order to have spin-based memory technology scalable to future technologies it has to sustain its reliability requirement. We showed that the dominating source of failure is the retention failure, and a key contributing factor is the thermal stability factor scaling. Assuming volumetric dependence for Δ, the thermal stability factor scales linearly with the technology nodes, sustaining high Δ despite the dimensional scaling demands, application of novel manufacturing solutions, or new material with higher coercivity at each generation. The alternative, as many have tried to show, is to use an architectural solution to relax the nonvolatility and the bound on the thermal stability. Unlike the claims of many previous publications, we showed that this alternative cannot facilitate continues linear scaling. It can accommodate only modest scaling. The result is summarized in Figure 17.

The green curve in Figure17 shows the required thermal stability for 10SDC FIT rate as the technology scales, with SECDED. This curve assumes the thermal stability for nonvolatile cells can be achieved at scaled dimensions. The blue line shows the scaling that can be supported with ECC plus scrubbing proposed in this article. The red curve shows the volumetric scaling of MTJ and Δ assuming consistent coercivity.

The gap between the red and the blue curve has to be closed by either system level reliability solutions or by manufacturing innovations and high coercivity material.

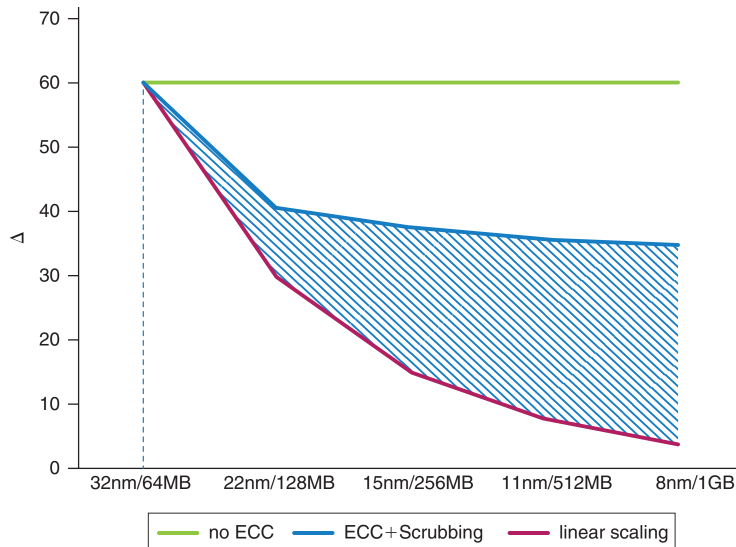


Figure 17: Thermal stability scaling trend with technology scaling (Source: Intel Corporation, 2013)

References

[1] A. Nigam, C. Smullen, V. Mohan, E. Chen, S. Gurumurthi, and M. R. Stan, "Delivering on the promise of universal memory for spin-transfer torque RAM (STT-RAM)," in *ISLPED*, Fukuoka, Japan, 2011.

[2] C. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *HPCA*, 2011.

[3] X. Cong, N. Dimin, Z. Xiaochun, S. h. Kang, M. Noak, and X. Yuan, "Device-architecture co-optimization of STT-RAM based memory for low power embedded systems," in *ICCAD*, 2011.

[4] L. Hai, W. Xiaobin, O. Zhong-Liang, W. Weng-Fai, Z. Yaojun, W. Peiyuan and C. Yiran, "Performance, Power, and Reliability Tradeoffs of STT-RAM Cell Subject to Architecture-Level Requirement," *IEEE Transactions on Magnetics,* Vol. 47, No. 10, pp. 2356–2359, 2011.

[5] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache revive: architecting volatile STT-RAM caches for enhanced performance in CMPs," in *DAC*, San Francisco, USA, 2012.

[6] Z. Sun, X. Bi, H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *Micro*, Porto Alegre, Brazil, 2011.

[7] X. Wang, Y. Zheng, H. Xi, and D. Dimitrov, "Thermal fluctuation effects on spin torque induced switching: Mean and Variations," *Journal of Applied Physics,* Vol. 103, No. 3, pp. 034507–034507–4, 2008.

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

[9] S. Lin and D. Costello, *Error Control Coding: Fundamentals and Applications*, Prentice Hall, 2004.

[10] D. Strukov, "The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories," in *ACSSC*, Asilomar, 2006.

[11] S. Schechter, G. H. Loh, K. Straus, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," in *ISCA*, Saint-Malo, France, 2010.

[12] R. Heindl, W. Rippard, S. E. Russek, M. R. Pufall, and A. B. Kos, "Validity of thermal activation model for spin-transfer torque switching in magnetic tunnel junctions," *Journal of Applied Physics,* Vol. 109, No. 7, pp. 073910–073910–5, 2011.

[13] A. Driskill-Smith, S. Watts, V. Nikitin, D. Apalkov, D. Druist, R. Kawakami, X. Tang, X. Luo, A. Ong and E. Chen, "Non-volatile spin-transfer torque RAM (STT-RAM): Data, analysis and design requirements for thermal stability," in *Symposium on VLSI Technology*, 2010.

[14] M. Pakala, Y. Huai, T. Valet, Y. Ding, and Z. Diao, "Ciritical Current distribution in spin-transfer-switched magnetic tunnel junctions," *Journal of Applied Physics,* Vol. 98, No. 5, 2005.

[15] T. Min, C. Qiang, R. Beach, G. Jan, H. Cheng, W. Kula, T. Torng, R. Tong, T. Zhong, D. Tang, W. Pokang, C. Mao-min, J. Z. Sun, J. K. Debrosse, D. C. Worledge, T. M. Maffitt and W. J. Gallagher, "A Study of Write Margin of Spin Torque Transfer Magnetic Random Access Memory Technology," *IEEE Transactions on Magnetics,* Vol. 46, No. 6, pp. 2322–2327, 2010.

[16] H. Zhao, A. Lyle, Y. Zhang, P. K. Amiri, G. Rowlands, Z. Zeng, J. Katine, H. Jiang, K. Galatsis, K. L. Wang, I. N. Krivorotov, and J.-P. Wang, "Low writing energy and sub nanosecond spin torque transfer switching of in-plane magnetic tunnel junction for spin torque transfer random access memory," Journal of Applied Physics, Vol. 109, No. 7, pp. 07C720–07C720–3, 2011.

[17] D. Montgomery and G. Runger, *Applied Statistics and Probability for Engineers*, John Wiley and Sons, 2010.

[18] Z. Diao, Z. Li, S. Wang, Y. Ding, A. Panchula, E. Chen, L. Wang, and Y. Huai, "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory," Journal of Physics: Condensed Matter, Vol. 19, No. 16, pp. 165209 (1–13), April, 2007.

[19] S. S. P. Parkin, C. Kaiser , A. Panchula , P. M. Rice , B. Hughes, M. Samant, and S.-H. Yang, "Giant tunneling magnetoresistance at room temperature with MgO (100) tunnel barriers," Nature Materials, Vol. 3, pp. 862–867, 2004.

[20] Z. Diao, A. Panchula, Y. Ding, M. Pakala, S. Wang, Z. Li, D. Apalkov, H. Nagai, A. DriskillSmith, L. Wang, E. Chen, and Y. Huai, "Spin transfer switching in dual MgO magnetic tunnel junctions," Appl. Phys. Lett., Vol. 90, No. 13, pp. 132508 (1–3), March, 2007.

[21] N. N. Mojumder, C. Augustine, D. E. Nikonov, and K. Roy, "Electronic Transport and Effect of Quantum Confinement in Dual Barrier Resonant Tunneling Spin-Torque-Transfer Magnetic Tunnel Junctions," J. Appl. Phys., Vol. 108, pp. 104306 (1–12), November, 2010.

[22] Y. Saito, H. Sugiyama, T. Inokuchi, and K. Inomata, "Interlayer exchange coupling dependence of thermal stability parameters in synthetic antiferromagnetic free layers," J. Appl. Phys., Vol. 99, pp. 08K702 (1–3), April, 2006.

[23] A. Raychowdhury, D. Somasekhar, T. Karnik, and V. De, "Design Space and Scalability Exploration of 1T-1STT MTJ Memory Arrays in the Presence of Variability and Disturbances," International Electron Device Meeting (IEDM) Tech. Dig., pp. 707–710, December, 2009.

[24] Y. Chen, Y. Wang, Y. Wang, C. Lin, US Patent 20090303779, 2009.

[25] S. Salahuddin, D. Datta, P. Srivastava, and S. Datta, "Quantum transport simulation of tunneling based spin torque transfer (STT) devices: Design tradeoffs and torque efficiency," International Electron Device Meeting (IEDM) Tech. Dig., pp. 121–124, December, 2007.

[26] D. Datta, B. Behin-Aein, S. Salahuddin, and S. Datta, "Quantitative model for TMR and spin-transfer torque in MTJ devices," International Electron Device Meeting (IEDM) Tech. Dig., pp. 548–551, December, 2010.

[27] J. C. Sankey, Y. T. Cui, R. A. Buhrman, D. C. Ralph, J. Z. Sun, and J. C. Slonczewski "Measurement of the spin-transfer-torque vector in magnetic tunnel junctions," Nature Phys., Vol. 4, pp.67–71, January, 2008.

[28] C. Augustine, A. Raychowdhury, D. Somsekhar, J. Tschanz, K. Roy, and V. De, "Numerical Analysis of Typical STT-MTJ Stacks for 1T-1R Memory Arrays," International Electron Device Meeting (IEDM) Tech. Dig., pp. 544–547, December, 2010.

[29] J. Li, P. Ndai, A. Goel, S. Salahuddin, and K. Roy, "Design paradigm for robust spin-torque transfer magnetic RAM (STT-MRAM) from circuit/architecture perspective," Transactions on VLSI Systems, Vol. 18, No. 12, pp. 1710–1723, December, 2010.

[30] A. R. Alvarez, B. L. Abdi, D. L. Young, H. D. Weed, J. Teplik, and E. R. Herald "Application of statistical design and response surface methods to computer-aided VLSI device design," IEEE Trans. Computer Aided Des. Integrated Circuits Syst., Vol. 7, No. 2, pp. 272–288, February, 1988.

[31] C. Augustine, A. Raychowdhury, D. Somsekhar, J. Tschanz, V. De, and K. Roy, "Design Space Exploration of Typical STT MTJ Stacks in Memory Arrays in the Presence of Variability and Disturbances," IEEE Transaction on Electron Devices, Vol. 58, No.12, pp. 4333–4343, December 2011.

[32] Shu Lin and Daniel J. Costello. 2004. Error Control Coding, Second Edition. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[33] S. E. Michalak, A. J. DuBois, C. B. Storlie, H. M. Quinn, W. N. Rust, D. H. DuBois, D. G. Modl, A. Manuzzato, S. P. Blanchard, "Assessment of the Impact of Cosmic-Ray-Induced Neutrons on Hardware in the Roadrunner Supercomputer," IEEE Transactions on Device and Materials Reliability, Vol. 12, No. 2, pp. 445–454, 2012.

[34] SPEC CPU2006: http://www.spec.org

[35] Kelin J. Kuhn, Martin D. Giles, David T. Becher, Pramod Kolar, Avner Kornfeld, Roza Kotlyar, Sean T. Ma, Atul Maheshwari, Sivakumar P. Mudanai, "Process Technology Variation," IEEE Transactions on Electron Devices, Vol. 58, No. 8, pp. 2197–2208, 2011.

Author Biographies

Helia Naeimi joined Intel in 2008 and is a member of the Microprocessor and Programming Research Lab. She received PhD and MS degrees from Caltech on 2008 and 2005 respectively, and her Bachelor degree from Sharif University of Technology on 2002. Helia is very passionate about reliability and efficiency. Her research interests lie at the intersection of reliability, low power, and efficient computing. Her work has received Intel Labs division recognition awards and best paper and poster awards.

Charles Augustine received the Bachelors in Electronics from BITS, Pilani, India in 2004 and the PhD degree in Electrical and Computer Engineering

from Purdue University in 2011. He is currently a senior research scientist in the Circuit Research Lab (CRL) at Intel Corporation in Hillsboro, Oregon. His primary research interests include low-power memory and logic using spin-torque devices, low-voltage CMOS circuits, and reliability issues associated with them. He received Best Paper in Session Award at SRC Techcon in 2009, Best Paper Award at ISLPED 2012, AMD Design Excellence Award from Purdue in 2008, nominated for Best Paper Award at ISQED in 2009 and won Bronze medal for academic excellence from BITS, Pilani in 2004. He has held positions at Texas Instruments, ST Microelectronics, Philips Semiconductors, and Freescale Semiconductor, where he worked on CMOS digital integrated circuits and memories, including spin-torque based memories. Charles has published more than 35 papers in refereed journals and conferences.

Arijit Raychowdhury is currently an associate professor in the School of Electrical and Computer Engineering at the Georgia Institute of Technology. He received his PhD in electrical and computer engineering from Purdue University and his BE in electrical and telecommunication engineering from Jadavpur University, India. His industry experience includes five years as a staff scientist in the Circuits Research Lab, Intel Corporation, (2007–2012) and a year as an analog circuit designer with Texas Instruments Inc. (2001–2002). His research interests include digital and MS circuit design, design of on-chip sensors, memory, and device-circuit interactions. Dr. Raychowdhury holds more than 25 U.S. and international patents and has published over 80 articles in journals and refereed conferences. He serves on the technical program committee of several conferences and has received several best paper awards.

Shih-Lien Lu received his BS in EECS from UC Berkeley, and MS and PhD both in CSE from UCLA. He is a principal researcher and leads the memory architecture team at Intel Labs. From 1984 to 1991 he was on the MOSIS project at USC/ISI, which provides research and education community VLSI fabrication services. He was on the faculty of the ECE Department at Oregon State University from 1991 to 2001. His research interests include computer microarchitecture, memory circuits, and VLSI systems design.

Jim Tschanz received the BS degree in computer engineering and the MS degree in electrical engineering from the University of Illinois at Urbana-Champaign, in 1997 and 1999, respectively. Since 1999, he has been a member of the Intel Circuit Research Lab in Hillsboro, Oregon, where he leads a team of researchers working on low-power circuit techniques. His research interests include low-power digital and memory circuits, design techniques, reliability, and methods for tolerating static and dynamic variations. He also taught VLSI design for seven years as an adjunct faculty member at the Oregon Graduate Institute in Beaverton, Oregon. He has published 53 conference and journal papers in this field, has authored three book chapters, and has over 41 issued patents.

ECC TECHNIQUES FOR ENABLING DRAM CACHES WITH OFF-CHIP TAG ARRAYS

Contributors

Wei Wu
Intel Labs Intel Corporation

Shih-Lien Lu
Intel Labs Intel Corporation

Dinesh Somasekhar
IAG Intel Corporation

Rajat Agarwal
IAG Intel Corporation

Error correcting codes (ECCs) are widely used to provide protection against data integrity problems in memory. With continue scaling of technology and lowering of supply voltage, failures in memory are becoming more prevalent. Moreover, the usage and organization of DRAM have also been expanded. Integrating a large-scale DRAM cache is a promising solution to address the memory bandwidth challenge, and this is becoming more compelling with 3D-stacking technology. To enable a high-performance DRAM cache, previous works have proposed storing a tag array off-chip with data in DRAM. The tag-and-data access inevitably changes the traditional access pattern to memory and brings new challenges to ECC schemes due to the granularity of the data access. How to design efficient ECC for new memory usage within the restrictions of commercial DIMMs has emerged as a new challenge.

In this article, we propose two new ECC techniques, Hybrid ECC and Direct ECC Compare. Hybrid ECC is a linear ECC that uses the same bit overhead as a Double Error Correction, Triple Error Detection (DEC-TED) ECC, but provides error correction for more frequent burst error patterns. Direct ECC Compare eliminates the delay and gate overhead caused by comparing multiple encoded words in parallel. The design for off-chip tag storage falls into two major categories, distributed and continuous. For distributed tag storage, we propose to store tags in the ECC chip, protected by Hybrid ECC. For continuous tag storage, we propose separate ECCs for each individual tag to reduce the bandwidth overhead for tag update and the use of Direct ECC Compare to improve the matching latency of encoded tags. A design based on 16-way set associative cache shows that a 30-percent gate count reduction and a 12-percent latency reduction are achieved.

"How to design efficient ECC for new memory usage within the restrictions of commercial DIMMs has emerged as a new challenge."

Introduction

Errors in dynamic random access memory (DRAM) devices have always been a concern in modern computing systems. Memory errors have many possible causes: for example, electrical or magnetic interference such as cosmic rays can spontaneously flip a bit to the opposite state, hardware defects can result in a cell being permanently damaged, and any problem along the data path can corrupt the value reading out of or writing into a bit. Due to continued scaling of technology and lowering of supply voltage, memory faults are becoming more prevalent.[17]

Error correcting codes (ECCs) are used to provide protection against data integrity problems in memory. The most commonly used technique is Single Error Correction, Double Error Detection (SEC-DED), which uses extra 8-bit

sets of ECC bits to protect 64 bits of data.[5] For mission-critical commercial systems, advanced reliability features are supported, such as Chip-Kill*[1][4][6] or Single Device Data Correction (SDDC)[2][8], which can protect data against single x4 or x8 DRAM device failure. More advanced technology called Enhanced Double Device Data Correction (DDDC)[2] even allows recovery from two sequential DRAM chip failures.

ECC design for memory is restricted to a limited bit overhead due to the fixed DRAM chip organization. Unlike normal DIMM, an ECC DIMM usually has 18 rather than 16 × 4 chips or 9 × 8 chips. The extra chips store the redundancy information used for detection and/or correction. The ratio between data bits and ECC check bits is 8:1. To enable stronger protection, smaller devices and wider access are preferred. For example, DDDC can only be enabled on x4 devices under double channel lockstep mode.[2] Therefore, changes to memory data organization may require changes to the ECC technology.

"…changes to memory data organization may require changes to the ECC technology."

Recent work[9][11][12][13][16] has proposed using DRAM as an off-chip cache to address the memory bandwidth problems. Die-stacking and System-in-Package (SiP) technologies enable multiple layers of DRAM to be integrated with processors[14][15], which makes the DRAM cache a more compelling idea. A key challenge for enabling high-performance DRAM cache is how to efficiently manage the tag array. As it's impractical to put the whole tag array in SRAM, a lot of research has been conducted on architecting the tag array off-chip. Based on the manner tags are organized in a DRAM row, most of the solutions fall into two categories, continuous[11][13] or distributed[16][12]. In continuous tag stores, one DRAM row is partitioned into two segments, one for a set of cache lines and the rest for their tags. In distributed tag stores, the tag is placed next to its cache line. The nature of cache involves reading both tag and data for each access. A tag is usually shorter than the memory bus width; checking and recalculating ECC for tags is neither straightforward nor convenient in either tag store scheme.

"A tag is usually shorter than the memory bus width; checking and recalculating ECC for tags is neither straightforward nor convenient in either tag store scheme."

In this article, we propose two ECC technologies that were invented at Intel, and one for each tag store scheme. For distributed tag stores where the tag is associated with its own data entry, we show that storing the tag on the ECC chip by occupying part of reserved ECC bits is feasible. To maintain the reliability feature with reduced check bits, we propose a new ECC called Hybrid ECC. This new code can correct both random errors and burst errors. Random errors are mostly introduced by soft errors such as cosmic rays. Local burst errors can be attributed to device failure or pin/channel fault. With only 19 bits, Hybrid ECC can correct 2-bit random errors or 4-bit burst errors for bit-interleaved 4 transfers (half cache line and a total of 288 bits), leaving 26 bits for tags per cache line.

For continuous tag stores, we propose to have individual ECC protection for each tag entry. Since multiple tags are packed into a few lines, updating a single tag and recalculating ECC require a read-modify-write to the

entire line, which increases the memory bandwidth overhead. In addition, we propose a "Direct ECC Compare" to speed up the address matching for encoded tags. Since the incoming address is always correct, the Direct ECC Compare performs approximate matching for encoded tags, instead of strict matching for decoded tags. It also saves energy by encoding only one incoming tag other than decoding all encoded tags in the same set. We compare two types of tag array designs based on a 16-way set associative cache. The results show a 30-percent gate count reduction and a 12-percent latency reduction.

The rest of the article is organized as follows. The next section, "Background," reviews the memory error mechanisms and ECC algorithm. The section "DRAM Cache Tag Arrays" discusses the tradeoff of two tag store schemes and high-level architecture of our two proposed ECC technologies. The Hybrid ECC for distributed tag stores is described in the section "ECC Technology I: Hybrid ECC." The continuous tag store with separate ECC and Direct ECC Compare is detailed in the section "ECC Technology II: Direct ECC Compare." This is followed by a section that summarizes the article and draws some conclusions.

Background

Errors in DRAM are a common cause for system failure. Researchers have observed that about one third of the machines and 8 percent of DIMMs in the study were affected by correctable errors each year, and the per DIMM correctable error rate is approaching 4000 per year.[17] When the number of affected cells exceeds the correction limit of the ECC, a machine shutdown is forced. If an uncorrectable error misses detection, it leads to applications using corrupted data or even system crash.

"…an uncorrectable error misses detection, it leads to applications using corrupted data or even system crash."

Hardware errors generally fall into two categories: soft (or transient) errors and hard errors. Soft errors mostly occur due to electrical or magnetic interference, such as cosmic rays or alpha particle strikes. Such events change the logical state of one or multiple bits, and cause incorrect data reading. They occur randomly and disappear when the bits are rewritten. Hard errors are related to permanent device damage, which cause a memory bit to return incorrect values consistently, such as a "stuck-at" fault. The faulty devices are usually replaced once detected. There is one kind of error that only lasts for a while or occurs only under certain conditions (such as with low voltage). Unlike hardware errors, they are not permanent. Such intermittent errors are sometimes counted as soft errors.

Memory ECC

The industry standard for DRAM protection is SEC-DED, which is 8 ECC check bits for each 64 bits of data. The check bits are stored in one extra x8 ECC chip or two x4 chips, as illustrated in Figure 1.

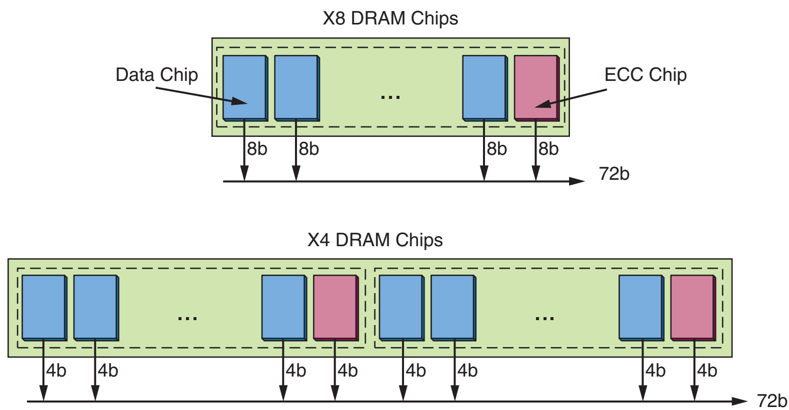


Figure 1: DDR DIMMS with x8 and x4 devices
(Source: Intel Corporation, 2013)

Servers have more critical reliability requirements and stronger protection is provided. For example, the Chip-Kill technology by IBM [6] can correct 4-bit errors in the same nibble. The code requires 16 ECC check bits on 128 bits of data. Intel has similar technology called Single Device Data Correction (SDDC)[2], which protects against single x4 or x8 DRAM device failure. Equivalent technologies are provided by other vendors with different names, such as Chip Sparing or Advanced ECC.[5] Recently, Intel® Xeon® system supports an even more advanced correction technology called Enhanced Double Device Data Correction (DDDC)[8], which allows recovery from two sequential DRAM device failures as well as one more single-bit soft error.

To enable stronger protection, smaller devices and more independent chips are preferred. As illustrated in Figure 1, the ratio between data and check bits is always 8:1. With fixed bus width, smaller device means more independent chips. Advanced ECC technologies have to pay extra bandwidth or capacity for stronger protection. That's why DDDC is only naturally supported on x4 devices. But to enable DDDC or SDDC on x8 devices, lockstep memory mode is mandatory, which requires two memory channels working as a single channel. It reduces the total system memory capacity by one third in a 3-channel system.

"To enable stronger protection, smaller devices and more independent chips are preferred."

"Advanced ECC technologies have to pay extra bandwidth or capacity for stronger protection."

Unlike commercial DRAM chips, on-chip caches are usually custom-designed. This provides more flexibility in choosing ECC protection schemes and granularity. For example, in OpenSPARC*, L2 cache has SEC-DED on a word (32-bit) basis. In AMD Opteron*, both L1 and L2 have 8 ECC check bits for each 64 bits of data. In Intel Xeon, L2 cache is protected by 10-bit SEC-DED while L3 has in-line DEC-TED.

Linear Block Code

All memory ECCs, including SEC-DED, DEC-TED and the Chip-Kill type of symbol correction code, belong to the family of systematic linear block code. The two ECC technologies we propose in this article are based

on the same type of code, and more specifically, the BCH.[7] In this section, we explain some coding basics and properties that will be used in the following sections.

An $[n, k, d]$ code is a linear block code with block length n, information length k, and Hamming distance d. Let $D[k]$ be the k-bit information. The codeword $V[n]$ is the multiplication of D and encoding matrix G.

$$D[k] * G[k, n] = V[n] \tag{1}$$

As a systematic code, the information bits (D) are retained in the codeword V, that is, $V = [D, C]$, while C is the set of ECC check bits. During error decoding, a parity-matrix, which is often referred to as the H-matrix, is multiplied by the codeword. The result is error syndrome, shortened as S.

$$V[n] \times H[n, n-k] = S[n-k] \tag{2}$$

"…any valid codeword must have its syndrome equal to zero, and a nonzero syndrome is essentially the footprint of errors."

The H- and G-matrices are designed in such a way that any valid codeword must have its syndrome equal to zero, and a nonzero syndrome is essentially the footprint of errors. The distributive property of a linear code ensures that a corrupted codeword $V' = V + E$ has its syndrome only determined by errors E:

$$V' \times H = (V + E) \times H = V \times H + E \times H = E \times H \tag{3}$$

For a single-bit error, the syndrome equals the value in the corresponding H-matrix column. For multi-bit errors, the syndrome is the sum of all corresponding columns.

Each ECC is uniquely defined by its H-matrix, since the H-matrix (more specifically, the columns) determines the error syndrome's composition, and thus the error correction and detection capability. For a binary $[n, k, d]$ code, the H-matrix is a binary matrix of the form n by $(n - k)$. However, to facilitate the study, the binary columns are often noted by the Galois Field (GF) elements, that is $\{1, a, a^2, ..., a^{n-2}\}$. For example, H_1 is a binary form of (7, 4, 3) Hamming code; it can also be represented by $GF(2^3)$ primitive element a as in H_2.

$$H_1 = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad H_2 = [a^6 a^5 a^4 a^3 a^2 a^1]$$

The two matrices are totally equivalent, but symbol and polynomial form is more convenient in studying the code properties and will be used instead of the binary form in the following sessions.

DRAM Cache Tag Arrays

Recent work has proposed using DRAM as an off-chip cache to address the memory bandwidth problems. It is desirable to organize the DRAM caches at cache line granularity, because larger page granularity requires too much memory bandwidth, and the miss rate is not low enough to overcome the bandwidth increase. At cache line granularity, the key challenge is the placement of the huge tag array. Storing the tag array on-chip in SRAM is impractical. For example, a 256-MB DRAM cache would require 24 MB of tag storage, which is larger than on-chip LLC.

"At cache line granularity, the key challenge is the placement of the huge tag array."

To avoid SRAM overhead, the alternative is placing the tags in DRAM. Most prior hardware-based approaches toward fine-grain DRAM caches have either stored tags in a continuous region[11][13] or stored with each single cache block[12][16]. We note them as continuous tag store and distributed tag store, as illustrated in Figure 2.

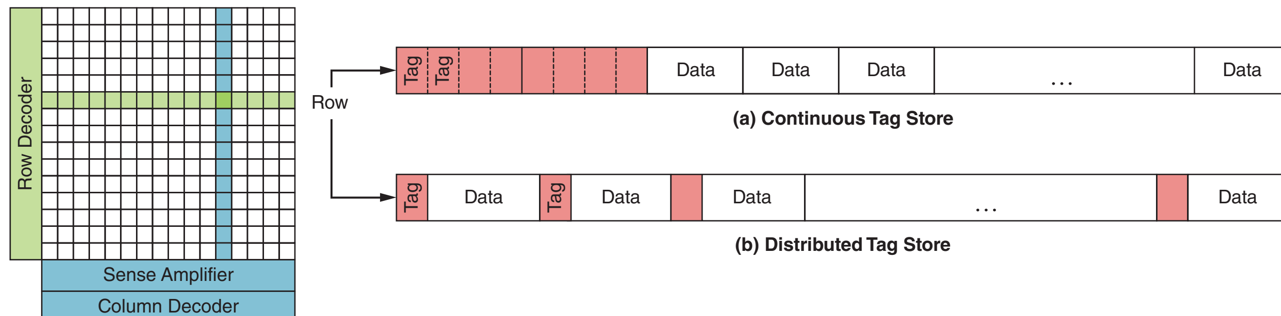


Figure 2: Two trends of placing tags in DRAM cache
(Source: Intel Corporation, 2013)

A cache access must obtain both tag and data, either sequentially or together. Naively placing the tags might result in two full memory accesses. Prior architecture optimizations have been focused on improving the performance, especially the DRAM cache access latency. For example, Loh and Hill[13] proposed reducing the access penalty by locating the tags and data for the entire set in the same row, such that the second access for data is guaranteed to hit in the row buffer. This is a typical continuous tag-store configuration, as we show in Figure 2(A). A recent work from Qureshi[16] eliminates the delay due to tag serialization by streaming data and tag together in a single burst. The design is based on a distributed tag store as shown in Figure 2(B) and it serves the DRAM cache hit much faster.

However none of the works about tag array placement considered the potential effect to the error protection scheme. The bit length of a single tag is obviously smaller than the memory bus width (64 bits). DRAM ECC is not designed for such fine-grain access. Updating a single tag entry would result in a read-modify-write of a whole encoded word, which is at least 72 bits. For advanced ECC technologies, the entire codeword is even longer and equal to a burst of 4 or 8 transfers.

In this article, we propose two ECC technologies, one for each tag store configuration. The base case ECC is an 8b SEC-DED for 64 bits of data.

Distributed Tag Store

For distributed configuration, the tag is stored with data locally. To save bandwidth and avoid the partial write problem during tag updates, we propose merging the data and tag into one "real" single burst; that is, by hiding the tag bits into reserved ECC check bits. By storing tag bits on the ECC chip, data, tag, and check bits will all be transferred together. However, with reduced space for ECC bits, the original code cannot be applied.

"…we propose merging the data and tag into one "real" single burst; that is, by hiding the tag bits into reserved ECC check bits."

To enable the ECC with reduced check bits, we propose a new ECC algorithm, *Hybrid ECC*. As advanced ECC technologies for servers have implied, stronger error correction capability can be achieved by combining the check bits from multiple transfers. The Hybrid ECC uses all partial check bits collected from four data transfers. It corrects both local burst errors and random global errors. The detailed code design and results are presented in the section "ECC Technology I: Hybrid ECC."

Continuous Tag Store

In this configuration, tags are stored in a continuous region. Any tag update would require a full access and recalculate the check bits for entire 64 bits. To reduce the ECC recalculation and entire codeword access overhead during tag update, we propose having separate ECC protection for each tag entry such that each tag entry is independent and can be accessed directly.

"…we propose having separate ECC protection for each tag entry such that each tag entry is independent and can be accessed directly."

At the memory controller side, prior to the tag matching, error detection and potential correction are required for encoded tags. We propose a *Direct ECC Compare* to reduce the address matching latency. The key observations are: the incoming address is always correct, and a codeword within a certain Hamming distance to a valid codeword is guaranteed to be correctable. We eliminate the multiple copies of the decoding circuit for tags and replace them with a single encoder logic circuit for the incoming address. The result shows a 30-percent gate count reduction and a 12-percent latency reduction. The details of this fast matching scheme are discussed in the section "ECC Technology II: Direct ECC Compare."

ECC Technology I: Hybrid ECC

Most ECCs previously considered in the literature have the property that their error-correction capabilities target a specific type of error, either random, burst, or symbol error. For example, a symbol error correction code cannot correct a burst error that ranges across the symbol boundary, and a burst-2 correction code cannot correct two errors that are not adjacent.

In this section, a new code called Hybrid ECC is investigated in which the codeword is protected against two types of errors, both random and burst errors. In a DRAM system, the random error is mostly due to soft errors, and local burst errors are attributed to device-level failure or pin/channel fault. Based on the error coverage, we also denote it as tEC-bBEC, which means the code can correct either t-bit random errors or b-bit burst errors, where b is greater than t.

"…the codeword is protected against two types of errors, both random and burst errors."

The proposed Hybrid ECC is constructed based on regular BCH, and more specifically DEC-TED BCH. The goal is to redesign the DEC-TED H-matrix, reuse the same check bits overhead, maintain the 2-bit random error correction, and at the same time maximize the correctable burst error length.

We have a main observation for standard BCH H-matrix that the matrix columns naturally form a geometric sequence. In the rest of the section, we present this observation and related properties. Then we show how to utilize these properties to quickly evaluate the correction capability for a given H-matrix. We also present a systematic column permutation method such that the new matrix can remain the column properties. Combining the quick evaluation and column permutation, we can easily find the H-matrix that can correct a large number of burst errors. Last, we show some results.

Observations for Standard BCH Code

BCH code is a family of linear block codes. Its SEC-DED and DEC-TED forms are widely used in current computer systems.[7] Let α be a primitive element of $GF(2^m)$ of order n, where $n = 2^{m-1}$ and $\alpha^n = 1$. A standard DEC-TED matrix H_{std} is defined as:

$$H_{std} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^i & \cdots & \alpha^{(n-1)} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \cdots & \alpha^{3i} & \cdots & \alpha^{3(n-1)} \end{bmatrix} \tag{4}$$

The first row has all 1s, which is an overall parity. The rest of the rows are comprised of the first n powers of consecutive powers of α. Let h_i denote the ith column. It's obvious that each column is proportional to its previous one with a fixed ratio:

$$h_{i+1} = K \times h_i, \quad \text{where} \quad K = \text{diag}\,[1, \alpha, \alpha^3] \tag{5}$$

Since K is a constant, the columns are in fact a geometric sequence. Recursively, the whole H-matrix can be represented by K and h_0 only:

$$H_{std} = [1, K, K^2, K^3, \ldots, K^{n-1}] \cdot h_0 \tag{6}$$

"…the syndrome of any burst error is a multiple of the syndrome of its first error bit"

It's easy to prove that the syndrome of any burst error is a multiple of the syndrome of its first error bit; the multiplier is a constant solely determined by the error pattern. For example, the syndrome for a burst −3 at bit i, $i + 1$, and $i + 2$ is:

$$S(b_i, b_{i+1}, b_{i+2}) = h_i + Kh_i + K^2h_i = (1 + K + K^2) \times S(b_i) \tag{7}$$

The coefficient $(1 + K + K^2)$ is fixed and independent of the starting bit position. Similarly, for a burst-4 error with pattern "1101" the syndrome multiplier would be $(1 + K + K^3)$. Each pattern has its own multiplier.

Fast Evaluation for Burst Error Correction Capability

A correctable error means the syndrome of this error is nonzero and distinct from all other correctable errors and detectable errors. To evaluate the correction capability for a given code (that is, H-matrix) by enumerating all possible error patterns and syndromes would be too expensive. An n-bit binary codeword has n single errors, $C(n, 2)$ double errors and $(n - d)$ instances for each burst error of length d. The computation complexity for cross-comparing all syndromes at all possible bit locations is $(n - d) \times (n + C(n, 2)) = O(n^3)$.

"…enumerating all possible error patterns and syndromes would be too expensive."

As burst errors with the same pattern share a common multiplier, a group of syndromes can be represented by a single coefficient. Combined with other BCH algebraic properties, the evaluation can be much simplified.

Let Parity be the part of syndrome that corresponds to the first row in H-matrix, let S_1 be the part to the second row, and let S_3 be the third row. We define the *normalized syndrome factor* for each syndrome as $\xi = (S_1^3/S_3)$.

"We define the normalized syndrome factor *for each syndrome…"*

Single Bit Error: a single error at the *i*th bit and the syndrome is denoted as $S(b_i)$, which equals the *i*th column of H-matrix. According to the definition, the syndrome must have:

$$\text{Parity} = 1 \;\&\&\; S_1^3/S_3 = 1 \tag{8}$$

Obviously, the normalized syndrome factor ξ for all single error is 1.

Burst Errors: for given burst pattern ε, the syndrome can be represented as

$$S(\varepsilon) = (1, K_1, K_3) \times S(b_i)$$

where i is the location of first error bit, and $(1, K_1, K_3)$ is determined by the burst pattern ε. It's easy to prove that all burst errors with pattern ε share a common normalized syndrome factor, that is $\xi = K_1^3/K_3$.

Double Bit Error: given a pair of error bits at position i and j, the syndrome is

$$S(b_i + b_j) = \begin{bmatrix} 1 \\ \alpha^i \\ \alpha^{3i} \end{bmatrix} + \begin{bmatrix} 1 \\ \alpha^j \\ \alpha^{3j} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 + \alpha^d \\ 1 + \alpha^{3d} \end{bmatrix} * S(b_i), \text{ where } d = i - j, \text{ and } \xi = \frac{(1 + \alpha^d)^3}{1 + \alpha^{3d}} \tag{9}$$

The factor ξ is determined by the first bit (i) and the relative bit distance (d). Since the finite field elements are cyclic, the maximum relative distance is no longer than half the loop, that is $n/2$. There are $n/2$ different normalized syndrome factors for all double-bit errors.

Two syndromes that have different ξ values must be different. Two syndromes that share a same ξ value but different start bits are also different, since they have distinct $S(b_i)$. Now the problem of comparing all syndromes becomes comparing all normalized syndrome factors ξ. Odd errors can be distinguished from even errors by simply checking the overall parity. Then, all odd burst errors must have ξ other than 1, that is, a single-bit error. And all even burst errors must have different ξ from any random double error. The total number of comparisons for b burst errors is only $O(n)$:

"Now the problem of comparing all syndromes becomes comparing all normalized syndrome factors ξ."

$$\#\,odd\ burst + \#\,Even\ Burst * \#\,Random\ double = \frac{b}{2} + \frac{b}{2} * \frac{n}{2} = \frac{b}{4} * (2 + n) = O(n)$$

Finding the Best H-Matrix

The burst error correction capability of the standard H-matrix may not be as much as user requires. An easy way to construct a new H-matrix is to rearrange the H-matrix columns. However, the nice property of geometric sequence will be lost if columns are arbitrarily permuted.

The rearrangement property of Fermat's Little Theorem in number theory says, if p and a are co-prime positive integers, then $\{a, 2a, 3a, \ldots, (p-1)a\}$ when reduced modulo p, becomes a rearrangement of the sequence $\{1, 2, 3, \ldots, p-1\}$. Based on this property, we rearrange the H-matrix column by picking one column every L columns, where L and n are co-prime, and n is the degree of the finite field GF(2^m) as we defined in previous section. We called L the step size of rearranged H-matrix. The modified H-matrix will be as the following, where the standard H-matrix is a special case with step size equals to 1:

$$H_{Rearranged} = [h_0 h_L h_{2L} h_{3L} \ldots h_{iL} \ldots h_{(N-1)L}]$$

Galois Field elements are cyclic, so $h_i = h_{i\%n}$. The above matrix is equivalent to H_{std}, only that the column orderings are changed. But all the columns still form a geometric sequence, such that the normalized syndrome factor and fast evaluation method can still be applied. The search space is well controlled, and the best H-matrix can be quickly identified.

Results

The system configurations for one complete code word are: burst length of 4, total transfer of 288 bits, including 32 bytes of data (half cache line), 13 bits of tag and 19 bits of check bits (the same as DEC-TED). Hybrid ECC can correct any one of the three errors illustrated in Figure 3:

"Hybrid ECC can correct any one of the three errors illustrated in Figure 3."

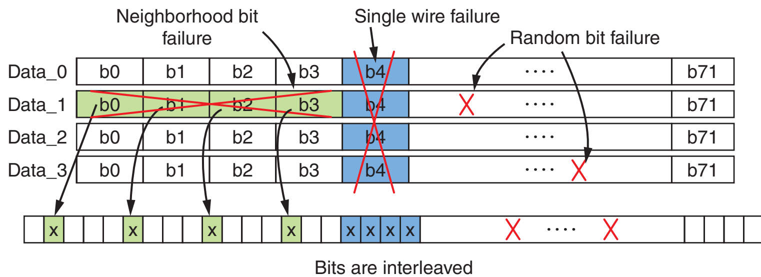


Figure 3: Data bursts per half cache line. The bits of four bursts are interleaved. Three error patterns are popular: 1) random bit error; 2) wire fault; 3) device-level (neighborhood bit) failure
(Source: Intel Corporation, 2013)

With frequent memory scrubbing, the possibility of having two types of error together is very low and can be neglected.

The Hybrid ECC is adapted from conventional DEC-TED and retains the capability of correcting two random-bit errors. The remainder of the multi-bit patterns are listed in Table 1. Two-bit burst errors are not included, since they are special case of 2-bit random errors. Remember that the four transfers are bit-interleaved to form the half cache line data. A single wire fault will manifest as an error in four adjacent bits. The neighborhood bit fault in a same transfer will be separated in a distance of four. The exact number of bit flips depends on the stored bit values.

Pattern Num.	Bit Pattern	Error Source	weight
1	111	Single Wire Fault	Odd
2	1101		Odd
3	1011		Odd
4	1111		Even
5	1___1___1___	Neighborhood Bits Fault	Odd
6	1___1___0___1___		Odd
7	1___0___1___1___		Odd
8	1___1___1___1___		Even

Table 1. Correctable burst error patterns with bit interleaving of four data bursts. “1” means bit-failure, “0” means valid bit, and “–” stands for a valid bit from a different data burst.
(Source: Intel Corporation, 2013)

A qualified H-matrix should have all burst patterns in Table 1 detectable and correctable. As a result, the syndromes must meet the following requirements:

- R1: All burst error syndromes are nonzero.
- R2: All burst error syndromes are distinct from each other.
- R3: All odd burst error syndromes are different from any single-bit error.
- R4: All even burst error syndromes are different from any double-bit error.

R1 is always true, because DEC-TED has a Hamming distance of 6, which means to have an error alias to a valid word requires at least 6 bits of difference. The maximum burst pattern has bit weight of 4. Therefore none of the syndromes would be zero.

Given the number of bits for this specific problem, we need GF(2^9) to cover all 272 bits. The degree of field is $2^9 - 1 = 511$, and the total number of co-primes for 511 is 432. So there are a total of 432 varieties of the H-matrix that we can test. Let the GF generator be $x^9 + x^4 + 1$; we find a working H-matrix to match all the requirements where L equals 47.

ECC Technology II: Direct ECC Compare

“The data comparison circuit is usually in the critical path of a pipeline stage…”

The data comparison circuit is usually in the critical path of a pipeline stage because the result of the comparison determines the flow of the succeeding operations. The common way of comparing two pieces of data with one or both protected by ECC is to retrieve the correct data first by running error check and perform the comparison later. The decoding stage exacerbates the latency criticality. In this section, we present a direct ECC compare technique [18], with which the encoded word can be compared with an incoming data without decoding it first. Direct ECC Compare reduces the critical latency and power consumption due to ECC decoding. The saving

is more significant for situations with multiple data comparison in parallel, such as cache tag match.

For cache tag match, the tag is accessed first, then it must go through ECC decoding and correction before the comparison operation can be performed. In the meantime, the corresponding data array is waiting for the comparison result to decide which way in the set to load the data from. In the DRAM cache, the latency of comparing tags is small compared to the long memory access latency. However, the direct compare technology would still be valuable for reducing total gate count and power consumption.

In Direct ECC Compare, instead of decoding the codeword prior to the comparison, we propose comparing the codeword with the encoded incoming address. By doing so, we replace the tag decoding latency by address encoding latency, which resides at the incoming data side and is less critical. This approach relies on one condition: one of the compare data must be known and valid. For tag match, the incoming address is believed to be valid since it is newly generated and has not been stored in a memory array.

"…replace the tag decoding latency by address encoding latency, which resides at the incoming data side and is less critical."

In the rest of the section, we first review the Hamming distance property of linear ECC codes. Then, we show how these properties can be utilized to compare a codeword (that potentially has an error) with a known valid codeword directly without decoding and correction.

Distance Metrics for Linear ECC Code

The minimum distance gives a measure of how strong a code is in detecting and correcting errors. Given a code that is capable of correcting any combination of t-bit errors and detecting up to r-bit errors, the minimum Hamming distance d equals $(t + r + 1)$. In other words, given a code with minimum distance d, the maximum number of correctable errors is: $t_{max} = \lfloor (d - 1)/2 \rfloor$. The corresponding detectable distance r_{max} is less than $d - t_{max} = \lfloor (d + 1)/2 \rfloor$. If d is an odd number, $r_{max} = t_{max}$; if d is an even number, $r_{max} = t_{max} + 1$.

Codeword Direct Compare

The key idea of Direct ECC Compare is to utilize the information carried by the valid incoming data (referred to as input) to circumvent the necessity of decoding and correction of the stored codeword, which may or may not have errors. For information protected with ECC, in most scenarios, the corrupted codeword is the only copy that contains the original information. Without redundancy provided by ECC there is no other way to retrieve it. However, for data comparison, the absolute values of the stored information are not that important, but rather the relative value to the incoming data is important for deriving the comparison result. In the following, we show that as long as we can determine if it is a match or mismatch, the absolute value itself is not required.

"…is to utilize the information carried by the valid incoming data (referred to as input) to circumvent the necessity of decoding and correction of the stored codeword…"

The stored codeword matches the input as long as the Hamming distance is equal to or less than t_{max}.

"If the Hamming distance falls in the annulus for detectable but uncorrectable errors, the original data cannot be resumed…"

If the stored codeword is error-free, then two codewords match each other only when they are exactly the same, that is, the Hamming distance between two codewords is zero. If the stored codeword is not valid, a potential match implies two facts: the error must be correctable in the first place, and the data recovered from ECC correction should be equal to the input. In other words, this erroneous codeword is within the correctable distance (t_{max}) of the input data. If the Hamming distance falls in the annulus for detectable but uncorrectable errors, the original data cannot be resumed, and this indicates a system fault.

We enumerate the four possibilities below. Given an input data v, let the encoded word be V, and the retrieved codeword from storage be U. The Hamming distance is $d = dist\,(V,U)$. Then the direct compare between V and U has four possible outcomes:

1. $d = 0$: U is valid and is equal to V
2. $d \neq 0$ and $d \leq t_{max}$: U is equal to V but with errors
3. $t_{max} < d \leq r_{max}$: U has an uncorrectable error
4. $d > r_{max}$: U is not equal to V

Hardware Design

Figure 4 shows the data flow of conventional tag matching. The encoded tags go through ECC decoders and ECC correction logic before they are compared with the tag field of the incoming address. If the incoming address does not match to any of the stored tags, a "cache miss" happens. The incoming tag is encoded by the "ECC Gen" logic (encoder) and will replace one out of the 16 ways in the set just referenced.

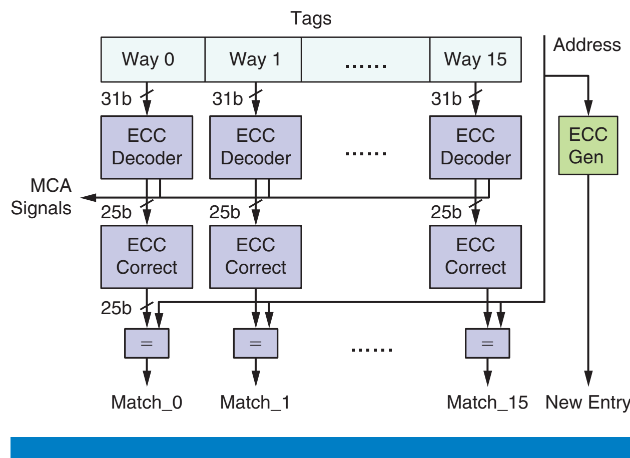


Figure 4: Original tag matching flow with encoded tags
(Source: Intel Corporation, 2013)

Figure 5 illustrates the data flow using the proposed fast compare approach. Information retrieved from the tag directory is compared directly with the incoming tag field of the address after it is encoded. Note that the encoding of the incoming tag can be performed during with tag access, since the memory

access latency is long enough. The decoding and correction time has been removed from the access time.

"The decoding and correction time has been removed from the access time."

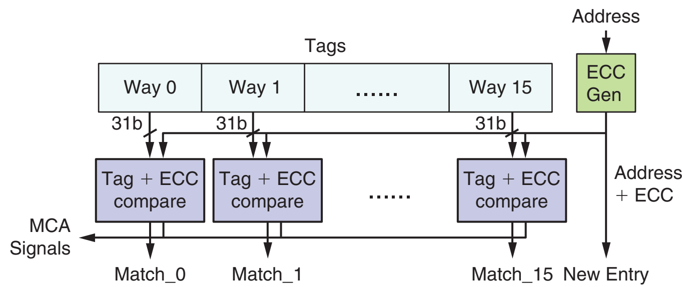


Figure 5: The data flow for Direct ECC Compare
(Source: Intel Corporation, 2013)

In both Figure 4 and Figure 5, the MCA signal refers to the Machine Check Architecture in which the microprocessor has detected an uncorrectable error and the system needs to take action. In general, there are a total of four possible outcomes:

1. Zero bit difference: two encoded tags match exactly and is a cache hit; there is no error in the stored tags.
2. Differ by one bit: A cache hit with correctable errors. The incoming address can be used for correction.
3. Differ by two bits: this is a fault. An uncorrectable error has been detected, MCA generated, and the machine will go into the special handling routine.
4. Differ by more than two bits: this is a mismatch. There are multiple causes of this mismatch. First, the tag address (with or without errors) read differs from the incoming address tag. Second, two address tags are supposed to be the same but with undetectable errors. However, there is no means to distinguish these two cases with the conventional ECC correction technique either. With the first case, detection and correction will be done when the matching address is probed. If it is never probed again, the potential error will be detected and corrected at replacement.

"…detection and correction will be done when the matching address is probed."

The "Tag + ECC Compare" block is implemented as a summing logic. One possible design is to use a parallel counter that counts the total number of different bits. The logic can be optimized by truncating higher significant bits in the carry propagation circuit, since the accurate sum is not required but the relative value is compared to t_{max} and r_{max}.

Results

We assume a 16-way set associative cache and the tag has 25 bits. With SEC-DED protection, the total length of the encoded tag is 31. Table 2 lists the logic implementation cost and latency estimate for each function unit.

	Area (Gate Count)				Latency (Gate Level)			
	AND2	**OR2**	**XOR2**	**Total**	**AND2**	**OR2**	**XOR2**	**Total**
ECC Generator	0	0	79	79	0	0	4	4
Decoder, Correct and Compare	130	30	160	320	4	6	6	16
DC-1	62	61	63	186	1	9	6	16
DC-2	60	90	61	211	0	8	6	14

Table 2: Function unit logic estimation
(Source: Intel Corporation, 2013)

DC-1 and DC-2 are two compare logic designs. DC-1 uses half adders and DC-2 uses full adders for partial sum calculation. Compared to the traditional correction-based design, both approaches have lower gate count. Regarding the latency, DC-1 is about the same as the original decoding-based design, and DC-2 is better than both by 12.5 percent, assuming equal latency for each gate.

The gate count for original decoding-based design is 4.8K. For direct compare design, the numbers are 3K and 3.4K, respectively. The new approach saves roughly 35 percent and 30 percent in total gate counts. With lower gate count and less area, the routing and interconnect complexity should be lower, resulting in smaller routing area and shorter routing latency. Since both designs are implemented in combinational logic, power will be proportional to the total gate count. We expect an approximately 30-percent power reduction.

"With lower gate count and less area, the routing and interconnect complexity should be lower…"

Conclusion

There are two tag storage methods in architecting DRAM caches with tag array off-chip. Architecture optimizations that don't take error protection overhead into consideration could result in read or write bandwidth overhead. To address the problem, we proposed two ECC technologies. One is for continuous tag storage. We suggested that individual tag protection is more efficient and a fast tag matching method is provided to reduce latency and power cost. The other is distributed tag storage. For such a tag store, a new ECC code called Hybrid ECC is presented. Hybrid ECC utilizes multiple transfers to provide similar error coverage with a reduced number of check bits.

References

[1] Sun Microsystems Inc., "OpenSPARC T2 System on Chip (SOC) Microarchitecture Specification," May 2008.

[2] J. Wuu, D. Weiss, C. Morganti, and M. Dreesen, "The asynchronous 24MB on-chip level-3 cache for a dual-core Itanium R-family processor," in *Proc. of the Int'l Solid-State Circuits Conf. (ISSCC)*, Feb. 2005.

[3] J. Huynh, "White Paper: The AMD Athlon MP Processor with 512KB L2 Cache," May 2003.

[4] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway, "The AMD Opteron processor for multiprocessor servers," *IEEE Micro*, Vol. 23, No. 2, pp. 66–76, Mar.-Apr. 2003.

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

[6] IBM, "Chipkill Memory," http://www-05.ibm.com/hu/termekismertetok/xseries/dn/chipkill.pdf

[7] C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," *IBM J. Research and Development*, Vol. 28, No. 2, pp. 124–134, Mar. 1984.

[8] Intel, "Intel® Xeon® Processor E7-8800/4800/2800 Product Families Datasheet," April 2011.

[9] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian. "CHOP: Adaptive filter-based dram caching for CMP server platforms," in *HPCA*-16, 2010.

[10] N. Madan, L. Zhao, N. Muralimanohar, A. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell, "Optimizing Communication and Capacity in a 3D Stacked Reconfigurable Cache Hierarchy," in *HPCA*-15, 2009.

[11] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, "Enabling efficient and scalable hybrid memories using fine-granularity dram cache management," in *Computer Architecture Letters*, Feb. 2012.

[12] L. Zhao, R. Iyer, R. Illikkal, and D. Newell, "Exploring DRAM cache architectures for CMP server platforms," in *ICCD*, 2007.

[13] G. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked DRAM caches," in *MICRO*, 2011.

[14] J.-S. Kim, C. Oh, H. Lee, D. Lee, H.-R. Hwang, S. Hwang, B. Na, J. Moon, J.-G. Kim, H. Park, J.-W. Ryu, K. Park, S.-K. Kang, S.-Y. Kim, H. Kim, J.-M. Bang, H. Cho, M. Jang, C. Han, J.-B. Lee, K. Kyung, J.-S. Choi, and Y.-H. Jun, "A 1.2V 12.8GB/s 2Gb Mobile Wide-I/O DRAM with 4x128 I/Os Using TSV-Based Stacking," in *Proc. of the Intl. Solid-State Circuits Conference*, San Francisco, CA, February 2011.

[15] J. T. Pawlowski, "Hybrid Memory Cube: Breakthrough DRAM Performance with a Fundamentally Re-Architected DRAM Subsystem," in *Proc. of the* 23rd *Hot Chips*, Stanford, CA, August 2011.

[16] Moinuddin K. Qureshi and Gabriel Loh, "Fundamental Latency Trade-offs in Architecting DRAM Caches," in *International Symposium on Microarchitecture (MICRO)*, Vancouver, BC, 2012.

[17] B. Schroeder, E. Pinheiro, and W-D. Weber, "DRAM Errors in the Wild: A Large-Scale Field Study," in *SIGMETRICS/Performance'09*, pp. 193–204.

[18] Wei Wu, Dinesh Somasekhar, and Shih-Lien Lu, "Direct Compare of Information Coded With Error-Correcting Codes," *IEEE Trans. VLSI System*, 20(11): 2147–2151, 2012.

Author Biographies

Wei Wu received her BS degree in radio engineering from Southeast University, Nanjing, China in 2000, and PhD degree in computer science from the University of California at Riverside in 2008. She then joined Intel labs in Hillsboro, Oregon and currently a staff research scientist in Memory Architecture team. Her research interests include reliability, error correction algorithm, low power and cache memory architecture design.

Shih-Lien Lu received his BS in EECS from UC Berkeley, and MS and PhD both in CSE from UCLA. He is a principal researcher and leads the memory architecture team at Intel Labs. From 1984 to 1991 he was on the MOSIS project at USC/ISI, which provides research and education community VLSI fabrication services. He was on the faculty of the ECE Department at Oregon State University from 1991 to 2001. His research interests include computer microarchitecture, memory circuits, and VLSI systems design.

Dinesh Somasekhar is a Senior Staff Scientist at Intel. He is currently responsible for the memory strategy on the high-performance-computing program under Intel Federal. He received the B.E. degree in Electronics Engineering from the Maharaja Sayajirao University Baroda, India, in 1989, the M.E degree in Electrical Communications Engineering from Indian Institute of Science Bangalore, India, in 1990, and the Ph.D. degree from Purdue University in West Lafayette in 1999. From 1991 to 1994 he was an I.C. Design Engineer at Texas Instruments, Bangalore, India, where he designed ASIC compiler memories and interface I.C.s. From 1999–2011 was with Circuits Research Lab, Intel R&D, Hillsboro, Oregon. From 2011–2012 he was with GlobalFoundries, Sunnyvale, CA. He has published 35 papers, 3 book chapters, and holds over 70 patents in the field of VLSI.
Dr. Somasekhar served as Mentor at the Semiconductor Research Consortium, and has participated in the Technical Program Committee of ISLPED, ISQED, DATE, GLVLSI and CICC.

Rajat Agarwal is a Principal Engineer with Intel Corporation. He is the lead architect for HPC memory architecture driving the memory architecture of Xeon Phi product line. His research interests are focused around new memory

technologies to scale the memory BW wall and memory reliability. Prior to his current role, Rajat has been a design manager with the server chipset group and drove the development of Seaburg chipset for workstation market. Before joining Intel, Rajat had worked with Qualcomm, ST Microelectronics and Indian Space research organization. Rajat holds 10 patents in memory controller policies and memory RAS.

IMPROVING ERROR CORRECTION IN NAND WITH DOMINANT ERROR PATTERN DETECTION

Contributors

Ningde Xie
Storage Technology Group,
Intel Corporation

Jawad Khan
Storage Technology Group,
Intel Corporation

With technology scaling, NAND devices suffer from high raw bit error rate (RBER) incurred by device physics variations at sub-20nm. At the same time, their error patterns also exhibit unbalanced characteristics. Circuit techniques have been explored to overcome the unbalanced error behaviors. However they cannot fully purge them and these techniques require extra overhead on the device side. On the other hand, the ECC engine usually treats the errors in NAND devices as random errors, which is not true in real NAND devices. This article proposes to fully utilize the error pattern characteristics abstracted from the NAND device to facilitate ECC decoding, so that we can simplify the NAND device design to reduce cost and/or improve the system's overall reliability. We describe the ECC engine process flow. We also show decoding gains under various flipping asymmetric bits.

Introduction

The continuous bit cost reduction of NAND flash memory mainly relies on aggressive technology scaling. Besides technology scaling, multi-level per cell (MLC) technique has been widely used to further improve effective storage density and hence reduce bit cost of NAND flash memory. Because of its obvious bit cost advantage, MLC NAND flash memory has been increasingly dominating the global flash memory market. In current design practice, most MLC NAND flash memories store 2 bits per cell, while 3-bit-per-cell NAND flash memories have been recently reported.[1]

"RBER in NAND continues to increase as industry is pushing technology scaling and storing more than one bit into a single cell."

RBER in NAND continues to increase as industry is pushing technology scaling and storing more than one bit into a single cell. Currently, errors are usually assumed to be randomly distributed, and error patterns are independent of pre-stored values. However, due to various physical phenomena such as program disturb, single bit charge loss (SBCL), intrinsic charge loss (ICL), and electron trap effect, the NAND device tends to suffer from certain specific error patterns.[2][3] For example, for the voltage of LSB cell in 3-bit-per cell(3bpc) NAND, the L0 cell in a "7–0–7" pattern write sequence is more likely to move upward because of program disturb. Also, because of the iterative program-verify algorithm for NAND programming, the cell has much larger probability of experiencing a lower state to higher state misplacement than a higher state to lower state misplacement. Additionally, with ICL and SBCL, the distribution of Vt tends to move downward.

In current NAND devices, people adopt circuit techniques to reduce these unbalanced cell effects as much as possible; however, it's not only expensive to take time and money to design, tune, and optimize the circuits but this strategy

also sacrifices performance in the NAND device itself due to the extra processing. At the same time, even with the help of these circuits, from the system side, we still observe unbalanced error patterns due to device inherent characteristics, and the NAND has to reserve a large read window budget for them.

Dominant error pattern detection in read channels has been proposed to facilitate the detector and ECC decoding.[4] In NAND, dominant error patterns vary through its whole lifetime, due to aging (program/erase cycle). They are also changing with NAND's retention and read/write disturb. Therefore, we propose to have a dominant error pattern list; the ECC engine can use a trial-and-error decoding until it goes through all patterns on the list or decoding succeeds. Such an error pattern list can be pre-established when the device is manufactured, or we can also store it in the controller. Although it is not mandatory, for efficient operation, the controller may also keep tracking the age of the NAND and data retention time so that more accurate error patterns are selected for ECC engine. Once the NAND logic is equipped with ECC designed for unbalanced errors, it is there in digital circuits. The error list may require updates but the ECC circuit doesn't need the extra effort to tune and optimize in each generation that the current NAND device does. Also, the overall performance overhead is small, because ECC is triggered based on need, unlike the circuit-level change in the device, which will be in function all the time.

"In NAND, dominant error patterns vary through its whole lifetime…"

BCH codes with classical hard-decision decoding algorithms[5] are being widely used in current NAND controllers. As the industry continues to push the technology scaling envelope and pursue aggressive use of multi-level per-cell storage, raw storage reliability of NAND flash memory inevitably continues to degrade, which quickly makes current design practice inadequate and hence naturally demands more powerful ECCs. Because of their well-proven superior error correction capability with reasonably low decoding complexity, advanced ECCs, such as low-density parity-check (LDPC) code with soft-decision decoding algorithms, appear to be promising candidates. Due to the sparse nature of its parity check matrix, we can further use the built-in parity checks in LDPC to help us identify error candidates without adding extra parity check bits as people normally do in the read channel.

Dominant Error Detection and Decoding in NAND

In this article, we propose to feed the information of dominant error patterns in NAND devices into the ECC engine at the SSD controller so that ECC is more informed about the NAND channel and able to use this extra information to improve its error correction capability. For example, once we have an ECC decoding failure, we can try to flip candidate bits based on the parity check failure and the dominant error patterns and try decoding again. In order to detect the dominant error pattern, we can add a weak parity on top of the existing ECC. However this will reduce the coding gain.

"…ECC is more informed about the NAND channel and able to use this extra information to improve its error correction capability."

Dominant Error Pattern Detection

Most recently, due to its superior error correction capability, LDPC is attracting more and more interest in the most recent SSD applications.[6][7][8] The sparse parity check matrix in LDPC gives us an opportunity to avoid adding any coding overhead to detect error patterns. For an LDPC code with parity check matrix H, X is a codeword if $H \cdot X = 0$. Therefore, "1" entries in each row of H check the parity of their corresponding bits in a codeword. For example, as Figure 1 shows, the fourth and fifth "1" entries in the second row of H check the fourth and fifth bits in a codeword respectively. If there is an error in one of the fourth and fifth bits (for example, the fourth bit becomes 1 because of noise in NAND), the parity check results will be nonzero. In conventional ECC decoding, it decodes based on the assumption that all errors are random, which is not true in NAND-based systems.

"…if the current dominant error pattern is "1" to "0", then the ECC engine will be told that bit "0" that violates parity check is a high risk candidate…"

$$\underbrace{\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}}_{H} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Figure 1: Parity Check in LDPC
(Source: Ningde Xie, 2013)

Based on the dominant error patterns in NAND and combining this parity check information provided by the sparse parity check matrix H, we are able to flip some suspicious error bits during decoding. For example, if the current dominant error pattern is "1" to "0", then the ECC engine will be told that bit "0" that violates parity check is a high risk candidate and it can flip it on purpose to help the decoding.

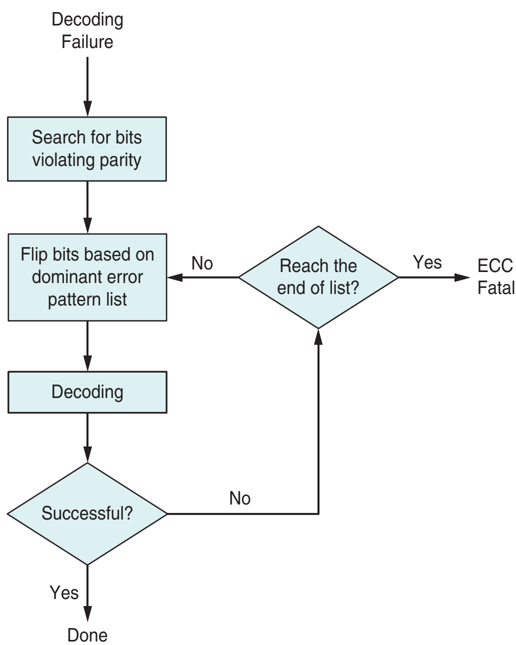


Figure 2: Decoding flow chart with dominant error patterns information
(Source: Ningde Xie, 2013)

Decoding with Dominant Error Patterns

Figure 2 shows the ECC engine process flow when a decoding failure happens. Whenever an ECC decoding fails, ECC engine checks those bits where their parity checks are nonzero. Then based on the dominant error pattern list, it starts to search for the error pattern on the top of the list. Whenever there is a match, it intentionally forces these bits to the value they are supposed to be. In order to increase the chance to flip the "right" error bits, this error pattern information should be combine with the "bit flipping"[9] decoding process in LDPC hard decoding.

NAND Channel Modeling

In order to quantitatively verify the effectiveness of the above proposed method, a mathematical channel that takes into account program method and common noise in NAND is developed according to the method proposed in [10].

Each NAND flash memory cell is a floating gate transistor whose threshold voltage can be configured (or programmed) by injecting a certain amount of charges into the floating gate. Before a flash memory cell is programmed, it must be erased (that is, all the charges from the floating gate must be removed, which sets the gate's threshold voltage to the lowest voltage window). It is well known that the threshold voltage of erased memory cells tends to have a wide Gaussian-like distribution.[11] Hence, we can approximately model the threshold voltage distribution of erased state as a Gaussian distribution function.

"Each NAND flash memory cell is a floating gate transistor whose threshold voltage can be configured (or programmed) by injecting a certain amount of charges into the floating gate."

During NAND programming, threshold voltage control is typically realized by using an incremental step pulse program (ISPP).[12] With such a program-and-verify strategy, each programmed state (except the erased state) associates with a verify voltage that is used in the verify operations and sets the target position of each programmed state threshold voltage window. Denote the verify voltage of the target programmed state as Vp, and program step voltage as ΔVpp. The threshold voltage of the programmed state tends to have a uniform distribution over [Vp, Vp + ΔVpp] with the width of ΔVpp.

Flash memory program/erasure (P/E) cycling causes damage to the tunnel oxide of floating gate transistors in the form of charge trapping in the oxide and interface states, which directly results in threshold voltage shift and fluctuation, and hence gradually degrades memory device noise margin. Major distortion sources include

- Electrons capture and emission events at charge trap sites near the interface developed over P/E cycling directly result in memory cell threshold voltage fluctuation, which is referred to as random telegraph noise (RTN).
- Interface trap recovery and electron detrapping gradually reduce memory cell threshold voltage, leading to the data retention limitation.

RTN causes random fluctuation of memory cell threshold voltage, where the fluctuation magnitude is subject to exponential decay. Hence, we can model the probability density function of RTN-induced threshold voltage fluctuation as a symmetric exponential function.

In NAND flash memory, the threshold voltage shift of one floating gate transistor can influence the threshold voltage of its neighboring floating gate transistors through parasitic capacitance-coupling effect.[13] This is referred to as cell-to-cell interference, which has been well recognized as the one of major noise sources in NAND flash memory. Threshold voltage shift of a victim cell caused by cell-to-cell interference can be estimated by adding all weighted neighbor threshold voltage shift of interfering cells that are programmed after the victim cell. Each weighting factor, which is also called coupling ratio, can be estimated by the ratio of parasitic capacitance between the interfering cell and the victim cell.[10]

Simulation Results

As a case study of this proposed work, we use the above channel model to show the improvement in NAND. For simulation setup, we follow the parameters in [10] where the most popular MLC NAND is used and carefully selected

cell-to-cell coupling factors; P/E cycles affects and retention time according to real-life NAND.

We first simulate the case where program disturbance is the dominant error source. It is a very common scenario for a user to program more than tens of pages of NAND and then come back to read the data back immediately. Therefore, we set up the simulation model by programming cells with random bits, which randomly disturb victim bits with various strengths. We also assume the retention time and P/E cycle to be 0 to simulate the almost new NAND. In this case, because of the program disturbance, we will see bit errors are mostly "1" to "0" misplacements. We implemented a hard decoding LDPC decoder using a bit flipping algorithm. (Note that although soft decoding brings the largest coding gain, it requires a very large read latency overhead in NAND; therefore we normally try hard decoding first whenever hard sensing results are available to the LDPC decoder). In this implementation, when a normal decoding fails, it will try to decode again by randomly flipping some of "0" bits that violate most parities.

"We implemented a hard decoding LDPC decoder using a bit flipping algorithm."

Figure 3 shows the simulation results for the proposed ECC decoding with dominant error flipping and conventional ECC decoding without dominant error flipping. We can see very obvious coding gain in this case.

"We can see very obvious coding gain in this case."

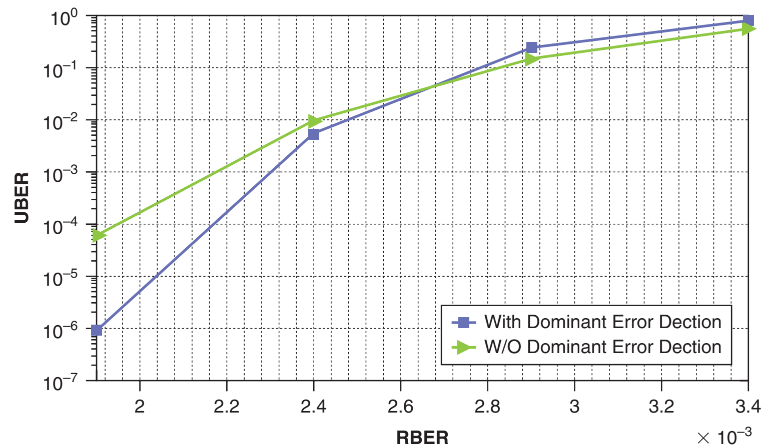


Figure 3: Simulation results for write disturb dominated errors (Source: Ningde Xie, 2013)

As discussed above, after NAND is cycled towards its end of life, combing the data retention, we will see more "0" to "1" bit errors. To simulate this scenario, we set the channel model to be cycled at 5000 and adjust the retention time to three months to simulate the charge loss. This is also a very common phenomenon in real life after the NAND has been used for a long time and the user tries to read data that was stored a while ago. Figure 4 shows the simulation results for the proposed ECC decoding with dominant error flipping and conventional ECC decoding without dominant error flipping. Again, with dominant error detection in ECC decoding flow, we get significant coding gain.

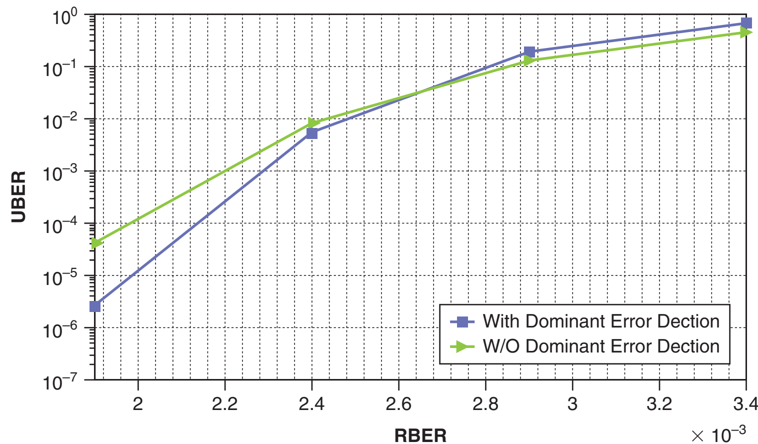


Figure 4: Simulation results for retention dominated errors (Source: Ningde Xie, 2013)

Comparing Figure 3 and Figure 4, we noticed that the coding gain is much less in Figure 4. It is because when we set larger P/E cycle and retention time, we actually mix all noises together, which include program disturbance as well. We also see the coding gain tends to increase with lower RBER in both cases. This is because with lower RBER, dominant error detection used in the simulation can more accurately detect error locations. This also explains why at very high RBER, we see no gain at all.

The above implementation is very simple and effective. It can be further optimized by utilizing other information in NAND. For example, we may use neighbor cell values to help the ECC decide if a bit needs to be flipped or not. Of course, this requires the knowledge of how the device does the interleaving internally so that we can locate the real neighbors bits of a victim cell.

"The above implementation is very simple and effective. It can be further optimized by utilizing other information in NAND."

Conclusions

With this proposed method, we can either simplify the process at the device level to reduce design cost and time to market without sacrificing the overall system reliability (for example, delivering the NAND devices for SSD employing this proposed method without adding peripheral circuits to mitigate program disturb, ICL, and SBCL can be faster than normal release of the NAND devices with those circuits ready in the device). Alternatively, we can also leave the NAND device as is and improve the overall system reliability by taking advantage of this information enhanced ECC correction capability. Since ECC has already been used in any NAND-based system, it requires very little hardware overhead to include this extra information. Also, the normal hard decoding failure works for the majority of the errors, so this extra effort is only triggered when normal decoding doesn't succeed the first time, which further reduces its impact on the overall system performance.

References

[1] Li, Yan, et al. "128Gb 3b/cell NAND flash memory in 19nm technology with 18MB/s write rate and 400Mb/s toggle mode." *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*. IEEE, 2012.

[2] R. Motwani and C. Ong, "Robust decoder architecture for multilevel flash memory storage channels," in Computing, Networking and Communications (ICNC), 2012 International Conference on. IEEE, 2012, pp. 492–496.

[3] Intel, Micron Introduce 25-Nanometer NAND—The Smallest, Most Advanced Process Technology in the Semiconductor Industry, http://www.intel.com/pressroom/archive/releases/20100201comp.htm, Feb., 2010.

[4] G. Sonu, "Dominant error correction circuitry for a viterbi detector," June 17 2003, US Patent 6,581,181.

[5] R. E. Blahut, Algebraic Codes For Data Transmission, Cambridge University Press, 2003.

[6] Wang, Jiadong, et al. "LDPC Decoding with Limited-Precision Soft Information in Flash Memories." *arXiv preprintarXiv: 1210.0149* (2012).

[7] N. Duann, "Error Correcting Techniques for Future NAND Flash Memory in SSD Applications," in Flash Memory Summit, 2009.

[8] R. Motwani, Z. Kwok, and S. Nelson, "Low density parity check (ldpc) codes and the need for stronger ecc," Flash Memory Summit, 2011.

[9] Jiang, Ming, et al. "An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes." *Communications Letters, IEEE* 9.9 (2005): 814–816.

[10] Dong, Guiqiang, et al. "Estimating Information-Theoretical NAND Flash Memory Storage Capacity and its Implication to Memory System Design Space Exploration." *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 20.9 (2012): 1705–1714.

[11] Takeuchi, Ken, Tomoharu Tanaka, and Hiroshi Nakamura. "A Double-Leve1-Vth Select Gate Array Architecture for Multilevel NAND Flash Memories." *IEICE Transactions on Electronics* 79.7 (1996): 1013–1020.

[12] Suh, Kang-Deog, et al. "A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming scheme." *Solid-State Circuits, IEEE Journal of* 30.11 (1995): 1149–1156.

[13] Lee, Jae-Duk, Sung-Hoi Hur, and Jung-Dal Choi. "Effects of floating-gate interference on NAND flash memory cell operation." *Electron Device Letters, IEEE* 23.5 (2002): 264–266.

Author Biographies

Ningde Xie received his BS and MS degrees in Radio Engineering from Southeast University, Nanjing, China, in 2004 and 2006, respectively, and the PhD in Electrical, Computer and Systems Engineering Department at Rensselaer Polytechnic Institute in 2010. He then joined the Storage Technology Group at Intel Corporation. His research interests include application of emerging nonvolatile memory (NVM) in computing, VLSI systems, and architecture design for storage and communication systems. Currently, he is working on the next generation NVM and its applications in computing.

Jawad Khan is an SSD architect with Intel Corporation and is responsible for SSD architectures, technical innovation, and invention in solid state storage devices. Jawad has particular expertise in innovating compression, encryption, and data recovery algorithms for solid state storage devices, and has two issued patents. Jawad received both is MS and PhD degrees from the University of Cincinnati.

MEMORY CONTROLLER–LEVEL EXTENSIONS FOR GDDR5 SINGLE DEVICE DATA CORRECT SUPPORT

Contributors

Javier Carretero
Intel Labs, Intel Corporation

Isaac Hernández
Visual and Parallel Computing, Intel Corporation

Xavier Vera
Intel Labs, Intel Corporation

Toni Juan
Visual and Parallel Computing, Intel Corporation

Enric Herrero
Intel Labs, Intel Corporation

Tanusú Ramírez
Intel Labs, Intel Corporation

Matteo Monchiero
Intel Development, Intel Corporation

Antonio González
Intel Labs, Intel Corporation

Nicholas Axelos
Intel Labs, Intel Corporation

Daniel Sánchez
Intel Labs, Intel Corporation

Support for Reliability, Availability, and Serviceability (RAS) is one of the quintessential features of computing systems targeting the server and mission-critical markets. Among these RAS features, Chipkill* stands out as the most crucial for main memory protection. IBM Chipkill protects the main memory from the failure of an entire memory chip, as well as multi-bit faults from any portion of a memory chip. Similar technologies from other vendors are Single Device Data Correction (SDDC) from Intel, Sun Extended ECC* and HP Chipspare*.

However, some advanced memory technologies (such as GDDR5) do not allow traditional SDDC implementation, since their specification does not include extra devices to store error correction codes (ECC codes).

Some future high performance computing products hitting the server market will be based on these advanced memory technologies. In this article we propose a method to provide SDDC (single device data correct) support at the memory controller level for memory technologies that inherently have no RAS support for memory contents protection. Specifically, we focus on how to provide single-device SDDC support for GDDR5 memory. The technique allows the failure of 1/8 of the memory devices to be tolerated by using 25 percent of the memory to store error correction codes.

We also describe how the technique can be implemented for RAS-less memory technologies feeding a wider data bus than GDDR5 (such as DDR3, which in fact uses narrower devices). This opens the possibility to offer high reliability with cheap DIMM devices. We also describe how to provide SDDC support without the use of lockstepped memory channels.

"…we propose a method to provide SDDC (single device data correct) support at the memory controller level for memory technologies that inherently have no RAS support for memory contents protection. Specifically, we focus on how to provide single-device SDDC support for GDDR5 memory."

Introduction

Advanced memory technologies post-DDR3 provide very high memory bandwidth with low implementation costs and high capacity. This is the case for GDDR5 with multiple memory channels. These features make these memory technologies very suitable not only for graphics cards but also for high performance computing (HPC) systems. Some HPC Intel products, such as the prototype product codenamed Knights Ferry (KNF) and the commercial product Intel® Xeon Phi™ (formerly codenamed Knight's Corner, or KNC), are based on these or similar advanced memory technologies, and they are also targeting the server segment.

Targeting the server segment inevitably implies offering memory-level RAS techniques, such as SECDED (single error correction / double error

detection), memory mirroring, memory sparing, and memory migration. However, there are several road-blocking issues related to these advanced memory technologies.

First, some memory technologies do not provide extra devices to protect memory contents. This is the case of the GDDR5 standard.[1] As a consequence, there is no native support to build these RAS techniques on top of GDDR5-based products. Second, due to cost, energy overhead, memory capacity restrictions or vendor strategies, the type of available memories may not include RAS features.

"…some memory technologies do not provide extra devices to protect memory contents. This is the case of the GDDR5 standard."

IBM Chipkill and related techniques allow the correction of errors resulting from the failure of one or several memory devices.[12] As an example, dual device data correction (DDDC[2]) memory controllers allow correcting two failing x4 DRAM devices provided the failures are separated in time out of a set of thirty-six x4 devices. Devices with more output pins tend to be more power efficient than those with fewer pins because the energy per memory access can be amortized over more output bits.[10] This is the reason why other Intel products, such as the Intel® Xeon® 5500 family and the Intel Xeon 6500/7500 series, also extend their protection to DDR x8 devices.

However, supporting wide devices comes with an increase either in the code overhead with respect to the protected word, an increase in the word size, or an increase in the access granularity (amount of data obtained from the memory). Clearly, for wide memory devices such as GDDR5 (two x16 devices per memory module), SDDC support represents a challenge.

"…for wide memory devices such as GDDR5 (two x16 devices per memory module), SDDC support represents a challenge."

To our knowledge, there is no previous solution addressing the problem of device error detection and/or correction for advanced memory with no RAS support. Yoon and Erez[3] propose using address virtualization in order to store the codes in regular memory devices. However, this technique requires microarchitectural changes in the core and OS modifications as well. Also, the paper targets regular DDR2 memories.

Background Information

In this section we first give a general description of existing memory RAS techniques for tolerating failures of memory chips. We then describe the high-level architecture of GDDR5 memory.

Implementation Examples

Several products have hit the market offering RAS capabilities for surviving to failures of an entire memory chip. We present some of them.

Oracle Sun UltraSPARC* T1/T2 and AMD Opteron*

Some products provide SSC-DSC (single symbol correct – double symbol detect) protection for x4 DDR2 DIMMs devices. This is the case of the Oracle Sun UltraSPARC T1/T2 and AMD Opteron systems.[4][5] Memory

RAS support is constructed by using b-adjacent error correction codes.[7] These two products use the 4-check-symbol error code.[8] In these cases, the four 4-bit adjacent check symbols protect thirty-two 4-bit data symbols. This allows a simple implementation when using two x4 DDR2 DIMMS residing on two different channels working in lockstep mode. With a burst size of 4, a whole cache line is accessed within the 4 accesses of a DDR2 burst. Specifically, in every access of a burst, one data word of 128 (16 data devices per DIMM × 2 DIMMs × 4 bits per device) bits is obtained and is protected with a code of 16 bits (2 RAS devices per DIMM × 2 DIMMs × 4 bits per device). Clearly, this allows recovering the failure of one x4 device, out of a set of 36 devices.

Intel's DDDC

Some Intel products improve memory reliability by providing dual-device data correction (DDDC) in lockstep mode for x4 DRAM chips, and additional single-bit error correction[2] (correcting two failing x4 DRAM devices "provided the failures are separated in time" out of a set of thirty-six x4 devices). DDDC was initially implemented on the Intel® Itanium™ 9300 series (IPF) and on the Intel Xeon E7-x8xx series (x86). At a high level, DDDC is implemented by using two x4 ECC-DIMMS in two channels working in lockstep mode with DDR3 burst chop mode (forced burst of 4).

DDDC is supported for x4 devices by means of lockstepped channels. However, SDDC (single device data correction) is supported for x4 and for x8 devices. Whereas SDDC for x8 devices also requires two lockstepped channels, SDDC for x4 devices can work on independent memory channels.

IBM Blue Gene/P*

In the IBM Blue Gene/P system every memory controller communicates through DDR2 protocol via a 160-bit-wide bus.[6] Of these bits, 128 are user data and 32 bits are devoted for RAS purposes. This represents an overhead of 25 percent. However, this redundancy allows storing address parity bits, spare bits, and enhanced ECC protection data. Overall, it can detect/correct up to six adjacent bits and tolerate the failure of two x8 DRAM chips. However, no details on the implementation are available.

"GDDR5 uses a burst size of 8…"

"The data input/output bus consists of 32 data pins…"

"…every bus clock cycle 4 chunks of 32 bits are transmitted."

"…an access with a burst size of 8 provides 256 bits over two clock cycles."

GDDR5 Basics

Before describing our technique, it is first necessary to understand the basics of the GDDR5 internal architecture.[1] GDDR5 uses a burst size of 8 (8n prefetch scheme) to achieve high-speed bus operation while decreasing the internal memory core frequency for power savings. The data input/output bus consists of 32 data pins (as opposed to the 64 bits of DDR3), and in every bus clock cycle 4 chunks of 32 bits are transmitted. Hence, an access with a burst size of 8 provides 256 bits over two clock cycles. There is no support for burst chop of 4, as opposed to DDR3 memory. The data bus is logically split into 4 bytes, and two extra signals are added to each byte: the data bus inversion signal (to reduce the noise on the high-speed interface and power dissipation) and an error detection and correction signal (to catch errors in the data transfer

through the bus). Moreover, partial writes where individual bytes are excluded from write operations are also supported through the use of an extra data mask pin associated with each data byte. This avoids costly read-modify-write operations.

"…partial writes where individual bytes are excluded from write operations are also supported…"

The link between the memory controller and GDDR5 is protected by means of a cyclic redundancy check (CRC) code. This CRC code allows the detection of all single and double bit faults occurring in the bus for data and bus inversion signals, as well all bursts errors of length no bigger than 8. Upon error detection, the command that caused the error should be repeated and also a retrain of the transmission line can be performed to adapt to varying operating conditions on such a high-bandwidth bus.

According to the standard[1], GDDR5 memories run with two different clocks: commands and addresses are referenced to the differential clock (CK and /CK). Commands are registered at every rising edge of CK, whereas addresses are registered at the rising edge of CK or /CK. Read and write data are referenced to both edges of a free-running differential forwarded clock (WCK, /WCK), which replaces the pulsed strobes used in previous DRAMs. This relation between clocks and data rates is depicted in Figure 1. This means that there is a x4 relationship between data rate and CK clock, as opposed to the x2 relationship in DDR3. Differential clocking allows a more precise communication and this is the reason of the dramatic increase in bandwidth provided by GDDR5 (it allows between 4 and 8 Gbps).

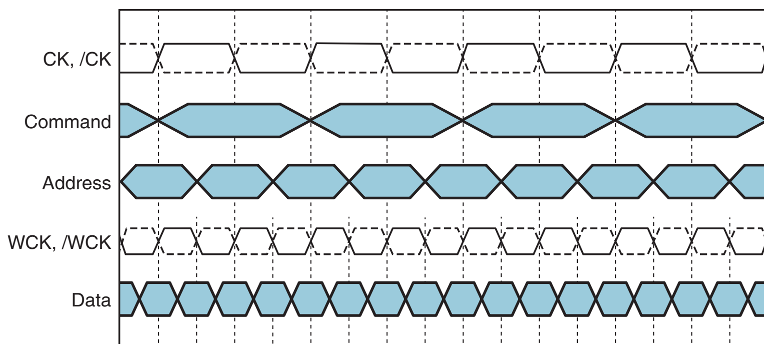


Figure 1: GDDR5 clocks and data rates
(Source: Intel Corporation, 2013)

On top of this, the GDDR5 standard supports a new mode of operation called *2x16 clamshell mode*. Basically, as Figure 2 shows, this feature allows doubling the memory capacity by adding an extra GDDR5 module on every existing memory channel, at the expense of a decrease in the bus clock frequency. The data travelling through the bus is provided by the two memory modules at the same time, each of them providing 16 bits. Every pair of 16 bits is provided by one of the two devices in a module. Note that during clamshell mode, we have four x16 devices per memory channel.

"…GDDR5 standard supports a new mode of operation called 2x16 clamshell mode.*"*

"…this feature allows doubling the memory capacity by adding an extra GDDR5 module on every existing memory channel, at the expense of a decrease in the bus clock frequency."

"The data travelling through the bus is provided by the two memory modules at the same time, each of them providing 16 bits."

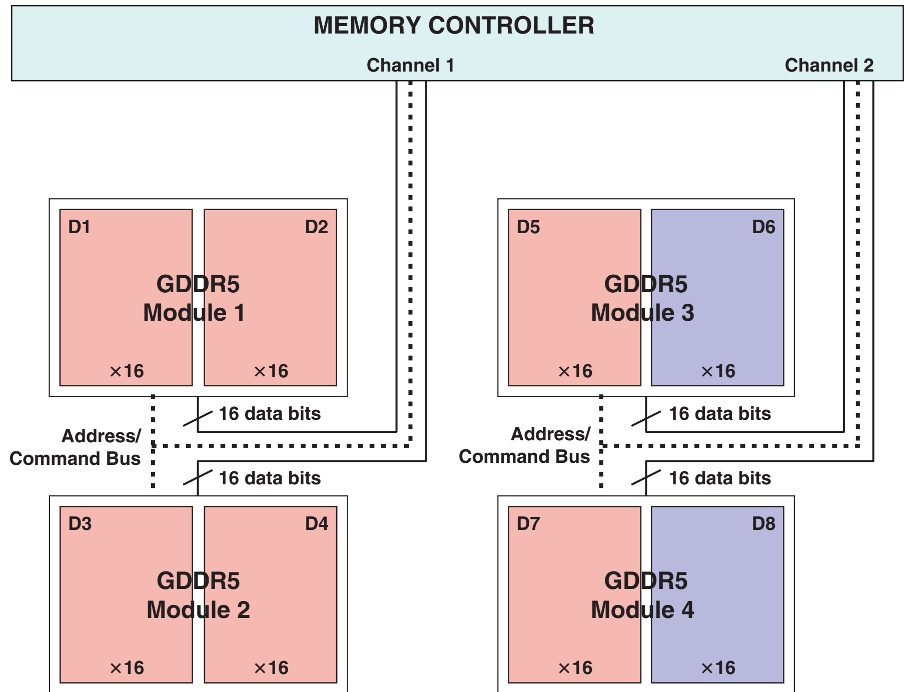


Figure 2: Two channel memory controller working in clamshell mode.
(Source: Intel Corporation, 2012)

SDDC Proposal

Our approach to provide SDDC support is to use *b-adjacent error detection-correction codes.*[7][3] Given that multi-bit errors typically affect contiguous bit locations, computing systems can exploit this fact and use error correction codes for adjacent faulty locations. This includes the b-adjacent error detection-correction codes, where a word is divided in chunks of *b* bits, and codes can correct any number of bit flips in at least one of these chunks. The different adjacent codes differ in the maximum size of the word they can protect, the code overhead, and the number of faulty chunks they can detect and/or correct.

"…we use the existing data devices for keeping error correction codes."

"…we propose stealing memory and devoting it to RAS purposes. Our technique is implemented at the memory controller level…"

"…works by using the two memory channels of a memory controller in lockstep mode…"

Given that GDDR5 does not provide extra devices or storage to accommodate error detection-correction codes, we use the existing data devices for keeping error correction codes. In order to do so, we propose stealing memory and devoting it to RAS purposes. Our technique is implemented at the memory controller level (this does not require cooperation among memory controllers) and is transparent to the OS. Also, it works by using the two memory channels of a memory controller in lockstep mode, hence 8 x16 devices are available for SDDC purposes. We propose using regular x16 devices to store the error correction codes. As an implementation example and because of its low overhead, we show how to obtain SDDC support with 2-redundant b-adjacent

Bossen's codes.[7] However, other codes with similar overheads could also be used in a similar fashion.[8][9] Specifically, 2-redundant b-adjacent Bossen's codes allow detecting and correcting a maximum of b-adjacent errors of a data word with a maximum length of $b \times b$ bits and an overhead of $2b$ bits. We propose a new way to map data and codes to devices that minimizes the amount of sequestered memory. Moreover, we propose several enhancements to deal with the performance problems that this data layout induces.

Data Mapping to x16 Devices

For GDDR5 memory, a device is connected to 16 data input/output pins (that is, a x16 device), and hence Bossen's codes would allow the protection of a maximum of 256 data bits with 32 bits of overhead ($b = 16$), as long as each one of the sixteen chunks of 16 bits (256 bits) is stored in a different x16 device. Otherwise, if we were to store 256 bits in less than 16 devices, it would imply that at least one device would be providing 32 bits or more of the word. In case that device was faulty, we could not correct the data word when using an adjacent error detection/correction code for $b = 16$.

However, two lockstepped channels cannot form a word of 256 bits, because there are 8 available x16 devices when working in clamshell mode. Our intra-memory controller SDDC technique is based on protecting 96 bits of data with 32 bits of code. This is done by using 25 percent of the memory to store the codes. Regarding recovery, our technique allows recovering 1 out of 8 failing devices, whereas DDDC can recover 2 nonadjacent devices out of 36, when using x4 DDR3 DIMMs.

"…our technique allows recovering 1 out of 8 failing devices, whereas DDDC can recover 2 nonadjacent devices out of 36, when using x4 DDR3 DIMMs."

Figure 3 shows an example on how cache lines can be stored across the different eight devices in the two memory channels. For clarification purposes, Figure 2 depicts the different devices and their names connected to a memory controller through two channels working in clamshell mode. However, it would be possible to assign the chunks of data and codes to the existing devices in different ways. From now on, we will assume the layout depicted in Figure 2.

Cache line data (512 bits long) is scattered across six x16 devices and across the two memory channels. Two devices are devoted to store the codes. It is interesting to note that contiguous cache lines start at different devices, and the pattern repeats after three cache lines.

"Cache line data (512 bits long) is scattered across six x16 devices and across the two memory channels."

"Two devices are devoted to store the codes. It is interesting to note that contiguous cache lines start at different devices, and the pattern repeats after three cache lines."

In order to access a cache line, two bursts of eight accesses are performed. The two consecutive bursts allow activating the four different devices in every channel. A simple way to achieve this is to use the most significant bit of the column address to select the device within a GDDR5 module. Therefore, two consecutive bursts will have the most significant bit with opposite values. The memory controller determines the device where a cache line begins, the starting column, and what columns to skip. Given that the cache lines layout is regular and repeats every three cache lines, the memory controller just needs to perform fixed modulo 3 operations.

LAYOUT OF DATA AND CODES ACROSS DEVICES
32 blocks of 16 bits are 512 bits (cache line size)

	CHANNEL 1				CHANNEL 2			
	D1	D3	D2	D4	D5	D7	D6	D8
0	1	2	3	4	5	6	C1	C1'
1	7	8	9	10	11	12	C2	C2'
2	13	14	15	16	17	18	C3	C3'
3	19	20	21	22	23	24	C4	C4'
4	25	26	27	28	29	30	C5	C5'
5	31	32	33	34	35	36	C6	C6'
6	37	38	39	40	41	42	C7	C7'
7	43	44	45	46	47	48	C8	C8'
8	49	50	51	52	53	54	C9	C9'
9	55	56	57	58	59	60	C10	C10'
10	61	62	63	64	65	66	C11	C11'
11	67	68	69	70	71	72	C12	C12'
12	73	74	75	76	77	78	C13	C13'
13	79	80	81	82	83	84	C14	C14'
14	85	86	87	88	89	90	C15	C15'
15	91	92	93	94	95	96	C16	C16'

Figure 3: Example of data and codes layout. Three cache lines are shown
(Source: Intel Corporation, 2012)

Figure 4 shows an access example where the yellow cache line is accessed. The two access bursts allow providing 1024 bits of data. Of these, 512 bits of data correspond to the cache line, the excess data corresponds to a partially prefetched cache line (half of the blue one in the example) and the codes of the partially prefetched cache line.

Example: C1.C1' protects chunks 1.2.3.4.5.6
C2.C2' protects chunks 7.8.9.10.11.12

CHANNEL 1				CHANNEL 2				time
D1	D3	D2	D4	D5	D7	D6	D8	
1	2			5	6			BURST 1
7	8			11	12			BURST 1
13	14			17	18			BURST 1
19	20			23	24			BURST 1
25	26			29	30			BURST 1
31	32			35	36			BURST 1
37	38			41	42			BURST 1
43	44			47	48			BURST 1
		3	4			C1	C1'	BURST 2
		9	10			C2	C2'	BURST 2
		15	16			C3	C3'	BURST 2
		21	22			C4	C4'	BURST 2
		27	28			C5	C5'	BURST 2
		33	34			C6	C6'	BURST 2
		39	40			C7	C7'	BURST 2
		45	46			C8	C8'	BURST 2

Figure 4: Accesses to obtain the yellow line. Partial prefetch of the blue cache line is achieved
(Source: Intel Corporation, 2012)

Enhancements

In order to avoid wasting memory space, cache lines start at different device boundaries, and as a consequence, there are column addresses that can store segments of two different cache lines (column 5 and 10 in Figure 3 are two examples). For the blue cache line in Figure 5, chunks 31 and 32 do not constitute a part of the blue cache line, but these fragments must be read and written along with the blue cache line, since they are involved in the code C6 and C6'. We refer to these two data chunks as the *preamble* of the blue cache line. Similarly, the same happens for chunks 65 and 66, and we refer to these fragments as the *postamble* of the blue cache line. Despite the fact that preambles and postambles are read/written upon the blue cache line request, unless no enhancement is introduced at the memory controller they do not represent useful data for the request being serviced.

Similarly, these columns (preambles/postambles) are problematic during cache line writes because without extra optimizations, it would be necessary to read the cache line again to obtain the preamble and postamble data (see Figure 5) so that the codes could be computed. This implies that a total of the request would be satisfied after three bursts (one parallel burst across the two channels to obtain the preamble and postamble, and two parallel bursts across the two channels to accomplish the write request), incurring important performance overhead on writes.

Preamble and postamble used to optimize writes. If preamble and postamble were not cached, we would have to read the line L again in order to determine 31.32 and 65.66 because they are part of codes C6.C6' and C11.C11'. 31.32 and 65.66 etc would not be modified during writes (using write masks)

Preamble is 31.31
Line L is 33 … 64 **Postamble is 65.66**

31	32	33	34	35	36	C6	C6'
37	38	39	40	41	42	C7	C7'
43	44	45	46	47	48	C8	C8'
49	50	51	52	53	54	C9	C9'
55	56	57	58	59	60	C10	C10'
61	62	63	64	65	66	C11	C11'
67	68	69	70	71	72	C12	C12'
73	74	75	76	77	78	C13	C13'

Figure 5: Preamble and postamble example
(Source: Intel Corporation, 2012)

Caching and Prefetching Optimizations

As a way to speed up writes, we "cache" in the memory controller the preamble and postamble (if any) as well as the cache line address of the last read cache line(s). If a cache line write matches the address of the last read line(s), there is no need to perform a read to obtain the preamble and/or postamble to compute the codes for overlapping columns. This can be generalized to "cache" any number of preambles/postambles. Also, since GDDR5 supports write masks, there is no need to write again the data that has not been touched. Every cache line has a maximum of 64 bits for preamble and postamble, hence introducing a low overhead. Since a GDDR5 page is 2 KB long, and we use 4 GDDR5 modules in a lockstepped and clamshell manner upon a cache request, it means

"If a cache line write matches the address of the last read line(s), there is no need to perform a read to obtain the preamble and/or postamble to compute the codes for overlapping columns."

that there are globally 96 cache lines (data) in all the row buffers of the eight devices (for a single bank). The cache lines layout follows a repetitive pattern, which indicates that in order to store *all* the preambles/postambles in a page of a bank, roughly 768 bytes of data are needed plus 96 × 2 bits (24 bytes) to indicate whether the preamble/postamble data is valid (has been cached). To implement this feature, a direct access memory is the simplest option: the cache line ID is used to access one of the 96 entries, where each entry holding 8 bytes for preamble/postamble. Upon a cache line write request, first this information is obtained in order to compute the codes for the columns holding the preamble and postamble. Just in case neither the preamble nor the postamble is available in the memory, an additional read request will have to be performed. After that, and in parallel to the cache line writing, a maximum of two entries (adjacent cache lines) are accessed to update their information. However, since four cycles are needed to perform the writing of a cache line, enough timing is provided for the update, as well as for previous preamble/postamble query.

"Regarding the excess data obtained during a cache line access, we manage this data as prefetched data."

"…this prefetched data is kept at the memory controller level so that subsequent read requests arriving to the memory controller check if part of the information to be retrieved is already "cached.""

Regarding the excess data obtained during a cache line access, we manage this data as prefetched data. To improve performance, this prefetched data is kept at the memory controller level so that subsequent read requests arriving to the memory controller check if part of the information to be retrieved is already "cached." This cached data allows advancing the column address to be read and as a consequence, prefetching more data. Overall, sequential (forward or backward) reads of three cache lines can be satisfied with an average cost of four groups of bursts across two channels. As an example, Figure 6 shows a forward read access of the yellow, blue, and green cache lines. The first request, as previously commented, returns the yellow cache line and prefetches half of the blue cache line. Afterwards, when there is a request for the blue cache line, the column address will be moved to column 8 and the whole green cache line will be prefetched in the memory controller. This allows that the future request for the green line can be satisfied with no access to the GDDR5 modules.

To implement that, whenever a cache line is being read from the main memory, the memory controller groups together the data and code chunks that come from the same column address and identifies whether they correspond to the cache line being requested, it is a column where two different lines reside, or it is a column from another different line. This can be easily inferred because the cache lines layout follows a repetitive pattern. Once a read has been completed, the above-mentioned preamble and/or postamble "direct access memory" is filled. As commented, the entire last read data (potentially for every bank) is kept in a buffer in the memory controller along with the starting column address.

For the next read request the memory controller will determine the number of overlapping column addresses that overlap with the previous cached information. If there is some overlap, the read column address can be moved forward (or backward) as many columns as the number of overlapping columns with respect to the previous cached information. The new read data belonging to the requested cache line is merged with the data coming from the last request (the one cached) and returned to the requesting agent. The same caching process starts again.

	CHANNEL 1				CHANNEL 2				
	D1	D3	D2	D4	D5	D7	D6	D8	
0	1	2			5	6			BURST 1
1	7	8			11	12			
2	13	14			17	18			
3	19	20			23	24			
4	25	26			29	30			
5	31	32			35	36			
6	37	38			41	42			
7	43	44			47	48			
0			3	4			C1	C1'	BURST 2
1			9	10			C2	C2'	
2			15	16			C3	C3'	
3			21	22			C4	C4'	
4			27	28			C5	C5'	
5			33	34			C6	C6'	
6			39	40			C7	C7'	
7			45	46			C8	C8'	
8	49	50			53	54			BURST 3
9	55	56			59	60			
10	61	62			65	66			
11	67	68			71	72			
12	73	74			77	78			
13	79	80			83	84			
14	85	86			89	90			
15	91	92			95	96			
8			51	52			C9	C9'	BURST 4
9			57	58			C10	C10'	
10			63	64			C11	C11'	
11			69	70			C12	C12'	
12			75	76			C13	C13'	
13			81	82			C14	C14'	
14			87	88			C15	C15'	
15			93	94			C16	C16'	

Green cache line is fully cached in the memory controller. 0 bursts are needed

Figure 6: Example of cost amortization
(Source: Intel Corporation, 2012)

Other Implementations

Several implementations of the proposed idea are possible. As commented, the assignment of data chunks to devices is arbitrary. Also, other kinds of adjacent error detection/correction codes with different amounts of overhead are also possible.

On another axis, in order to avoid overlapping cache lines, an alternative solution may simply waste the extra space at the end of a cache line. This simplifies the write method, because there is no need to keep preambles or postambles, but it comes at a cost in extra space (33.33 percent or 16/48 of the memory is lost). Also, the prefetching optimization is for performance purposes so designers may want to drop this feature for simplicity and less hardware overhead.

Applications to Other Memory Technologies

The same idea can be applied to other types of memory technologies where devices are grouped to feed a wider data bus. For example, common DDR, DDR2, and DDR3 modules (as well as other future memory technologies)

"The same idea can be applied to other types of memory technologies where devices are grouped to feed a wider data bus. For example, common DDR, DDR2, and DDR3 modules…"

feed a bus of 64 data bits. Normally, this modules come prepackaged in the form of four x16 devices, eight x8 devices, or sixteen x4 devices.

Table 1 briefly lists possible applications of the proposed implementation to this kind of memory modules. Notice that the list of possible implementations is not exhaustive.

As an example, the very same idea presented for GDDR5 can be applied to x4 DIMMs that have no RAS support (no extra devices to store ECCs), and output 64 bits per each access of every burst. We can even apply the technique to achieve SDDC support by using just one DIMM and avoiding lockstepped channels. This can be achieved with the same overhead of 25 percent of sequestered memory by using a 2-redundant b-adjacent Bossen code for $b = 8$, or by using a 4 check symbol code[8] for $b = 4$. Specifically,

Device Size	#DIMMs	Error Detection Code	Area Cost	Access granularity (bits)
x4	1	2-redundant b-adjacent Bossen code ($b = 8$) *or* 4 check symbol code ($b = 4$)	4/16	768 (burst of 4) 1024 (burst of 8)
	2	4 check symbol code ($b = 4$)	4/32	1024 (burst of 4) 1024 (burst of 8)
	4	4 check symbol code (b = 4)	4/64	1024 (burst of 4) 2048 (burst of 8)
x8	1	2-redundant b-adjacent Bossen code ($b = 8$)	2/8	768 (burst of 4) 1024 (burst of 8)
	2	4 check symbol code ($b = 8$)	4/16	1024 (burst of 4) 1024 (burst of 8)
	4	4 check symbol code (b = 8)	4/32	1024 (burst of 4) 2048 (burst of 8)
	$n \leq 1024$, n power of 2	4 check symbol code ($b = 8$)	$4/8n$	$n \times 4 \times 64$ (burst of 4) $n \times 8 \times 64$ (burst of 8)
x16	2	2-redundant b-adjacent Bossen code ($b = 16$)	2/8	1024 (burst of 4) 1024 (burst of 8)
	4	2-redundant b-adjacent Bossen code ($b = 16$)	2/16	1024 (burst of 4) 2048 (burst of 8)
	4	4 check symbol code ($b = 16$)	4/16	1024 (burst of 4) 2048 (burst of 8)
	$n \leq (64 \times 1024 \times 1024) - 1$, n power of 2	4 check symbol code ($b = 16$)	$4/4n$	$n \times 4 \times 64$ (burst of 4) $n \times 8 \times 64$ (burst of 8)

Table 1: Possible SDDC implementations on RAS-less memory technologies with a 64-bit bus
(Source: Intel Corporation, 2012)

4 check symbol codes allow protecting $2^{2b} + 1 - 4b$ bits against b-adjacent faults with $4b$ bits of code.[8] This means that for $b = 4$, $b = 8$, a maximum word of 241, and 65505 bits can be protected with an overhead of 16 and 32 bits, respectively.

For the x4 case where one DIMM is used, in order to obtain a cache line (given a burst size of 8), two consecutive accesses are necessary. This allows obtaining a cache line and prefetching half of another one. The cache lines layout pattern stays unchanged with respect to the GDDR5 SDDC implementation (so MC address determination is simple). The same happens for the x8 implementation using one DIMM.

Since only one bank is accessed and only one DIMM is required to provide the requested cache lines, several benefits can be obtained from this lockstep-free implementation. First, the number of devices activated is reduced to the bare minimum of 16 chips for a x4 DIMM. A reduction in the active power consumption can also be achieved because fewer devices are activated to satisfy a cache line request. Regarding performance, despite an access to a cache line is served (in the worst case when there is no previous prefetching) using an additional burst (hence, increasing the read latency), performance can also be boosted because there is an increase in bank-level and rank-level parallelism (no interlocked channels/ranks/banks). Given the expected low row locality for future multi-threaded tera-scale processors[11], a SDDC solution using a single memory module is more desirable in terms of power and performance.

Summary

We have presented a memory controller level solution to provide SDDC support for memory technologies with no RAS support. The technique can be applied to several types of memory, such as GDDR5 or common memory with no special devices for ECC. Our proposal allows recovering failures in one out of eight devices with a memory capacity cost of 25 percent.

"We have presented a memory controller level solution to provide SDDC support for memory technologies with no RAS support. The technique can be applied to several types of memory, such as GDDR5 or common memory with no special devices for ECC. Our proposal allows recovering failures in one out of eight devices with a memory capacity cost of 25 percent."

Additionally, we have shown that this SDDC technique can be implemented for memory technologies feeding a wider data bus (such as DDR3, not GDDR5) without the use of lockstepped channels (see Table 1).

It is also worth noting that this way of implementing SDDC opens the possibility to offer high reliability with cheap DIMM devices.

References

[1] JEDEC Solid State Technology Association, "GDDR5 SGRAM," 2009.

[2] Intel Corporation, "Intel Xeon Processor E7 Family: Reliability, Availability and Serviceability," White Paper

[3] Yoon D.H, Erez M., "Virtualized and Flexible ECC for Main Memory," Architectural Support for Programming Languages and Operating Systems Conference (ASPLOS), 2010.

[4] Sun Microsystems Inc, OpenSPARC T2 System on Chip (SOC) Microarchitecture Specification, May 2008.

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

[6] IBM Blue Gene Team, "Overview of the IBM Blue Gene/P Project," IBM Journal of Research and Development 2008 Bossen D.C., "b-Adjacent Error Correction," IBM Journal of Research and Development, 1970.

[7] Bossen D.C., "b-Adjacent Error Correction," IBM Journal of Research and Development, 1970.

[8] Chen, C.L. "Symbol error correcting codes for memory applications," International Symposium on Fault-Tolerant Computing (FTCS), 1996.

[9] Chen C.L., Hsiao M.Y., "Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review," IBM Journal of Research and Development, 1984.

[10] Jacob B., Ng S.W, Wang D.T., "Memory Systems: Cache, DRAM, Disk," Morgan Kaufmann.

[11] Udipi A.N., Muralimanohar N., Chatterjee N., Balasubramonian R., Davis A., Jouppi N.P, "Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores," International Symposium on Computer Architecture (ISCA), 2010.

[12] Dell T.J. "A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory," IBM Report, 1997.

Author Biographies

Javier Carretero received an MS degree in Computer Engineering from the Universitat Politècnica de Catalunya (UPC) at Barcelona, Spain, in 2005. Since April 2006, he has been a research scientist at Intel Labs Barcelona. His main research interests include processor microarchitecture, hardware reliability, and lightweight on-line testing. He can be contacted at javier.carretero.casado@intel.com

Isaac Hernández is a silicon architecture engineer who has been with Intel for 10 years. He was the memory controller architect on KNF and Intel® Xeon Phi™ products. He is currently working on interconnect and coherence

protocols for future Intel® Many Integrated Core Architecture products. He can be contacted at isaac.l.hernandez@intel.com

Xavier Vera received an MS degree in computer science from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 2000, and a PhD from Mälardalens Högskola, Västerås, Sweden, in 2004. He has been with Intel since February 2004, participating in research in the area of reliable and variations-aware microarchitectures. He can be contacted at xavier.vera@intel.com

Toni Juan is a principal engineer with more than 10 years of experience at Intel and has been the uncore architect for KNF and Intel® Xeon Phi™. He is currently working in the uncore architecture of future Intel MIC generations. His contributions span from memory hierarchy, memory controllers, on-die interconnect, memory coherence and performance modeling. He can be contacted at toni.juan@intel.com

Enric Herrero received his MS in Electronics and PhD in Computer Science from the Universitat Politècnica de Catalunya (UPC), in Barcelona, and an MS in Electricity from the Royal Institute of Technology (KTH), in Stockholm, Sweden. He was a visiting researcher at the University of California San Diego (UCSD) in 2009. He now works as a research scientist at Intel. His research focuses on reliability and memory hierarchy. He can be contacted at enric.herrero@intel.com

Tanausú Ramírez has been a research scientist at Intel Labs Barcelona since December 2009. He received BS and MS degrees in Computer Science from the University of Las Palmas de Gran Canaria, Spain. In 2010 he received his PhD from the Universitat Politècnica de Catalunya, Barcelona. His current research interests include architectural optimizations for future processors, hardware reliability, and variations-aware microarchitectures. He can be reached at tanausu.ramirez@intel.com

Matteo Monchiero is a performance architect in the Intel® Xeon® Performance Architecture Group in Intel Santa Clara. Matteo joined Intel in 2010 in the Intel Barcelona Research Lab where he worked on several topics in post-silicon validation and reliability. He is coauthor of more than thirty international publications and three granted patents. He can be contacted at matteo.monchiero@intel.com

Antonio González holds a PhD in Computer Engineering from the Universitat Politècnica de Catalunya (UPC), in Barcelona. He is the founding director of Intel Labs Barcelona, started in 2002, whose research focuses on computer architecture. Prior to his work at Intel, he joined the faculty of the Computer Architecture Department of UPC in 1986, and became a full professor in 2002. Antonio holds over 40 patents, has published over 300 research papers and has given over 100 invited talks in the areas of computer architecture and compilers. He has also made significant contributions to the design of the architecture of several Intel processors. Antonio has served as program chair for ISCA, MICRO, HPCA, ICS, and ISPASS among

other symposia and has been associate editor of the IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, IEEE Computer Architecture Letters, ACM Transactions on Architecture and Code Optimization, and Journal of Embedded Computing. He can be contacted at antonio.gonzalez@intel.com

Nicholas Axelos joined Intel in 2010 and works as a research scientist at Intel Barcelona Research Center. His interests include computer architecture, reliability estimates, fault tolerance, and heterogeneous architectures. He can be contacted at nicholasx.axelos@intel.com

Daniel Sánchez received his MSc and PhD degrees in computer architecture in 2007 and 2011, respectively, from the University of Murcia (Spain). He now works as a research scientist at Intel Barcelona Research Center, which he joined in 2011. His research interests include general computer architecture, fault tolerance and reliability, chip multiprocessors, and performance simulation. He can be contacted at danielx.sanchez@intel.com

TOWARDS PROPORTIONAL MEMORY SYSTEMS

Contributors

Doe Hyun Yoon
HP Labs

Min Kyu Jeong
Oracle Labs

Michael Sullivan
The University of Texas at Austin

Mattan Erez
The University of Texas at Austin

Off-chip memory systems are currently designed to present a uniform interface to the processor. Applications and systems, however, have dynamic and heterogeneous requirements in terms of reliability and access granularity because of complex usage scenarios and differing spatial locality. We argue that memory systems should be proportional, in that the data transferred and overhead of error protection be proportional to the requirements and characteristics of the running processes. We describe two techniques and specific designs for achieving aspects of proportionality in the main memory system. The first, dynamic and adaptive granularity, utilizes conventional DRAM chips with minor DIMM modifications (along with new control and prediction mechanisms) to access memory with either fine or coarse granularity depending on observed spatial locality. The second, virtualized ECC, is a software/hardware collaborative technique that enables flexible, dynamic, and adaptive tuning between the level of protection provided for a virtual memory page and the overhead of bandwidth and capacity for achieving protection. Both mechanisms have a small hardware overhead, can be disabled to match the baseline, and provide significant benefits when in use.

Introduction

With increasing levels of system integration, the need to improve power and energy efficiency is paramount. With voltage scaling leveling off, process technology alone cannot provide the necessary sustained improvements in efficiency. An attractive alternative to improving performance is to waste less power when resources are not fully utilized. This idea of proportionality, where the resources consumed are proportional to actual requirements, is thus key to achieving necessary efficiency, and hence, necessary performance improvements. In this article we argue the importance of proportionality in the memory system and describe two mechanisms that enable it. The discussed research was conducted at the University of Texas at Austin with support from the Intel URO Memory Hierarchy Program.

"...proportionality, where the resources consumed are proportional to actual requirements, is key to achieving necessary efficiency, and hence, necessary performance improvements."

The memory system continues to consume significant power and energy resources as available computation increasingly outpaces memory bandwidth. At the same time, the need for increasing memory capacity and the inherent error sensitivity of memory devices require hardware error protection for high reliability. Existing systems attempt to manage these bandwidth and reliability issues by using coarse-grained (CG) memory accesses and by applying error checking and correcting (ECC) codes uniformly across all memory locations. Large granularity accesses reduce cache miss rates and amortize control for spatially local requests and can thus maximize peak bandwidth. Coarse-grained ECC codes provide low-redundancy error tolerance enabling strong protection with acceptable overhead.

While the coarse-grained approach can maximize peak bandwidth with required reliability levels, it is ill-suited for many applications and scenarios. The optimal memory access granularity and error detection and correction schemes are, by nature, application specific. When a program lacks spatial locality, CG accesses waste power, memory bandwidth, and on-chip storage. Furthermore, protecting all data to the same degree carries either a large associated cost, forces CG accesses, or carries a hidden risk of silent data corruption. This one-size-fits-all approach is inherently not proportional, in that it does not account for specific needs and usage. Thus, CG accesses waste the scarcest resources: off-chip (or off-package) bandwidth, power and power. As a result, it is unlikely that uniform granularity and protection will provide adequate system efficiency and reliability for future workloads and system scales.

In contrast to uniform schemes, we advocate a cooperative and flexible memory system that transfers data and protects it from error in a manner that is proportional to the requirements and characteristics of the running processes. This work describes two techniques for achieving proportionality in the main memory system. Our techniques are unique in that they do not target low-utilization phases or background power. Instead, by taking into account application-specific memory access properties, these techniques provide scalable, efficient, and reliable operation for a wide range of demanding applications and operating environments. While adding significant flexibility and enabling new levels of performance and efficiency, our proposed mechanisms have only a small hardware overhead and may provide benefits even without the support of the application developer.

"...we advocate a cooperative and flexible memory system that transfers data and protects it from error in a manner that is proportional to the requirements and characteristics of the running processes."

In the rest of this article we explain the importance of proportionality in the memory system, describe our two proportional mechanisms, and discuss their potential future applications and impact. The first of the two mechanisms, the dynamic granularity memory system (DGMS) enables memory to be accessed with granularity that can vary dynamically to maximize utilization of off-chip links, performance, and efficiency. In our experiments, DGMS is able to improve performance by up to 180 percent in a bandwidth-constrained environment and power efficiency by an even higher 280 percent over a CG-only system. The second mechanism, virtualized ECC (VECC), decouples ECC information from the data it protects and enables ECC schemes that are dynamic, adaptive, and proportional. With VECC, each memory page can receive its own ECC scheme from an available palette, thus eliminating the waste associated with a uniform error code. Furthermore, VECC relaxes many design constraints by storing ECC information as data. We show that this design flexibility offers significant efficiency advantages—memory protection can be increased with a stronger ECC code while at the same time improving system energy-delay product over a system with conventional uniform error codes.

The Need for Proportional Memory Systems

While proportionality is an appealing approach to improve efficiency, and hence, performance, much of the work on proportionality has been on the processor side. Advanced power management features, such as flexible and

hardware-managed DVFS and both coarse- and fine-grained clock and power gating minimize wasted power when the full compute capabilities are not needed.[6][17] The memory system, despite consuming a large and (in servers at least) growing fraction of overall consumed power, has received less attention with respect to proportional mechanisms. Similar to compute-oriented techniques, the little memory proportionality research that exists focuses on times when the memory system is not fully utilized. At these times, devices can be placed in a low power mode, DVFS can be applied to links and potentially even memory arrays, and refresh periods can be extended.

"When a program lacks spatial locality, CG accesses waste power, memory bandwidth, and on-chip storage."

We argue that the importance of main memory provides sufficient motivation to develop fine-grained proportional mechanisms that can be used not just when utilization is low. Specifically, new mechanisms are needed that minimize unnecessary data transfers and wasted storage resources. As explained earlier, memory systems currently favor coarse-grained accesses to amortize control and uniform ECC redundancy, thus maximizing peak bandwidth when spatial locality is high. When a program lacks spatial locality, however, CG accesses waste power, memory bandwidth, and on-chip storage. Uniform ECC wastes both capacity and bandwidth when protection needs are variable. We explain the need for proportionality of granularity and ECC in the subsections below.

Accesses Granularity

Figure 1 shows the spatial locality of various benchmarks by profiling the number of 8-byte words accessed in each 64-byte cache line before the line is evicted (in a 1-MB cache). Most applications touch less than 50 percent of each cache line. For these applications, a CG-only memory system wastes off-chip bandwidth and power by fetching unused data. A memory system that makes only a fine-grained (FG) access eliminates this minimum-granularity problem and may achieve higher system throughput than a CG-only memory system. An FG-only memory system, however, incurs high ECC (error checking and correcting) overhead since every FG data block needs its own ECC. High-end vector processors (such as Cray's Black Widow[2]) often use the FG-only approach but relinquish the benefits of CG accesses when spatial locality is high.

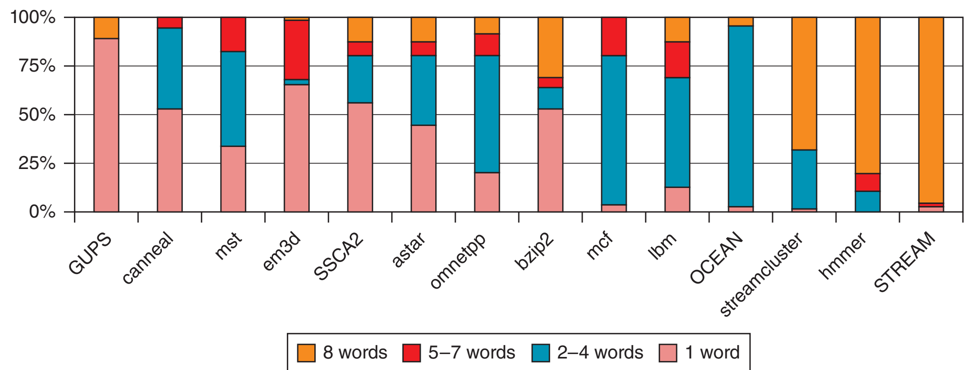


Figure 1: Number of touched 8-byte words in a 64-byte cache line before the line is evicted (Source: University of Texas at Austin, 2011)

Unlike CG and FG memory systems, our dynamic-granularity DGMS is proportional. It amortizes overhead and uses CG accesses when spatial locality is high, while enabling FG accesses when spatial locality is lacking. In the section "Dynamic and Adaptive Granularity," we describe two variants of this system. The first relies on software to adapt granularity and the second is a pure-hardware microarchitectural approach.

Memory Error Protection

The traditional approach for memory error protection applies error checking and correcting (ECC) codes uniformly across all memory locations, potentially resulting in system inefficiency or at the risk of data corruption. With uniform ECC, each memory access is extended to also include redundant information that is used to detect and correct potential errors. Memory protection is necessary because memory capacity requirements dictate both denser and more memory chips. Unfortunately, this increase in capacity comes with a corresponding increase in memory failures.

Currently, uniform ECC is typically implemented by adding additional memory chips to each memory rank or DIMM. ECC DIMMs are most often used to provide single-bit error correction and double-bit error detection (SEC-DED) for each DRAM rank, which does not reduce memory system performance. Tolerating failures of entire DRAM chips requires the use of Chipkill correct, which "spreads" a DRAM access across multiple chips and uses a wide ECC to allow high error tolerance.[9] While this conveniently provides a fixed level of error tolerance, uniform ECC cannot efficiently provide the error tolerance levels that will be required in future computing platforms without a significant increase in cost. The reason is that the increasing granularity, DRAM interface width, and burst length necessitates either a corresponding increase in memory access granularity or an increase in redundancy. Increasing granularity is undesirable for the reasons mentioned earlier. Increasing redundancy may be even more costly because it requires both additional storage resources and additional I/O pins to communicate more redundant information with each access. For example, providing Chipkill with current DRAM packaging technology is best when x4 DRAMs are used, which have a 4-bit interface width. These narrow chips consume roughly 30 percent more energy for a given total DIMM capacity with respect to the more efficient x8 configurations.[5] This extra overhead is required for all the memory, which may be multiple terabytes in large-capacity systems.

To complicate and constrain designs even further, the error tolerance level, hence the cost of reliability, must be determined at design time based on a "worst-case" scenario of error propensity. This worst case may never arise in many systems, but may actually be exceeded in others. This is a poor combination that results in the overprovisioning of reliability techniques in most environments and a potential compromise to reliable operation in others. We present a proportional alternative to uniform ECC. Virtualized ECC is a general scheme for virtualizing memory error correction to provide design-time flexibility and runtime adaptivity. Virtualized ECC maps the redundant

"...the cost of reliability must be determined at design time based on a "worst-case" scenario of error propensity."

"This worst case results in the overprovisioning of reliability techniques in most environments and a potential compromise to reliable operation in others."

information needed to correct errors into the memory namespace itself. This mechanism enables flexible memory protection, as opposed to the fixed error tolerance level of uniform ECC. Furthermore, Virtualized ECC enables error correction mechanisms that adapt to user and system demands.

Dynamic and Adaptive Granularity

We describe two variants of DGMS that proportionally adapt memory access granularity. Both variants use a similar overall design and differ in how granularity information is obtained and communicated and in how data and ECC information are laid out across memory chips. An overview of the design is shown in Figure 2. DGMS uses sectored caches[14] with eight 8-byte subsectors to maintain FG information in the on-chip memory hierarchy, a modified memory controller that can schedule FG and CG requests, and a sub-ranked memory DIMM design that we leverage to enable low-cost independent access to each DRAM chip in a rank[3][29]. In a sub-ranked DIMM, a register or demux chip can decouple a rank of DRAM chips and direct memory command signals to only sub-ranks of one or more chips, while the datapath is still dedicated to each chip as in a regular memory module.

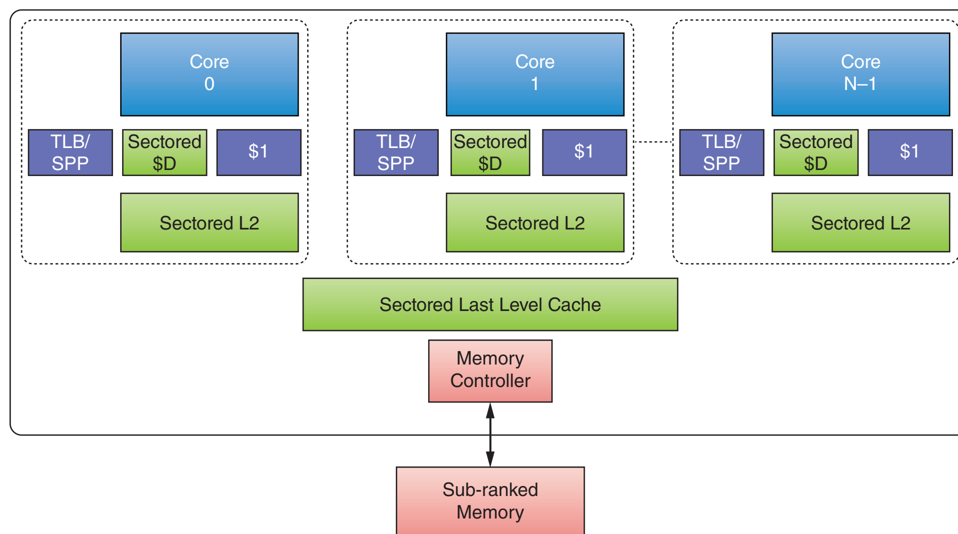


Figure 2: Overview of DGMS design, which uses sectored caches throughout the hierarchy, identifies FG/CG accesses either with information communicated from software through the TLB or with a spatial pattern predictor (SPP), and uses a modified memory controller that is aware of mixed granularity and controls a sub-ranked memory module (Source: University of Texas at Austin, 2012)

"AGMS enables the processor to selectively use FG accesses only when software determines FG to be beneficial and still maintains the efficiency of CG accesses by default."

Our first variant, the adaptive granularity memory system (AGMS)[27] relies on software to communicate a static granularity for each memory page. AGMS enables the processor to selectively use FG accesses only when software determines FG to be beneficial and still maintains the efficiency of CG accesses by default. The information for determining whether an access is FG or CG is provided at page granularity and is communicated by the operating system (OS)

through the page table and is stored in the TLB. With AGMS, each memory page is assigned an access granularity when it is allocated. An important reason for using this static approach is to enable different layouts of data and ECC information that are better tuned for CG and FG accesses. Figure 3 compares the data layouts for CG and FG pages. An FG access can achieve high throughput when spatial locality is low, but pays with increased ECC overheads.

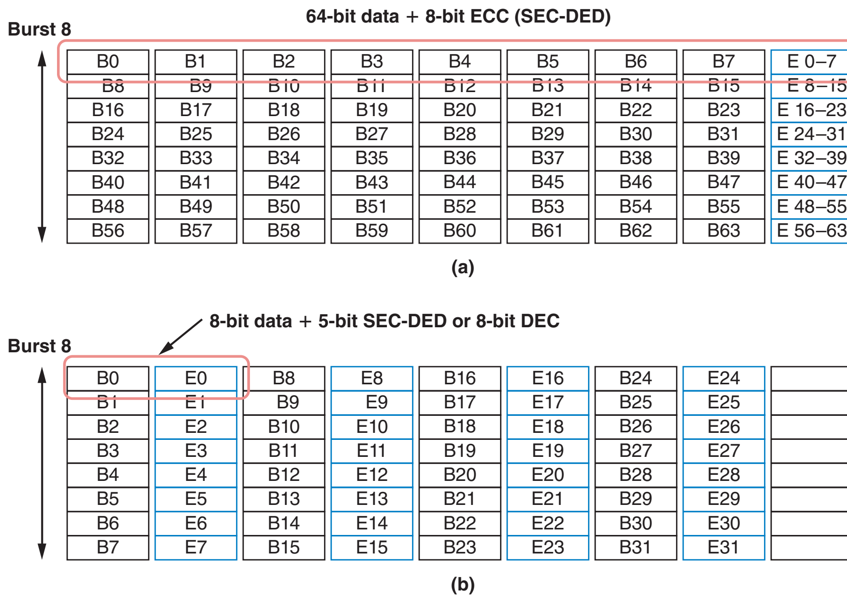


Figure 3: CG and FG accesses in AGMS. (a) Coarse-grained: Bx represents the xth byte in a 64-byte block, and Ey-z is 8-bit SEC-DED ECC for data By to Bz. (b) Fine-grained: Bx represents the xth byte in a 64-byte block, and Ex is 8-bit SEC-DED ECC for data Bx
(Source: University of Texas at Austin, 2012)

Due to the different data and ECC layouts for CG and FG, AGMS requires changes to (and collaboration between) all system levels, from the memory system to user-space applications: the application dictates the preferred granularity during memory allocation; the OS manages per-page access granularity by augmenting the virtual memory interface; a sector cache manages fine-grained data in the cache hierarchy; and a sub-ranked memory system and mixed granularity memory scheduling handle multiple access granularities within the off-chip memory system.

Our second variant extends AGMS with dynamic mechanisms that offer numerous and substantial benefits. We refer to the resulting system as the Dynamic Granularity Memory System (DGMS).[28] DGMS supports both CG and FG accesses to a single, uniform memory space. Eliminating the strict separation of CG and FG data pages enables true dynamic adaptivity and has the potential to simplify the implementation of an AGMS system.

"DGMS supports both CG and FG accesses to a single, uniform memory space. Eliminating the strict separation of CG and FG data pages enables true dynamic adaptivity and has the potential to simplify the implementation of an AGMS system."

The data layout of DGMS shares the same memory, including ECC, between CG and FG accesses. This allows FG accesses to benefit from the same low-redundancy

error tolerance as CG accesses, eliminating the 100-percent FG ECC overhead required for the original AGMS design. This reduction in error-protection overhead affects the capacity, bandwidth, and power efficiency of FG accesses.

We encode the data within each 64-byte data chunk such that each 8-bit SEC-DED ECC protects the eight bytes transmitted out of a single DRAM chip over all bursts. The eight bytes of DGMS ECC protect the full 64-byte data chunk with the same redundancy as the conventional CG-only system. Since each 8-bit SEC-DED ECC protects an independent DRAM chip, the layout supports both CG and FG accesses. Figure 4(a) illustrates how an FG request is serviced with the proposed data layout. To avoid contention in the ECC DRAM chip, we spread ECC blocks across sub-ranks in a uniform, deterministic fashion, as shown in Figure 4(b), similar to RAID-5.

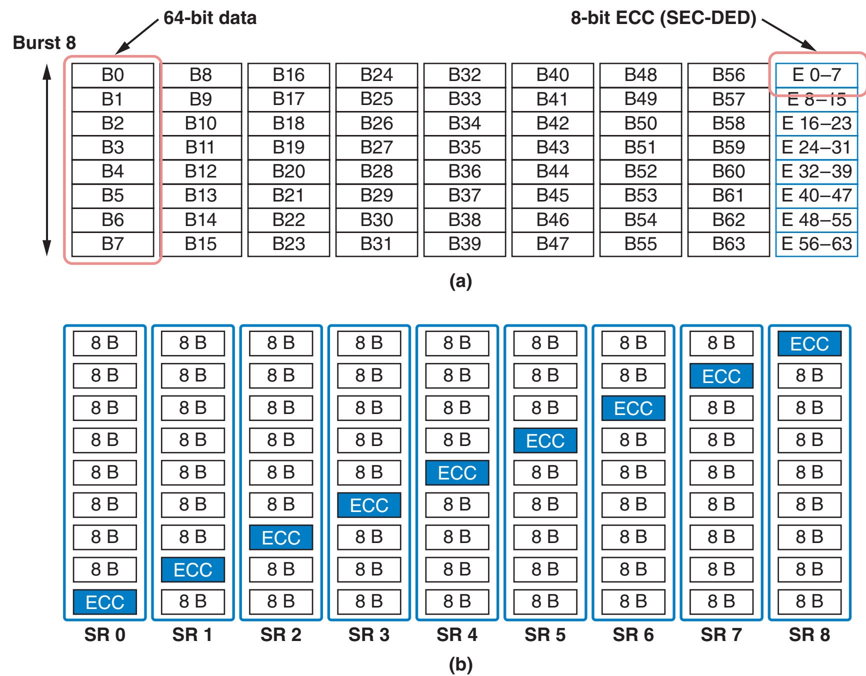


Figure 4: The data layout used by DGMS to support multiple access granularities and the method used to lessen bank conflicts in the ECC DRAM chip. (a) Proposed data layout: Bx represents the xth byte in a 64-byte block, and Ey-z is 8-bit SEC-DED ECC for data By to Bz. (b) Spreading ECC locations in sub-ranks (SR: sub-rank)
(Source: University of Texas at Austin, 2012)

"...the layout of DGMS permits pages to service both CG and FG accesses, which enables the dynamic prediction of access granularities..."

Because the layout of DGMS permits pages to service both CG and FG accesses, it enables the dynamic prediction of access granularities and does not need predetermined per-page granularity information and complicated virtual memory mechanisms. Dynamic locality and granularity speculation allows DGMS to operate as a hardware-only solution, without application knowledge, operating system support, or the need for programmer intervention. DGMS modifies previously proposed spatial pattern predictors[7][11] to operate at the main memory interface of a multi-core CPU, and adds layers to account for

memory scheduling and the interactions between cores. This study shows dynamic granularity adjustment to be an effective method for improving performance and system efficiency. Hardware-only DGMS provides comparable performance to software-controlled AGMS and demonstrates superior DRAM traffic and power reductions. Our ISCA paper describes more details in hardware granularity prediction in the context of chip-multiprocessors.[28]

DGMS Evaluation

We use cycle-based simulation and multi-programmed workloads to evaluate DGMS. The detailed evaluation methodology can be found in our ISCA paper.[28]

"...AGMS improves system throughput by 20–220 percent in many applications with low spatial loactily."

Figure 5 compares the system throughput of the CG baseline, AGMS, and DGMS, with and without ECC. AGMS improves the system throughput by 20–220 percent in many applications with low spatial locality. In general, the performance of DGMS is very close to that of AGMS, with the exception of some outliers. The performance of DGMS suffers slightly with respect to AGMS due to increased bank conflicts from accessing the randomly distributed ECC information—with ECC disabled, DGMS outperforms AGMS. AGMS avoids bank conflicts by using a separate memory layout for FG pages, but does so at the cost of memory capacity, increased memory traffic, and increased energy spent accessing memory. Note that DGMS, with the proposed unified data/ECC layout, also does not require any changes in the OS, the virtual memory interface, compiler, or application. DGMS is a hardware-only solution, yet it achieves almost the same advantages of AGMS with "expert" program annotations.[1] In addition, we believe a better granularity prediction and multigranularity memory scheduling can improve the performance of DGMS in the future.

"...DGMS is a hardware-only solution, yet it achieves almost the same advantages of AGMS, which require "expert" program annotation."

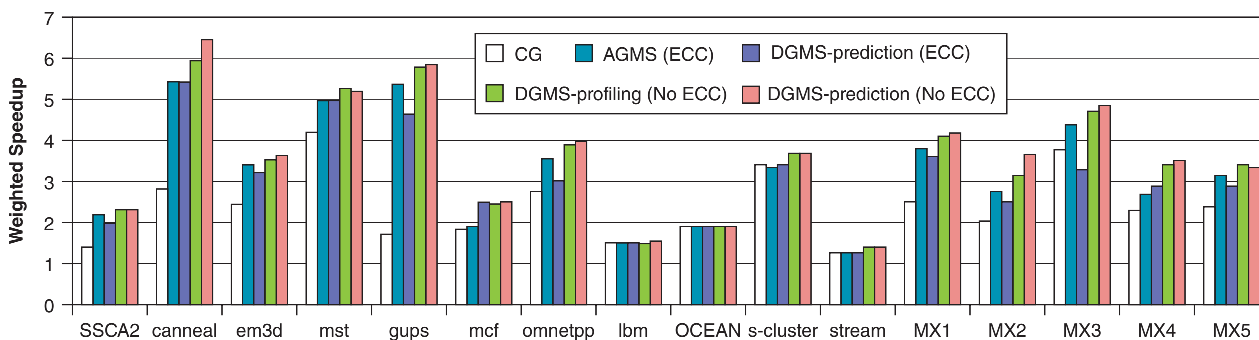


Figure 5: System throughput of AGMS and DGMS with and without ECC (Source: University of Texas at Austin, 2012)

Virtualized ECC

To achieve proportionality of memory protection overhead, we offer a fundamentally different approach to storing and manipulating redundant DRAM storage, which brings flexibility to memory error protection.

[1] Extracted from dynamic memory profiling information.

The Virtualized ECC architecture has four important advantages over uniform ECC. First, VECC[26] is a general OS/architecture mechanism for virtualizing DRAM ECC protection and decoupling the mapping of redundant information from the mapping of data. Second, with VECC we enable two-tiered protection for DRAM and show how to improve reliability guarantees and reduce the power consumption of the DRAM system by using wider-access configurations. Third, combining virtualization and multiple tiers, VECC can adapt error protection levels to match application, user, and system needs. Fourth, VECC can provide ECC protection for systems with standard non-ECC DIMMs without requiring changes to data mapping. We describe the VECC architecture and how it achieves these advantages below.

"Virtualized ECC can dynamically tune the memory protection scheme based on system configurations, environmental conditions, and application needs."

Virtualized ECC Architecture

The main innovation of Virtualized ECC is that it offers great flexibility in the method and level of memory protection. This flexibility allows Virtualized ECC to dynamically tune the memory protection scheme based on system configurations, environmental conditions, and application needs. In this way, Virtualized ECC can avoid the need to uniformly pay the overhead for the worst-case operating scenario, increasing system efficiency.

There are two basic mechanisms underlying Virtualized ECC: an augmented virtual memory (VM) interface that allows a separate virtual-to-physical mapping for data and for its associated redundant ECC information, and a generalization of DRAM ECC into a two-tiered protection mechanism, inspired by prior research.[18][24] Virtualized ECC uses a tier-one error code (T1EC) to detect errors on every access and a tier-two error code (T2EC) to correct the (rare) occurrence of an error.[25] Figure 6 compares a traditional VM ECC mapping with the decoupled two-tiered approach of Virtualized ECC. The traditional VM mapping, shown in Figure 6(a) translates a virtual address from the application name space to a physical address in DRAM. A DRAM access then retrieves or writes both the data and the ECC information, which is stored in alignment with the data in dedicated ECC DRAM chips.

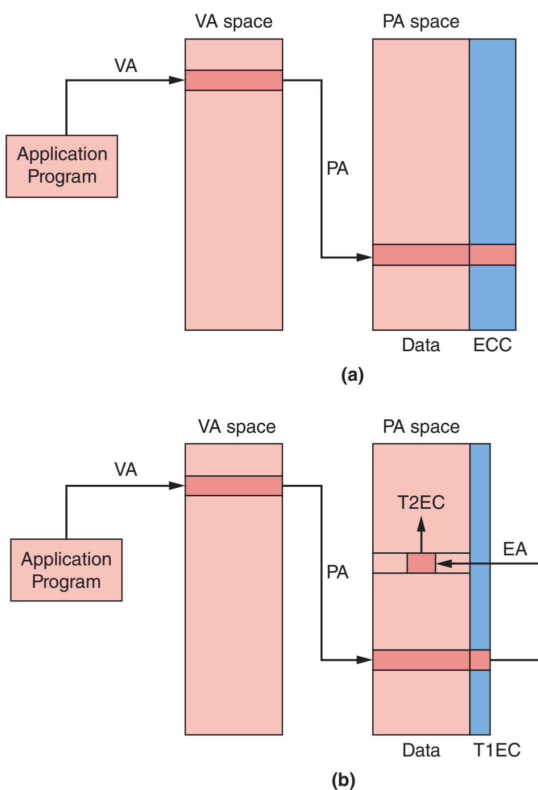


Figure 6: High-level view of memory accesses in a conventional virtual memory with fixed ECC, and Virtualized ECC with a two-tiered flexible protection scheme; (a) conventional architecture (b) Virtualized ECC architecture
(Source: University of Texas at Austin, 2010)

Figure 6(b) gives an example of a flexible mapping enabled by Virtualized ECC in which a portion of the redundant information, the T1EC, is aligned with the data, but the T2EC part is mapped to a different physical address.

The data and the T2EC share the same physical address space and storage devices, and the OS and hardware memory management unit ensure that data and ECC are always matched and up to date. Less total data is accessed on a read in Virtualized ECC than in the conventional approach because T2EC is only read on the very rare event of an error. Data writes, however, may have higher corresponding overhead because all ECC data needs to be updated, requiring a second DRAM access. To mitigate the potential degradation in performance, we utilize the processor cache to reduce the amount of ECC traffic and discuss this in detail in the following subsection. Another advantage of the decoupled mapping and two-tiered approach is that different memory pages can have different protection types. For example, clean pages do not require any T2EC storage, potentially increasing the effective memory capacity.

Cache-DRAM Interface

The cache filters requests from the core to DRAM and can also help improve the performance of Virtualized ECC. Because we store redundant information in the same physical namespace as data, we can cache ECC information on chip and can improve ECC access bandwidth using the same principles that make caches advantageous for data. Unlike application data, however, ECC is only accessed by the memory controller when it needs to address off-chip DRAM and is not shared among multiple processor cores. Thus, the redundant information is stored in the cache level to which the memory controller has direct access—the last-level cache (LLC) bank to which it is attached. Due to this arrangement, ECC information does not participate in any coherence protocol and is kept up to date by the memory controller. Virtualized ECC does not require significant changes to the existing cache interface; the only exceptions are the additional hardware for ECC address translation and the ability to maintain and write back partially valid cache lines. The latter is necessary because the cache has up-to-date ECC information only for data that is generated on chip (in two-tiered DRAM ECC). We describe this in detail below and address the different operations needed for two-tiered protection and for ECC with Non-ECC DIMMs.

Two-Tiered DRAM ECC

Figure 7 shows two-tiered Virtualized ECC on top of a generic memory system configuration with the LLC connected to two ranks of DRAM with dedicated ECC chips. We use the ECC chips to store T1EC, which can detect all errors of interest but cannot correct them without the additional T2EC information. The T2EC is mapped to the data DRAM chips such that data and its associated T2EC are in two different ranks.

"We use the ECC chips to store T1EC, which can detect all errors of interest but cannot correct them without the additional T2EC information. The T2EC is mapped to the data DRAM chips such that data and its associated T2EC are in two different ranks."

The numbers in the text below refer to operations shown in Figure 7. Handling a fill into the LLC on a cache miss follows the same operations as in a conventional system; a data burst and its aligned T1EC are fetched from main memory, and error detection is carried out (1). This differs from a conventional system only insomuch that any detected errors cannot be immediately corrected. Evicting a dirty line and writing it back to DRAM (2), however, requires additional operations when compared to a conventional memory hierarchy. The memory controller must update the T2EC information associated with the evicted line, which starts with translating the data address to the location of the ECC address (EA) (3 and 4). If the translated EA is already in the LLC, it is simply updated in place. Otherwise, we allocate an LLC line to hold the T2EC. We do not need to fetch any information from memory because T2EC is only read when an error is detected, and any writes render the prior information obsolete. Thus, we compute the new T2EC and write it into the LLC along with a mask that indicates what portion of the LLC line contains valid T2EC information. As we explain later, our coding schemes use T2ECs that are 16–128 bits long and thus require very few valid bits. Depending on the exact ECC used to protect the LLC itself, it may even be possible to repurpose the LLC ECC bits to store the valid mask. We can ignore errors in a T2EC line in the LLC because there is no need to add a third level

of redundancy and protect T2EC information from errors. This mask is used when a T2EC LLC line is evicted back to DRAM (5) as invalid portions of the line must not overwrite T2EC data in DRAM.

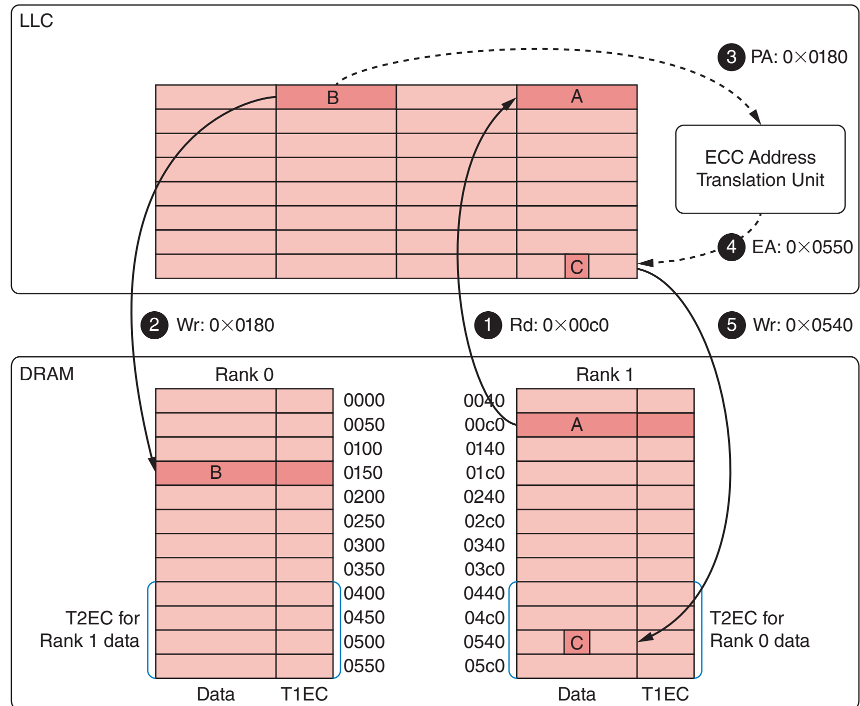


Figure 7: Operations of DRAM and LLC for accessing a two-tiered Virtualized ECC configuration
(Source: University of Austin, 2010)

When an error is detected by T1EC, correction is carried out using the corresponding T2EC. If the T2EC is not in the cache, correction requires an additional DRAM access to fetch the redundant information. This additional latency, however, does not significantly impact performance because errors in a particular memory channel are very rare. Frequent errors indicate a hard fault and can be mitigated by data migration, as suggested by Slayman.[19]

"Even if physical memory does not provide ECC storage, we can use Virtualized ECC to protect memory."

Virtualized ECC Interface with Non-ECC DIMMs

Even if physical memory does not provide ECC storage, we can use Virtualized ECC to protect memory. In the non-ECC DIMM configuration, we cannot store an aligned T1EC; therefore we place all the redundant information in the virtualized T2EC instead (we still refer to this as T2EC to keep the notation consistent). When data is read from main memory, we use the ECC address translation unit to find its EA. A T2EC LLC miss will fetch the T2EC from main memory because without ECC DIMMs, the information is required for error detection, and not just for correction. Unlike the two-tiered scenario, we fetch an entire cache-line's worth of T2EC data on a miss to amortize the DRAM access, and expect spatial locality to reduce the cost of following memory accesses. We only return data to the cache controller after the ECC information is fetched, and the data is verified. On a dirty write-back, the PA

is translated to an EA, and a T2EC is fetched from main memory if the EA is not already cached.

Virtualized ECC Protection Schemes

We now discuss possible DRAM configurations for Virtualized ECC, assuming a memory system that is representative of servers requiring Chipkill correct error protection. The baseline memory system is composed of a 128-bit wide DDR2 DRAM channel with an additional 16 bits of dedicated ECC. Our techniques work as well, or better, with DDR3 because it uses longer bursts and limits use of traditional Chipkill techniques.[3]

Virtualized ECC with ECC DIMMs

While traditional Chipkill uses a 4-check-symbol error code, Virtualized ECC, with two-tiered protection and T2EC virtualization, enables a more efficient 3-check-symbol error code.[8] In two-tiered error protection, the first two check symbols of the 3-check-symbol code construct a T1EC that can detect up to two symbol errors, while the T2EC is the third check symbol. If the T1EC detects a single symbol error, it is corrected using all three check symbols of both tiers. Our scheme uses 8-bit symbols for x4 and x8 DRAM configurations and 16-bit symbols with x16 chips. In the x4 system, we use two consecutive transfers of 128 bits so that we have an 8-bit symbol from each DRAM chip for the 8-bit symbol based error code. This effective 256-bit access does not actually change the DRAM access granularity, which is still 64 bytes as in the baseline system. The Virtualized ECC configurations using ECC DIMMs are summarized in Table 1, which also presents the details of the baseline Chipkill technique.

"While traditional Chipkill uses a 4-check-symbol error code, Virtualized ECC, with two-tiered protection and T2EC virtualization, enables a more efficient 3-check-symbol error code."

ECC x4 uses x4 chips, but utilizes the two-tiered approach to improve energy efficiency. We store two 8-bit check symbols in two ECC DRAM chips (in an ECC DIMM) to serve as a T1EC that can detect up to two chip failures. The third check symbol is the T2EC, which is stored in the data chips. Thus, ECC x4 only requires two ECC chips instead of the four chips of the conventional approach, saving eight pins and the associated costs of storage, power, and bandwidth.

ECC x8 is an efficient scheme for Chipkill protection using ECC DIMMs with x8 chips. We use the two ECC chips in a rank for the 2-symbol T1EC and store the third check symbol in data memory as the T2EC. Thus, we access 16 x8 data chips and two additional ECC chips on every read for the same 64-byte access granularity and redundancy overhead of the conventional Chipkill approach. Without virtualizing the T2EC, an additional DRAM chip to hold the third symbol would be touched on every access, increasing power and pin redundancy to a fixed 18.5 percent[3] as well as requiring nonstandard DIMMs.

Virtualized ECC with Non-ECC DIMMs

Another advantage of Virtualized ECC is the ability to add ECC protection to systems that use non-ECC DIMMs. We suggest schemes that are based on a 2-check-symbol Reed Solomon (RS) code[16], which can detect and correct one

	DRAM type	# Data DRAMs per rank	# ECC DRAMs per rank	Rank Organization	T2EC access	T2EC per cache line				
					read	write	No Protection	Chipkill detect	Chipkill correct	Double Chipkill correct
Baseline Chipkill Correct										
Baseline x4	x4	32	4	2 ECC DIMMs	N	N	N/A	N/A	N/A	N/A
Virtualized ECC										
ECC x4	x4	32	2	1 ECC DIMM and 1 Non-ECC DIMMM	N	Y	N/A	0 B	2 B	4 B
ECC x8	x8	16	2	2 ECC DIMMs	N	Y	N/A	0 B	4 B	8 B
Non-ECC x4	x4	32	N/A	2 Non-ECC DIMMs	Y	Y	0 B	2 B	4 B	8 B
Non-ECC x8	x8	16	N/A	2 Non-ECC DIMMs	Y	Y	0 B	4 B	8 B	16 B
Non-ECC x16	x16	8	N/A	2 Non-ECC DIMMs	Y	Y	0 B	8 B	16 B	32 B

Table 1: DRAM configurations for Chipkill correct of the baseline system and Virtualized ECC
(Source: University of Texas at Austin, 2010)

symbol error—in our case, a code that can tolerate any number of bit errors as long as they are confined to a single chip. The details for this scheme using x4, x8, and x16 DRAM chips are also summarized in Table 1. All three non-ECC DIMM configurations have the same protection capability, but the access properties differ. The wider symbols needed for x16 DRAMs imply that fewer T2EC words fit into an LLC line.

Flexible Protection Mechanisms

"A single processor with virtualized ECC can support different levels of error tolerance by varying the T2EC size..."

Virtualized ECC allows error protection levels and overhead to be proportional to dynamic application, user, and system needs. A single processor with virtualized ECC can support different levels of error tolerance by varying the T2EC size (see "T2EC per cache line" columns in Table 1). Supporting flexible protection tuning requires that the memory controller be able to compute and decode different codes as well as select which ECC technique to use for any given DRAM access. One elegant method to achieve the latter is to augment the OS page table and the TLB to include protection information for each page. We do not explore the benefits of protection tuning in this article but list two potential scenarios that we will explore in future work. The first is an opportunity to reduce system power by protecting

more critical data with stronger codes and potentially leaving some data unprotected. Another potential use is to adapt the protection level to the changes in environmental conditions, such as higher temperature or a higher energetic particle flux (while on an airplane, or if the system is located at a high altitude).

Evaluation

We evaluate the impact of Virtualized ECC on performance and energy efficiency relative to the baseline Chipkill system. The details of our evaluation methodology are available in our ASPLOS paper.[26]

Chipkill Performance and Energy

Figure 8 presents the execution time and system energy-delay product (EDP) of the Virtualized ECC configurations described in Table 1 normalized to those of the baseline x4 Chipkill ECC. For system EDP, we calculate system power consumption as the sum of processor core power, LLC power, and DRAM power. Virtualized ECC with ECC DIMMs has a very small impact on performance. With the exception of the PARSEC canneal workload, all applications have lower than 0.5-percent performance difference. This very low penalty is a result of effective T2EC caching and the fact that the additional DRAM traffic for writing out T2EC information is not on the computation critical path. Even the write-intensive GUPS microbenchmark that has no T2EC locality and very low arithmetic intensity only suffers a 10-percent reduction in performance. ECC x4, unfortunately, has little positive impact on EDP. ECC x8, on the other hand, shows a significant improvement in energy efficiency. DRAM power is reduced by an average of almost 30 percent, and EDP is improved by an average of 12 percent. EDP improvement is consistent using x8 DRAMs, with only two outliers: mcf and STREAM have a 20-percent and 18-percent improvement respectively. Both benchmarks place significantly higher pressure on the memory system, thus benefiting more from increased memory power efficiency. GUPS demands even higher memory performance. While the EDP of GUPS is improved by 10 percent in ECC x8 with more energy-efficient x8 DRAMs, it is degraded by 23 percent in ECC x4, mainly due to the increase in DRAM power consumption (7 percent). Note that supporting Chipkill with x8 DRAMs in conventional systems is not possible unless custom-designed DIMMs with higher redundancy or increased access granularity are used.

“Virtualized ECC with ECC DIMMs has a very small impact on performance.”

“ECC x8, shows a significant improvement in energy efficiency. DRAM power is reduced by an average of almost 30 percent, and EDP is improved by an average of 12 percent.”

Virtualized ECC can also bring DRAM error tolerance to systems that use non-ECC DIMMs. The extra DRAM accesses required for every read (and not just for writes) result in a larger impact on performance. Even with this extra traffic, however, application performance is degraded by 3 percent, 6 percent, and 9 percent using x4, x8, and x16 chips, respectively, on average. While the x4 configuration slightly degrades EDP, wider DRAM configurations improve EDP by 5 percent (x8) and 12 percent (x16) when compared to a standard Chipkill that uses ECC DIMMs.

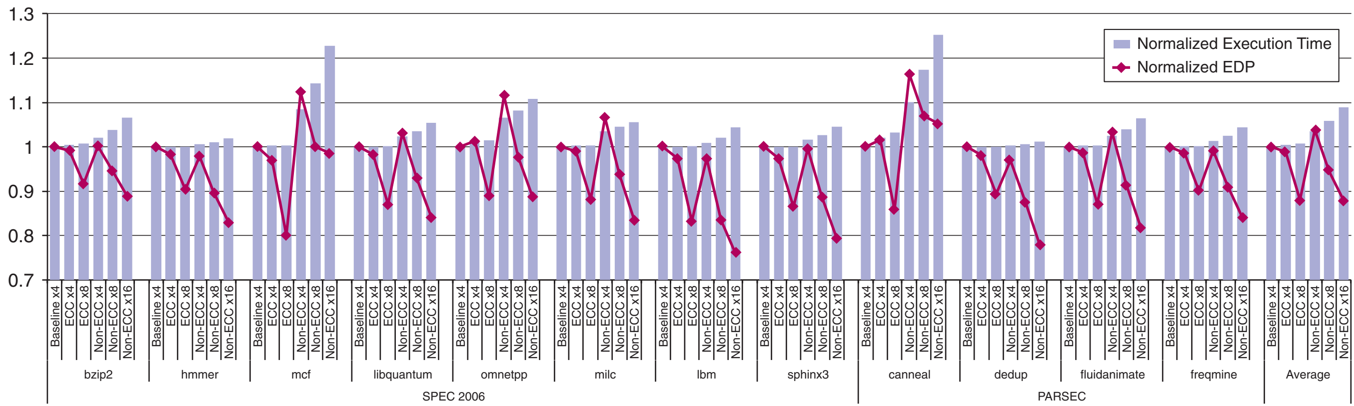


Figure 8: Performance and system EDP for baseline and Virtualized ECC Chipkill correct
(Source: University of Texas at Austin, 2010)

Flexible Protection

Virtualized ECC enables flexibility in choosing the error protection level based on dynamic application, user, and system needs. To assess the new tradeoff that Virtualized ECC enables, we evaluate the effect of different T2EC sizes, which are summarized in Table 2. The detailed evaluation of individual applications can be found in our ASPLOS paper.[26]

	ECC x8			Non-ECC x8				Non-ECC x16			
	Chipkill detect	Chipkill correct	Double Chipkill correct	No protection	Chipkill detect	Chipkill correct	Double Chipkill correct	No protection	Chipkill detect	Chipkill correct	Double Chipkill correct
Performance penalty	0%	0.7%	1%	0%	3.4%	5.8%	8.9%	0%	5.8%	8.9%	12.8%
DRAM power reduction	29%	27.8%	27.2%	37.1%	32.6%	30.1%	26.7%	59.5%	53.4%	50.1%	46.2%
System EDP gain	14.6%	12%	11.2%	17.3%	10.4%	5.6%	–0.9%	27.8%	17.8%	12.1%	4.9%

Table 2: Summary of flexible error protection results with Virtualized ECC. The x4 configurations are omitted since they do not show significant gain
(Source: University of Texas at Austin, 2010)

"...increasing the protection level increases EDP and execution time. The impact of adding the capability to tolerate a second dead chip, however, has a fairly small overhead overall..."

As expected, increasing the protection level increases EDP and execution time. The impact of adding the capability to tolerate a second dead chip, however, has a fairly small overhead overall when using ECC DIMMs. Double Chipkill correct increases execution time by at most 0.3 percent relative to single Chipkill correct, and system EDP is still 10–20 percent better than that of conventional x4 Chipkill.

Significance and Potential Impact of Proportional Memory

We have shown that DGMS and VECC offer substantive advantages for current systems and technologies through memory proportionality. Trends in computer architectures, workloads, and memory technologies will make proportional memory systems even more valuable in future systems.

DGMS

The idea of dynamic, hardware-only, multigranularity memory access has great potential to improve system efficiency in current and future architectures where off-chip bandwidth is scarce. We describe both near-term and long-term potential impacts of DGMS in detail below.

"The idea of dynamic, hardware-only, multigranularity memory access has great potential to improve system efficiency in current and future architectures where off-chip bandwidth is scarce."

Memory I/O Power

Recent advances in off-chip interconnect technology enable high-speed I/O in DRAM: DDR3 and DDR4 use sophisticated I/O signaling to overcome signal integrity issues, at the cost of increased I/O power. For instance, on-die termination used in DDR3 burns out static power on the terminating pins when transferring data. By transferring only useful data over I/O pins, DGMS can reduce the static power burnt in terminating I/O pins.

DGMS for SIMD and Vector Architectures

Wide SIMD architectures have been gaining popularity, as seen in GPUs and increasing SSE operand widths. Utilizing wide SIMD for complex algorithms frequently requires non-unit stride and indexed gather/scatter memory operations. Such SIMD architectures will suffer from poor effective throughput with conventional coarse-grain-only memory systems. DGMS has great potential to achieve higher throughput for non-unit stride and indexed gather/scatter operations. In addition, the notion of vector memory operations can help to predict memory access granularities more accurately.

Emerging Applications

Emerging workloads (such as social network services, data analytics, and data-intensive computing) frequently leverage graph structures. Our initial evaluation indicates that graph applications (such as SSCA2) benefit significantly from DGMS due to their pointer-chasing intensive memory-access patterns. We believe that DGMS can provide the high graph-traversal rates demanded by emerging systems without sacrificing efficiency. Furthermore, the higher fine-grained access throughput of DGMS may allow software engineers to rethink the data structures used for graph storage and sparse matrices. With DGMS, a more intuitive data structure using pointer operations can perform comparable to (or may even outperform) a cache-conscious data structure such as a compressed-sparse-row based approach.

Bandwidth Scaling

Chip multiprocessor (CMP) architectures are ubiquitous, and trends indicate rapidly increasing numbers of integrated cores and threads. Off-chip bandwidth scaling, on the other hand, is limited because of scarce pins and

power. As a result, most applications, if not all, will become bandwidth-limited, and efficient utilization of the finite off-chip bandwidth is key to continued performance scaling.

DGMS judiciously applies fine-grained accesses to boost the available effective bandwidth when spatial locality is low. DGMS is unique in that it is orthogonal to other circuit-level techniques (such as high-speed signaling or optical communication) and in that it reduces power consumption at the same time as it improves throughput (unlike other bandwidth-enhancing techniques). As such, DGMS is a cost-effective way to scale memory throughput and efficiency.

DGMS for Emerging Memory Technologies

Recently, new memory technologies have emerged that can revolutionize memory systems by providing new capabilities and significantly increased capacity: nonvolatile memory (NVRAM) including phase-change memory and memristors; and Micron's hybrid memory cube (HMC) with 3D stacking. Unfortunately, the interface bandwidth connecting these off-chip components to the processor will scale much more slowly than potential capacities. Thus, compared to current systems, the ratio of capacity to bandwidth of main memory is expected to get worse, necessitating better utilization of available bandwidth. DGMS offers an effective solution for accessing only the required data without sacrificing transfer efficiency. We believe the dynamic tradeoff between amortizing control bandwidth, ECC overhead, and fine-granularity accesses will increase in importance once NVRAM and HMC are commoditized.

DGMS for Mass Storage

The underlying idea of DGMS can help improve any system where the interface bandwidth is constrained. Memory architectures such as disaggregated memory[13], Violin memory[22], Fusion I/O[10], or PCIe-attached PCRAM[4] are of great interest commercially and improve database performance by orders of magnitude. However, these memory architectures have a relatively low-bandwidth interface; as such, a technique similar to DGMS can better utilize the limited channel in such emerging memory systems.

Virtualized ECC (VECC)

"Virtualized ECC ..."

"...enables new tradeoffs between cost, error protection, and performance. More importantly, these new tradeoffs can be tuned at runtime to meet dynamically changing reliability needs..."

VECC is a general scheme for virtualizing the memory error tolerance mechanisms. Virtualized ECC is based on two key ideas: (1) decoupling data from its ECC-redundant information; and (2) virtualizing the redundant information to allow it to share space with the data in both memory and caches. This unique combination enables new tradeoffs between cost, error protection, and performance. More importantly, these new tradeoffs can be tuned at runtime to meet dynamically changing reliability needs while simultaneously relaxing DRAM system design constraints. We believe that these capabilities will have a lasting and important influence on memory error protection, and we discuss the potential for other impact below.

Double Chipkill and More

Even though the cost of providing Chipkill protection is high, the industry is offering even stronger memory reliability for high-end servers. Implementing schemes such as the double device data correction (DDDC) of Itanium® 2 and new Intel® Xeon® based systems, however, requires more redundancy and constrains the design space of the memory system. This results in less energy-efficient memory system configurations, such as requiring the use of x4 DRAM chips. Configurations with more energy-efficient x8 DRAMs are highly desirable; our system provides a cost-effective way to provide strong protection is such systems. Moreover, Virtualized ECC can apply the stronger and more costly protection only to memory pages that actually require it, minimizing the cost of double Chipkill correct.

"Virtualized ECC can apply stronger and more costly protection only to memory pages that actually require it…"

Protection for Low Power and Stacked DRAM

Recent research has offered new solutions for better energy efficiency by avoiding DRAM "overfetch," where a large DRAM page is activated but only a fraction is actually transferred to the processor. These proposals include MC-DIMM[3] as well as more recent research that redesigns the DRAM array itself[20]. These designs achieve energy efficiency through a narrow data path and through the activation of fewer bits in DRAM; however, such techniques make memory protection difficult. Contrary to conventional reliability, Virtualized ECC can effectively protect such systems from memory errors. Recent work has picked up on some of the ideas of VECC and extended aspects of the design to stacked memories and memory cubes, showing the versatility of this approach.[21]

Tunable/Adaptive Reliability

We are excited about the potential of using Virtualized ECC as the basis of a tunable/adaptive reliability framework. Such a dynamically adaptive paradigm is presented in recent cross-layer reliability[1] collaborative efforts in academia and industry and is discussed in papers such as "Mixed-Mode Multicore Reliability"[23]. To the best of our knowledge, Virtualized ECC was the first mechanism to enable tunable and adaptive memory protection. A recently proposed mechanism offers an alternative design for adapting memory protection, where access granularity can be dynamically increased rather than adapting capacity overhead.[12] Such a design offers a tradeoff between capacity and granularity and may offer interesting interactions between VECC and DGMS.

GPU Memory Protection

Graphics processing units (GPUs) are already being used as compute accelerators, and memory protection is essential for integrating GPUs in larger high-end systems. GPU memory systems use high-bandwidth memory products (such as GDDR5), where the dedicated storage for ECC is not available. Virtualized ECC for non-ECC DIMMs can be straightforwardly applied and enable memory protection in GPU systems. At the moment, NVIDIA's latest GPUs[15] support SEC-DED protection with GDDR5 memory, and we believe their mechanism is quite similar to the technique

we presented (but without adaptivity). We argue that SEC-DED (and static in-memory ECC) is not enough for GPU systems. Unlike general purpose systems, GPUs do not utilize memory modules so the entire card needs to be replaced if only a single memory device fails (or manifests any intolerable hard failures). In future systems, a more stringent protection mechanism (such as Chipkill) will be required. The flexible memory protection enabled with Virtualized ECC can allow GPU systems to proportionally support the required error tolerance level. In essence, the cost of Chipkill will only be incurred after a memory device fails, enabling graceful performance degradation at low cost.

Emerging NVRAM

Virtualized ECC can also protect emerging nonvolatile memory (NVRAM) such as phase-change memory (PCRAM). This new memory technology has finite write-endurance so tolerating hard errors is a challenge. With Virtualized ECC, we can adapt error tolerance levels to NVRAM wear-out status, increasing error tolerance levels as NVRAM devices wear out.

Conclusions

"To provide efficient and reliable operation across differing applications and usage scenarios, future memory systems must transfer and protect data in a manner that is proportional to application, system, and environmental needs."

Conventional memory system techniques will not be able to satisfy the dynamic and heterogeneous requirements of future workloads with scarce available off-chip bandwidth. To provide efficient and reliable operation across diffing applications and usage scenarios, future memory systems must transfer and protect data in a manner that is proportional to application, system, and environmental needs. This article describes two mechanisms that offer such memory proportionality. The first, DGMS, is a minimally intrusive hardware-only mechanism that transfers data from off-chip memory at a granularity appropriate for the spatial locality of running programs. The second, VECC, allows the level of error protection to be dynamically tuned and adapted. These mechanisms offer performance, power, and reliability advantages in current systems, and will become necessary in future systems given computing trends.

References

[1] A. DeHon, N. Carter, and H. Quinn (Editors), Final Report for CCC Cross-Layer Reliability Visioning Study, 2011. http://www.xlayer.org

[2] D. Abts, A. Bataineh, S. Scott, G. Faanes, J. Schwarzmeier, E. Lundberg, M. Byte, and G. Schwoerer. "The Cray Black Widow: A highly scalable vector multiprocessor." In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 2007.

[3] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber. "Future scaling of processor-memory interfaces." In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 2009.

[4] A. Akel, A. M. Caulfield, T. I. Mollov, R. K. Gupta, and S. Swanson. "Onyx: A protoype phase-change memory storage array." In Proceedings of the USENIX conference on Hot topics in storage and file systems (Hot Storage), 2011.

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

[6] L. Barroso and U. Holzle. "The case for energy-proportional computing." Computer, 40(12):33–37, 2007.

[7] C. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos. "Accurate and complexity effective spatial pattern prediction." In Proceedings of the International Symposium on High Performance Computer Architecture (HPCA), 2004.

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

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

[10] Fusion-io. "Fusion-io ioDrive performance testing: A comparative study on storage performance improvement using Fusion-io technology." http://www.fusionio.com/fusion-io-iodrive-performance-testing-a-comparative-study-on-storage-performance-improvement-using-fusion-io-technology/, 2011.

[11] S. Kumar and C. Wilkerson. "Exploiting spatial locality in data caches using spatial footprints." In Proceedings of the International Symposium on Computer Architecture (ISCA), 1998.

[12] S. Li, D. H. Yoon, K. Chen, J. Zhao, J. H. Ahn, J. B. Brockman, Y. Xie, and N. P. Jouppi. "MAGE: Adaptive granularity and ECC for resilient and power efficient memory systems." In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 2012.

[13] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch. "Disaggregated memory for expansion and sharing in blade servers." In Proceedings of the International Symposium on Computer Architecture (ISCA), 2009.

[14] J. S. Liptay. "Structural aspects of the System/360 Model 85, part II: The cache." IBM Systems Journal, 7:15–21, 1968.

[15] NVIDIA. "Fermi architecture." http://www.nvidia.com/object/fermi_architecture.html.

[16] I. S. Reed and G. Solomon. “Polynomial codes over certain finite fields.” Journal of the Society for Industrial and Applied Mathematics, 8:300–304, 1960.

[17] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E.Weissmann. “Power management architecture of the Intel, microarchitecture code-named Sandy Bridge.” IEEE Micro, 32(2):20–27, 2012.

[18] N. N. Sadler and D. J. Sorin. “Choosing an error protection scheme for a microprocessor’s L1 data cache.” In IEEE International Conference on Computer Design (ICCD), Oct. 2006.

[19] C. Slayman. “Impact of error correction code and dynamic memory reconfiguration on high-reliability / low-cost server memory.” In Proceedings of the International Integrated Reliability Workshop (IIRW), 2006.

[20] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. Jouppi. “Rethinking DRAM design and organization for energy constrained multi-cores.” In Proceedings of the International Symposium on Computer Architecture (ISCA), 2010.

[21] A. N. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis, and N. P. Jouppi. “LOT-ECC: Localized and tiered reliability mechanisms for commodity memory systems.” In Proceedings of the International Symposium on Computer Architecture (ISCA), 2012.

[22] Violin Memory Inc. “Scalable memory applicance.” http://violin-memory.com/DRAM.

[23] P. M. Wells, K. Chakraborty, and G. S. Sohi. “Mixed-mode multicore reliability.” In Proceedings of the International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2009.

[24] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-L. Lu. “Reducing cache power with low-cost, multi-bit error correcting codes.” In Proceedings of the International Symposium on Computer Architecture (ISCA), June 2010.

[25] D. H. Yoon and M. Erez. “Memory mapped ECC: Low-cost error protection for last level caches.” In Proceedings of the International Symposium on Computer Architecture (ISCA), 2009.

[26] D. H. Yoon and M. Erez. “Virtualized and flexible ECC for main memory.” In Proceedings of the International Symposium on

Architectural Support for Programming Languages and Operating Systems (ASPLOS), Mar. 2010.

[27] D. H. Yoon, M. K. Jeong, and M. Erez. "Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput." In Proceedings of the International Symposium on Computer Architecture (ISCA), 2011.

[28] D. H. Yoon, M. K. Jeong, M. B. Sullivan, and M. Erez. "The dynamic granularity memory system." In Proceedings of the International Symposium on Computer Architecture (ISCA), June 2012.

[29] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. "Mini-rank: Adaptive DRAM architecture for improving memory power efficiency." In Proceedings of the International Symposium on Microarchitecture (MICRO), 2008.

Author Biographies

Doe Hyun Yoon is a post-doctoral researcher at Hewlett-Packard Labs. He has a BS in Electrical Engineering (Yonsei, 1998), an MS in Electrical and Computer Engineering (Yonsei, 2000), an MS in Electrical Engineering (Stanford, 2007), and a PhD in Electrical and Computer Engineering (the University of Texas at Austin, 2011). His research includes energy efficiency and reliability in caches, DRAM, and nonvolatile memory. He can be contacted at doe-hyun.yoon@hp.com

Min Kyu Jeong is a senior hardware engineer at Oracle Labs. Min Kyu holds a BS in Computer Science and Engineering from Seoul National University, and an MSE and a PhD in Electrical Engineering from the University of Texas at Austin. His research interests are in the area of computer architecture, with emphasis on the memory system. He can be contacted at min.jeong@oracle.com

Michael Sullivan is a graduate research assistant at the University of Texas at Austin who studies the design of dependable and power-efficient systems with an emphasis on reliable arithmetic. He holds a BS in computer engineering, a BA in mathematics, and an MS in computer science from George Mason University. He also holds an MSE in computer engineering from the University of Texas at Austin. Michael can be contacted at mbsullivan@utexas.edu.

Mattan Erez is an Associate Professor of Electrical and Computer Engineering at the University of Texas at Austin. Mattan hold a BA in Physics (Technion, 1999) and a BSc (Technion, 1999), MS (Stanford, 2002), and PhD (Stanford, 2007) in Electrical Engineering. His research interests include computer architecture and programming models. Contact him at mattan.erez@utexas.edu.

ERROR ANALYSIS AND RETENTION-AWARE ERROR MANAGEMENT FOR NAND FLASH MEMORY

Contributors

Yu Cai
Carnegie Mellon University

Gulay Yalcin
Barcelona Supercomputing Center

Onur Mutlu
Carnegie Mellon University

Erich F. Haratsch
LSI Corporation

Adrian Cristal
Barcelona Supercomputing Center

Osman S. Unsal
Barcelona Supercomputing Center

Ken Mai
Carnegie Mellon University

With continued scaling of NAND flash memory process technology and multiple bits programmed per cell, NAND flash reliability and endurance are degrading. In our research, we experimentally measure, characterize, analyze, and model error patterns in nanoscale flash memories. Based on the understanding developed using real flash memory chips, we design techniques for more efficient and effective error management than traditionally used costly error correction codes.

In this article, we summarize our major error characterization results and mitigation techniques for NAND flash memory. We first provide a characterization of errors that occur in 30- to 40-nm flash memories, showing that retention errors, caused due to flash cells leaking charge over time, are the dominant source of errors. Second, we describe retention-aware error management techniques that aim to mitigate retention errors. The key idea is to periodically read, correct, and reprogram (in-place) or remap the stored data before it accumulates more retention errors than can be corrected by simple ECC. Third, we briefly touch upon our recent work that characterizes the distribution of the threshold voltages across different cells in a modern 20- to 24-nm flash memory, with the hope that such a characterization can enable the design of more effective and efficient error correction mechanisms to combat threshold voltage distortions that cause various errors. We conclude with a brief description of our ongoing related work in combating scaling challenges of both NAND flash memory and DRAM memory.

"...we summarize our major error characterization results and mitigation techniques for NAND flash memory."

Introduction

During the past decade, the capacity of NAND flash memory has increased more than 1000 times as a result of aggressive process scaling and multilevel cell (MLC) technology. This continuous capacity increase has made flash economically viable for a wide variety of applications, ranging from consumer electronics to primary data storage systems. However, as flash density increases, NAND flash memory cells are more subject to various device and circuit level noise, leading to decreasing reliability and endurance. The P/E cycle endurance of MLC NAND flash memory has dropped from ~10K for 5x-nm (that is, 50- to 59-nm) flash to around ~3K for current 2x-nm (that is, 20- to 29-nm) flash.[1][5] The reliability and endurance are expected to continue to decrease when 1) more than two bits are programmed per cell, and 2) flash cells scale beyond the 20-nm technology generations. This trend is forcing flash memory designers to apply even stronger error correction codes (ECC) to tolerate the increasing error rates, which comes at the cost of additional complexity and overhead.[4]

"...as flash density increases, NAND flash memory cells are more subject to various device and circuit level noise,..."

In our research at Carnegie Mellon University, we aim to develop new techniques that overcome reliability and endurance challenges of flash memory to enable its scaling beyond the 20-nm technology generations. To this end, we experimentally measure, characterize, analyze, and model error patterns that occur in existing flash chips, using an experimental flash memory testing and characterization platform we have developed.[2] Based on the understanding we develop from our experiments, we aim to develop error management techniques that aim to mitigate the fundamental types of errors that are likely to increase as flash memory scales. Our goal is to design techniques that are more effective and more efficient than stronger error correction codes (ECCs), which has been the traditional way of improving endurance and reliability of flash memory. In this article, we provide an overview of the results of our recent error characterization experiments[3][6] and describe some error mitigation techniques.[4]

"Our goal is to design techniques that are more effective and more efficient than stronger error correction codes (ECCs), which has been the traditional way of improving endurance and reliability of flash memory."

In particular, we have recently experimentally characterized complex flash errors that occur at 30- to 40-nm flash technologies[3], categorizing them into four types: retention errors, program interference errors, read errors, and erase errors. Our characterization shows the relationship between various types of errors and demonstrates empirically using real 3x-nm flash chips that retention errors are the most dominant error type. Our results demonstrate that different flash errors have distinct patterns: retention errors and program interference errors are program/erase-(P/E)-cycle-dependent, memory-location-dependent, and data-value-dependent. Since the observed error patterns are due to fundamental circuit and device behavior inherent in flash memory, we expect our observations and error patterns to also hold in flash memories beyond 30-nm technology node.

"Our characterization shows the relationship between various types of errors and demonstrates empirically using real 3x-nm flash chips that retention errors are the most dominant error type."

Based on our experimental characterization results that show that the retention errors are the most dominant errors, we have developed a suite of techniques to mitigate the effects of such errors, called Flash Correct-and-Refresh (FCR).[4] The key idea is to periodically read each page in flash memory, correct its errors using simple ECC, and either remap (copy/move) the page to a different location or reprogram it in its original location by recharging the floating gates before the page accumulates more errors than can be corrected with simple ECC. Our simulation experiments using real I/O workload traces from a variety of file system, database, and search applications show that FCR can provide 46x flash memory lifetime improvement at only 1.5 percent energy overhead, with no additional hardware cost.

Finally, we also briefly describe major recent results of our measurement and characterization of the threshold voltage distribution of different logical states in MLC NAND flash memory.[6] Our data shows that the threshold voltage distribution of flash cells that store the same value can be approximated, with reasonable accuracy, as a Gaussian distribution. The threshold voltage distribution of flash cells that store the same value gets distorted as the number of P/E cycles increases, causing threshold voltages of cells storing different values to overlap with each other, which can lead to the

"...we hope that the characterization, understanding, models, and mechanisms provided in this work (and in our aforementioned previous works) would enable the design of new and more effective error tolerance mechanisms that can make use of the observed characteristics and the developed models."

incorrect reading of values of some cells as flash cells accumulate P/E cycles. We find that this distortion can be accurately modeled and predicted as an exponential function of the P/E cycles, with more than 95-percent accuracy. Such predictive models can aid the design of more sophisticated error correction methods, such as LDPC codes[7], which are likely needed for reliable operation of future flash memories. Even though we will not describe these models in detail in this article, the interested reader can refer to Cai et al.[6] for more detail.

As flash memory continues to scale to smaller feature sizes, we hope that the characterization, understanding, models, and mechanisms provided in this work (and in our aforementioned previous works[3][4][6]) would enable the design of new and more effective error tolerance mechanisms that can make use of the observed characteristics and the developed models.

Flash Memory Background

NAND flash memory can be of two types: single level cell (SLC) flash and multilevel cell (MLC) flash. Only one bit of information can be stored in an SLC flash cell, while multiple bits (2 to 4 bits) can be stored in an MLC flash cell.[8][9][10] MLC flash represents n bits by using 2^n non-overlapping threshold voltage (V_{th}) windows. The threshold voltage of a given cell is mainly affected by the number of electrons trapped on the floating gate. Figure 1 shows the bit mapping to V_{th} and the relative proportion of electrons on the floating gates of a 2-bit MLC flash.

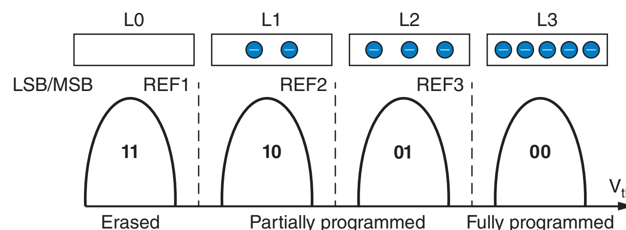


Figure 1: Threshold voltage distribution example of 2-bit MLC flash
(Source: Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, 2012[3])

A NAND flash memory chip is composed of thousands of blocks. Each block is a storage array of floating gate transistors. A flash block usually has 32 to 64 wordlines. The cells on the same wordline can be divided into two groups: even and odd, depending on the physical location. For SLC flash, each group corresponds to just one logical page, that is, even pages and odd pages. As an MLC flash cell stores multiple bits, the bits corresponding to the same logical location of a cell in a group form one logical page. For example, all the most significant bits (MSBs) of the cells of an even group form one MSB-even page. Similarly, other types of pages are MSB-odd page, LSB-even page, and LSB-odd page. The page number assignments for each bit of the flash memory

are shown in Figure 2(a), ranging from 0 to 127 for the selected flash in this article. The size of each page is generally between 2 KB and 8 KB (16k and 64k bitlines). The stack of flash cells in the bitline direction forms one string. The string is connected to a bit line through SGD (the select gate at the drain end) and connect to the common source diffusion through SGS (the select gate at the source end) as shown in Figure 2(b). Flash memories generally support three fundamental operations as follows:

"Flash memories generally support three fundamental operations..."

Figure 2: NAND flash organization and operations: (a) Partial block organization and program operation on page 118; (b) Read operation
(Source: Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, 2012[3])

Erase

During the *erase* operation, a high positive erase voltage (for example, 20V) is applied to the substrate of all the cells of the selected block and the electrons stored on the floating gate are tunnelled out through Fowler-Nordheim (FN) mechanisms.[9] After a successful erase operation, all charge on the floating gates is removed and all the cells are configured to the L0 (11) state. The erase operation is at the granularity of one block.

Program

During the *program* operation, a high positive voltage is applied to the wordline, where the page to be programmed is located. The other pages sharing the same wordline are inhibited (from being programmed) by applying 2V to their corresponding bitlines to close SGD and boost the potential of corresponding string channel. The voltage bias for programming page 118 is shown in Figure 2(a) as an example. The programming process is typically realized by the incremental step pulse programming (ISPP) algorithm.[11] ISPP first injects electrons into floating gates to boost the V_{th} of programmed cells through FN mechanisms and then performs a verification to check whether the V_{th} has reached the desired level. If V_{th} is still lower

than the desired voltage, the program-and-verify iteration will continue until the cell's V_{th} has reached the target level. Note that the NAND flash program operation can only add electrons into the floating gate and cannot remove them from the gate. As a result, the threshold voltage can only shift toward the right in Figure 1 during programming. The program operation is executed at page granularity.

Read

The read operation is also at the page granularity and the voltage bias is shown in Figure 2(b). The SGD, SGS, and all deselected wordlines are turned on. The wordline of selected read page is biased to a series of predefined reference voltages and the cell's threshold voltage can be determined to be between the most recent two read reference voltages when the cell conducts current.

Flash Memory Error Classification

We test the NAND flash memory using the cycle-by-cycle programming model shown in Figure 3. During each P/E cycle, the selected flash block is first erased. Then data are programmed into the block on a page granularity. Once a page has been programmed, it cannot be reprogrammed again unless the whole block is erased for the next P/E cycle. The stored data will be alive in the block until it becomes invalid. Before the stored data becomes invalid, it can be accessed multiple times. Once a page is programmed, we can test how long it retains data by reading the data value of the page after a *retention interval,* and comparing it to the original programmed value. Whether or not the data is retained correctly between two accesses depends on the time distance of two consecutive accesses. We repeat the above per-P/E-cycle procedure for thousands of cycles until the flash memory block becomes unreliable and reaches the end of its lifetime. Errors could happen in any stage of this testing process. We classify the observed errors into four different types, from the flash controller's point of view:

"We classify the observed errors into four different types, from the flash controller's point of view:."

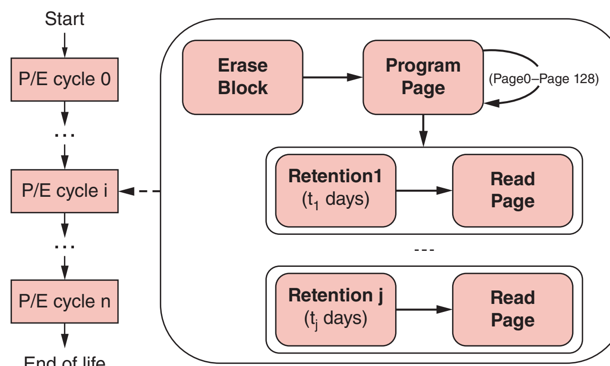


Figure 3: NAND flash programming model for error characterization
(Source: Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, 2012[3])

- *An erase error* happens when an erase operation fails to reset the cells to the erased state. This is mainly due to manufacturing process variations or defects caused by trapped electrons in the tunnel oxide after stress due to repeated P/E cycles.
- *A program interference error* happens when the data stored in a page changes (unintentionally) while a neighboring page is being programmed due to parasitic capacitance-coupling.
- *A retention error* happens when the data stored in a cell changes over time. The main reason is that the charge programmed in the floating gate may dissipate gradually through the leakage current.
- *A read error* happens when the data stored in a cell changes as a neighboring cell on the same string is read over and over.

"A retention error *happens when the data stored in a cell changes over time."*

Error Characterization Methodology

The following section describes the error characterization methodology.

Experimental Hardware

To characterize the error patterns, we built a hardware test platform that allows us to issue commands to raw flash chips without ECC.[2] The test platform mainly consists of three components: a HAPS–52 board with Xilinx Virtex-5 FPGAs used as NAND flash controller, a USB daughter board used to connect to the host machine, and a custom flash daughter board. The flash memory under test is a 2-bit MLC NAND flash device manufactured in 3x-nm technology. The device is specified to survive 3000 P/E cycles stress under 10-year data retention time if ECC with 4-bit error correction per 512 bits is applied. Details of the experimental flash test platform we use to collect our data are provided in [2].

"To characterize the error patterns, we built a hardware test platform that allows us to issue commands to raw flash chips without ECC."

Flash Error Testing Procedure

To test the P/E-cycle-dependence of errors, we stress-cycle flash memory blocks up to a certain number of erase cycles and check if the data is retained. This is achieved by iteratively erasing a block and programming pseudorandom data into it at room temperature.

We test whether the data is retained after T amount of time, to characterize retention errors. T is called the *retention test time* and is varied in the range of 1 day, 3 days, 3 weeks, 3 months, 1 year, and 3 years. We consider T = {1 day, 3 days} to be short-term retention tests, while the remaining values of T are long-term retention tests. Short-term retention errors are characterized under room temperature. Long-term retention errors are characterized by baking the flash memory in the oven under 125° C. According to the classic temperature-activated Arrhenius law[12], the baking time at 125° C corresponds to about 450 times of the lifetime at room temperature (25° C).

We refer the reader to Cai et al.[3] for our testing and characterization methodology for program interference and read errors.

Error Characterization Results

We provide our experimental measurements of the errors in the state-of-the-art 3x-nm MLC NAND flash memory we have tested using our infrastructure. NAND flash errors show strong correlation with the number of P/E cycles, location of the physical cells, and the data values programmed into the cells. The following subsections analyze detailed error properties and briefly describe the causes of the observed phenomena. Our main focus in this article is retention errors, but our previous work analyzes all types of errors in detail[3], and we refer the reader to [3] for characterization and analysis of program interference, read, and erase errors.

Error Rate Analysis for Different Error Types

Figure 4 shows the bit error rate due to various types of NAND flash errors. The x-axis shows the number of P/E cycles and the y-axis depicts the raw bit error rate. Error rates are obtained characterized from the beginning of the flash chip's life until the region of >100x times of its specified lifetime (3000 P/E cycles for the chips we tested). We make several observations about error properties.

"...all types of errors are highly correlated with P/E cycles."

First, all types of errors are highly correlated with P/E cycles. At the beginning of the flash lifetime, the error rate is relatively low and the raw bit error rate is below 10^{-4}, within the specified lifetime (3K cycles). As the P/E cycles increase, the error rate increases exponentially. The P/E cycle-dependence of errors can be explained by the deterioration of the tunnel oxide under cycling stress. During erase and program operations, the electric field strength across the tunnel oxide is very high (for example, several million volts per centimeter). Such high electric field strength can lead to structural defects that trap electrons in the oxide layer. Over time, more and more defects accumulate and the insulation strength of the tunnel oxide degrades. As a result, charge can leak through the tunnel oxide and the threshold voltage of the cells can change more easily. This leads to more errors for all types of flash operations.

"The long-term retention errors are the most dominant; their rate is highest."

Second, there is a significant error rate difference between various types of errors. The long-term retention errors are the most dominant; their rate is highest. The program interference error rate ranks the second and is usually between error rates of 1-day and 3-day retention errors. The read error rate is slightly less than 1-day retention error rate, while the erase error rate is only around 7 percent of the read error rate.

Third, retention error rates are highly dependent on retention test time. If the time before we test for retention errors is longer, the floating gate of flash memory is more likely to lose more electrons through leakage current. This eventually leads to V_{th} shift across V_{th} windows and causes errors (see [6] for more detail). From our experimental data, we can see that the retention error rate increases linearly with the retention test time. For example, the 3-year retention error rate is almost three orders of magnitude higher than one-day retention error rate.

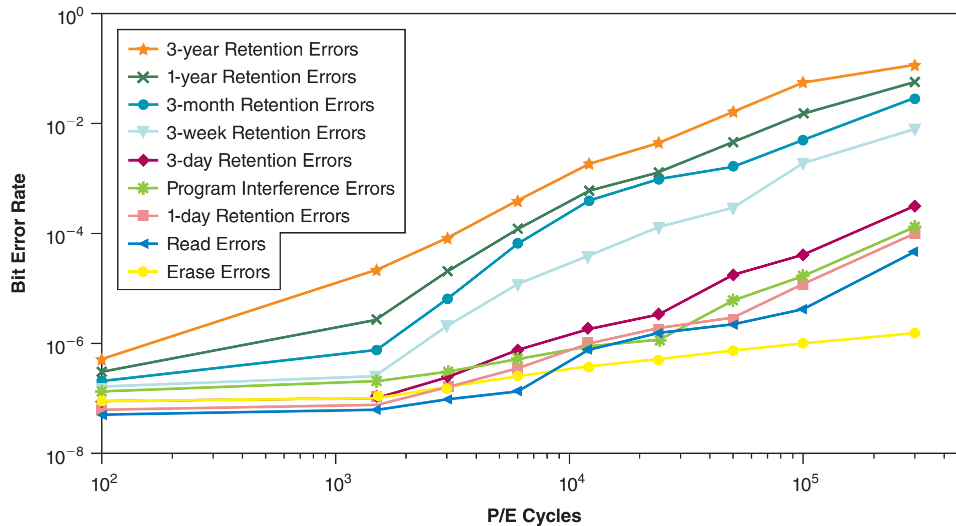


Figure 4: Rates of various types of errors as P/E cycles increase
(Source: Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, 2012[3])

Retention Error Analysis

Value dependence of retention errors: We find that the retention errors are value dependent; their frequency is asymmetric with respect to the value stored in the flash cell. Figure 5 demonstrates this asymmetric nature of retention errors by showing how often each possible value transition was observed due to an error. We characterized all possible error transitions, in the format *AB*→*CD*, where *AB* are the two bits stored in the cell before retention test, while *CD* are the two bits recorded in the cell after retention test. If the errors are not value dependent, the fraction of erroneous changes between each of the different value pairs should be equal. But, we find that this is not the case. The most common retention errors are 00→01, 01→10, 01→11 and 10→ 11, with their relative percentage over all retention errors being 46 percent, 44 percent, 5 percent, and 2 percent, respectively. The relative percentages among various error transitions are almost constant for different P/E cycles.

"We find that the retention errors are value dependent; their frequency is asymmetric with respect to the value stored in the flash cell."

To understand the reasons for value dependence, we need to observe Figure 1 in conjunction with the value transition observed in the most common retention errors. We find that the most common retention errors (00→01, 01→10, 01→11, and 10→ 11) are all cases in which V_{th} shifts towards the left (see Figure 1). This can be explained by an understanding of the retention error mechanisms. During retention test, the electrons stored on the floating gate gradually leak away under stress induced leakage current (SILC). When the floating gate loses electrons, its V_{th} shifts left from the state with more electrons to the state with fewer programmed electrons (as seen in Figure 1, states to the left have fewer electrons trapped on the gate than states to the right). It is significantly less likely for the cells to shift right in the opposite direction because this requires the addition of more electrons. As the states of 00 and 01 hold the largest number of electrons on the floating gates, SILC is higher in

these states and therefore it is more likely for the V_{th} of the cells in these two states to shift left, which leads to the observation that most common errors are due to shifting from these states (00→01, 01→10, 01→11).

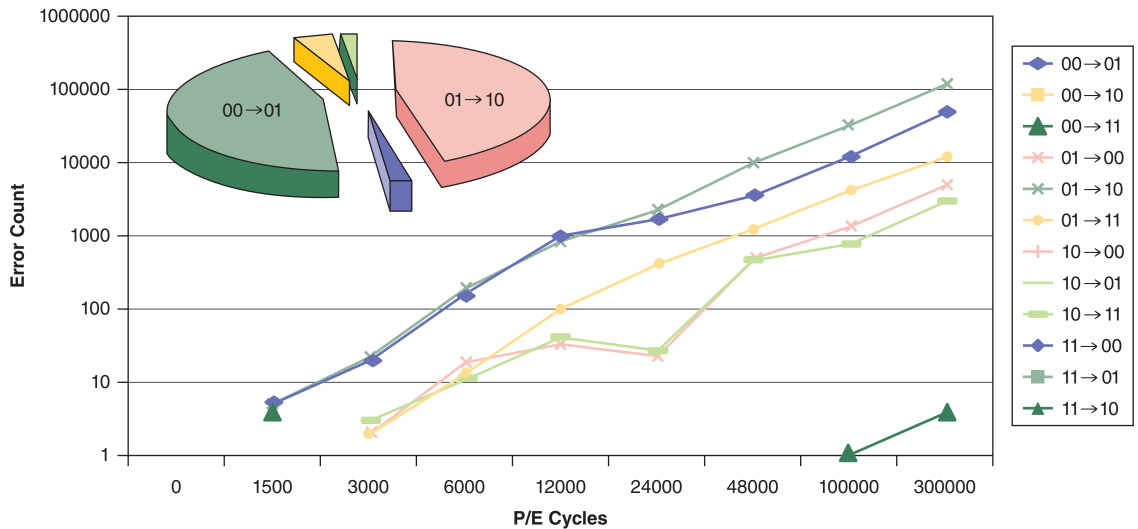


Figure 5: Value dependence of retention errors
(Source: Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, 2012[3])

"We also characterized the relation between retention errors and their physical locations."

Location dependence of retention errors: We also characterized the relation between retention errors and their physical locations. The experimental results are shown in Figure 6. The x-axis shows the wordline number of a block and the y-axis shows the bit error rates of pages on the corresponding wordline (observed after 50K P/E cycles). Each wordline contains four pages, including LSB-even, LSB-odd, MSB-even, and MSB-odd. The bit error rates of these four types of pages are shown in Figure 6. Several major observations are in order.

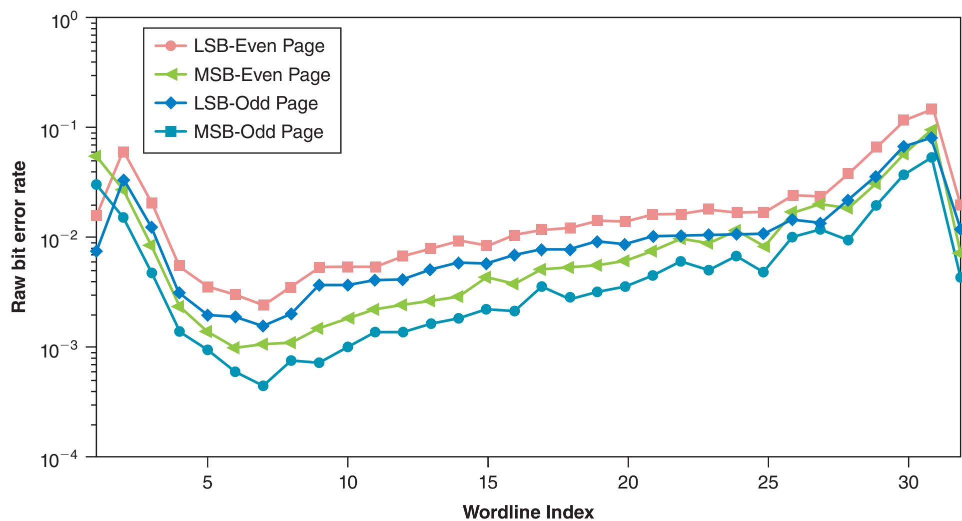


Figure 6: Retention error rate vs. physical location
(Source: Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, 2012[3])

First, the error rate of the MSB page is higher than that of the corresponding LSB page. In our experimental data, the MSB-even page error rate is 1.88 times higher than the LSB-even page error rate and the MSB-odd page error rate is 1.67 times higher than the LSB-odd page error rate on average. This phenomenon can be explained by understanding the bit mapping within the flash memory. Dominant retention errors are mainly due to the shifting of V_{th} between two adjacent threshold voltage levels, that is shifting of V_{th} from the ith level to the (i–1)th level. From the bit mapping in Figure 1, we can see that such a V_{th} shift can cause an LSB error only at the border REF2 between state L2 (01) and state L1 (10) because these are the only two adjacent threshold voltage levels where LSB differs. On the other hand, such a V_{th} shift can cause an MSB error on any border (REF1, REF2, REF3) between any two adjacent states because MSB differs between all possible adjacent threshold voltage levels. Hence, since the likelihood of a change in MSB when a V_{th} shift happens between adjacent states is higher than the likelihood of a change in LSB, it is more common to see retention errors in MSB than in LSB.

"...the error rate of the MSB page is higher than that of the corresponding LSB page."

Second, the retention error rate of odd pages is always higher than that of the corresponding even pages. For example, the error rate of MSB-odd pages is 2.4 times higher than that of MSB-even pages, and the error rate of LSB-odd pages is 1.61 times higher than that of LSB-even pages, on average. This result can be explained by the over-programming introduced by inter-page interference. Generally, the pages inside a flash block are programmed sequentially, and a block is programmed in order, that is, from page 0 to page 127. For the same wordline, even pages are programmed first followed by odd pages. When odd pages are programmed, a high positive program voltage is applied to the control gates of all the cells on the wordline, including the cells of the even page, which has already been programmed. Thus, the even page comes under programming current disturbance and some additional electrons could be attracted into the floating gates of the even page. As a result of this, the V_{th} of cells of the even pages shift slightly to the right. Consequently, the cells of the even pages hold more electrons than the cells of the odd pages, even if they are programmed to the same logic value and are in the same threshold voltage window (in some sense, the cells of the even pages are thus more resistant to leakage because they hold more electrons). When electrons leak away over time during the retention test, as a result, it is more likely for the cells of even pages to still keep their original threshold voltage window and hold the correct value. In contrast, since the cells of the odd pages hold fewer electrons, they are more likely to transition to a different threshold voltage window and hence acquire an incorrect value as electrons leak over time.

"...the retention error rate of odd pages is always higher than that of the corresponding even pages."

Third, the bit error rates of all the four types of pages have the same trend related to physical wordlines. For example, the error rates of the four types of pages are all high on wordline #31 and are all low on wordline #7. We conclude that error rates are correlated with wordline locations. This could possibly be due to process variation effects, which could be similar across the same wordline.

"The major takeaway from our measurement and characterization results is that the rate of retention errors, which are the most common form of flash errors, is asymmetric in both original cell value and the location of the cell in flash bit organization."

"The basic idea of the FCR schemes is to periodically read, correct, and refresh (reprogram or remap) the stored data before it accumulates more retention errors than can be handled by ECC."

The major takeaway from our measurement and characterization results is that the rate of retention errors, which are the most common form of flash errors, is asymmetric in both original cell value and the location of the cell in flash bit organization. This observation can potentially be used to devise error protection or correction mechanisms that have varying strength based on cell value and location.

Mitigating Retention Errors: Flash Correct-and-Refresh

We describe a set of new techniques, called Flash Correct-and-Refresh (FCR), that exploit the dominance and characteristics of retention errors to significantly increase NAND flash lifetime while incurring minimal overhead. The basic idea of the FCR schemes is to periodically read, correct, and refresh (reprogram or remap) the stored data before it accumulates more retention errors than can be handled by ECC. Thus, we can achieve a low uncorrectable bit error rate (UBER) while still using a simple, low-overhead ECC. Two key questions central to designing a system that uses FCR techniques are: (1) *how* to refresh the data in flash memory and (2) *when* to refresh the data. We address the first question with two techniques for how to refresh the data: remapping (in the section "Remapping-based FCR Mechanisms") and reprogramming in-place (in the section "In-Place Reprogramming-based FCR Mechanisms"). We then tackle the second question with two techniques for when to refresh: periodically and adaptively based on the number of P/E cycles (Section 6.3).

Remapping-based FCR Mechanisms

Unlike DRAM cells, which can be refreshed in-place[13], flash cells generally must first be erased before they can be programmed. To remove the slow erase operation from the critical path of write operations, current wear-leveling algorithms remap the data to another physical location rather than erasing the data and then programming in-place. The flash controller maintains a list of free blocks that have been erased in background through garbage collection and are ready for programming. Whenever a write operation is requested, the controller's wear-leveling algorithm selects a free block and programs it directly, remapping the logical block address to the new physical block.

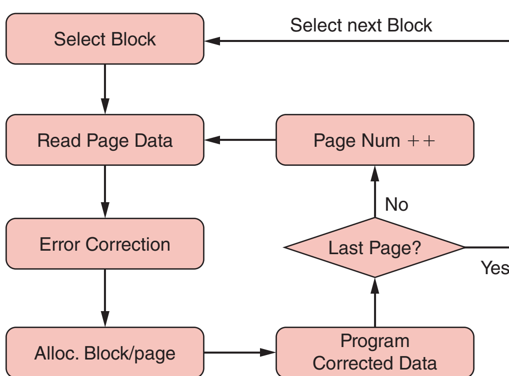


Figure 7: Operation of a remapping-based flash correct-and-refresh scheme (Source: Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai, 2012[4])

The key idea of remapping-based FCR is to leverage the existing wear-leveling mechanisms to periodically read, correct, and remap to a different physical location each valid flash block in order to prevent it from accumulating too many retention errors. Figure 7 shows the operational flow of remapping-based FCR: (1) During each refresh interval, a block with valid data that needs to be refreshed is selected. (2) The valid data in the selected block is read out page by page and moved to the SSD controller. (3) The ECC engine in the SSD controller corrects all the errors in the read data, including retention errors that have accumulated since the last refresh. After ECC, the data are error free. (4) A new free block is selected and the

error free data are programmed to the new location, and the logical address is remapped. Note that the proposed address remapping techniques leverage existing hardware and software of contemporary wear-leveling and garbage collection algorithms.

Unfortunately, periodic remapping of every block introduces additional erase cycles. This is because after the flash data are corrected and remapped to the new location, the original block is marked as outdated. Thus, the block will eventually be erased and reclaimed by garbage collection. The more frequent the remap operations, the more the additional erase operations, which wears out flash memory faster. As such, there might be an inflection point beyond which increasing the refresh rate in remapping-based FCR can lead to reduced lifetime. To avoid this potential problem, we next introduce enhanced FCR methods, which minimize unnecessary remap operations.

In-Place Reprogramming-based FCR Mechanisms

To reduce the overhead associated with periodic remapping, we describe a technique for periodic *in-place reprogramming* of the block most of the time, without a preceding erase operation, which can greatly reduce the overhead of periodic remapping. This in-place reprogramming takes advantage of the key observation that retention errors arise from the loss of electrons on the floating gate over time and *the flash cell with retention errors can be reprogrammed to its original correct value without an erase operation* using the incremental step pulse programming (ISPP) scheme used to program flash memory. We first provide background on ISPP.

"...the flash cell with retention errors can be reprogrammed to its original correct value without an erase operation *using the incremental step pulse programming (ISPP) scheme used to program flash memory.*"

ISPP

Before a flash cell can be programmed, the cell must be erased (that is, all charge is removed from the floating gate, setting the threshold voltage to the lowest value). When a NAND flash memory cell is programmed, a high positive voltage applied to the control gate causes electrons to be injected into the floating gate. The threshold voltage of a NAND flash cell is programmed by injecting a precise amount of charge onto the floating gate through ISPP.[11] During ISPP, floating gates are programmed iteratively using a step-by-step program-and-verify approach. After each programming step, the flash cell threshold voltage is boosted up. Then, the threshold voltage of the programmed cells are sensed and compared to the target values. If the cell's threshold voltage level is higher than the target value, the program-and-verify iteration will stop. Otherwise the flash cells are programmed once again and more electrons are added to the floating gates to boost the threshold voltage. This program-and-verify cycle continues iteratively until all the cells' threshold voltages reach the target values. Using ISPP, flash memory cells can only be programmed from a state with fewer electrons to a state with more electrons and cannot be programmed in the opposite direction.

Retention Error Mechanisms

Retention errors are caused by the loss of electrons from the floating gate over time. As such, a cell with retention errors moves from a state with more electrons to a state with fewer electrons. Figure 8(a) shows the relative relationship between the stored data value and its corresponding threshold voltage distribution for a typical MLC flash storing 2-bits per cell. The leftmost state is the erased state (state 11) with the smallest threshold voltage, and there is no charge on the floating gate. The states located on the right in Figure 8(a) are programmed with more electrons and have higher threshold voltages than the states located relatively to the left. Over time, as the electrons on the floating gate leak away, the threshold voltage of a cell shifts to the left, as shown in Figure 8(b). If the threshold voltage of a cell shifts too far to the left (that is, it loses too many electrons from the floating gate), it will cross the read reference voltage between adjacent states and can be misinterpreted during a read as the wrong value.

"Over time, as the electrons on the floating gate leak away, the threshold voltage of a cell shifts to the left..."

In-Place Reprogramming Can Fix Retention Errors

A cell with a retention error can be reprogrammed to the value it had before the floating gate lost charge by recharging additional electrons onto the floating gate through ISPP, as shown in Figure 8(c). Note that this does not require an erase operation because the only objective is to add more electrons (not to remove them), which can be accomplished by simple programming.

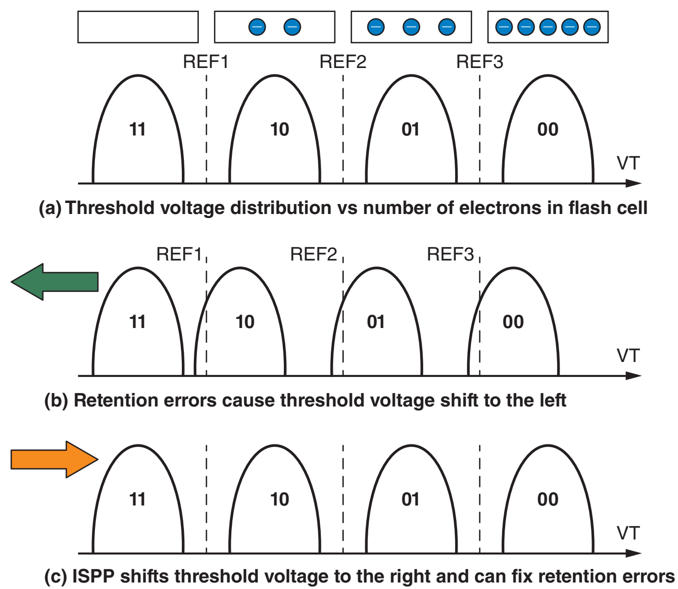


Figure 8: Retention errors are caused by threshold voltage shift to the left and can be fixed by programming in-place using ISPP
(Source: Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai, 2012[4])

6.2.4 Basic In-Place Reprogramming-based FCR Mechanism

A basic FCR mechanism that uses in-place reprogramming works as follows. Periodically, a block is selected to be refreshed and read page-by-page into the flash controller. By selecting a suitable refresh interval, we can ensure that the total error number is below the correction capability of the ECC. Then we can reprogram the flash cells in the same location with the error-corrected data, without erasing the whole block. If the new corrected value corresponds to a state with more charge than the old value, then the cell can be in-place reprogrammed to the correct value. If the corrected value is exactly the same as the original value, in-place reprogramming will not change the stored data value, as ISPP will stop programming the cell as soon as it detects that the target value has already been reached. Note that most of the cells are reprogrammed with exactly the same data value as error rates are generally significantly below 1 percent.

Problem: Accumulated Program Errors

While this basic mechanism can effectively fix retention errors, it introduces a problem because there is another error mechanism in flash cells that is caused by program operations, which are required to perform in-place reprogramming. When a flash cell is being programmed, additional electrons may be injected into the floating gates of its neighbor cells due to coupling capacitance.[14] The threshold voltage distribution of the neighbor cells will shift *right* as they gain more electrons, as shown in Figure 9(a). If the threshold voltage shifts right by too much, it will be misread as an error value that represents a state located to the right. This is called a program interference error (or simply a program error). Although it is a less common error mechanism than retention errors as we have shown in Figure 4, periodic reprogramming can exacerbate the effects of program errors.

"When a flash cell is being programmed, additional electrons may be injected into the floating gates of its neighbor cells due to coupling capacitance."

Two potential issues are: (1) As ISPP cannot remove electrons from the floating gate, program errors cannot be fixed by in-place reprogramming; (2) Reprogramming of a page can introduce additional program errors due to the additional program operations. Figure 9(b) illustrates both issues in the context of in-place programming. First, the original data is programmed into the page. This initial programming can cause some program errors (for example, value 11 is programmed as 10 on the second cell from the left). After some time, retention errors start to appear in the stored data (for example, the first cell changes from state 00 to 01). Note that there are generally many more retention errors than program errors. When the page is reprogrammed in-place, it is first read out and corrected using ECC. The error-corrected data (which is the same as the original data) is then written back (programmed) into the page. This corrects all the retention errors by recharging the cells that lost charge. However, this reprogramming does not correct the program error (in the second cell) because this correction requires the removal of charge from the second cell's floating gate, which is not possible without an erase operation. Furthermore, additional program errors can appear (for example, in the sixth cell) because the in-place program operation can cause additional disturbance.

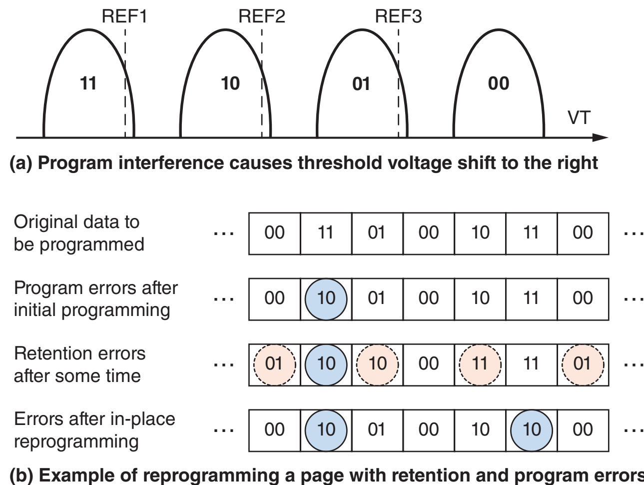


Figure 9: In-place reprograming can correct retention errors but not program errors because in-place programming can only add more electrons into the floating gate and cannot remove them. Note that red values with dotted circles are retention errors and blue ones with solid circles are program errors
(Source: Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai, 2012[4])

"To mitigate the errors accumulated due to periodic reprogramming, we propose a hybrid reprogramming/remapping-based FCR technique to control the number of reprogram errors."

Hybrid FCR

To mitigate the errors accumulated due to periodic reprogramming, we propose a hybrid reprogramming/remapping-based FCR technique to control the number of reprogram errors. The key idea is to monitor the right-shift error count present in each block. If this count is below a certain threshold (likely most of the time) then in-place reprogramming is used to correct retention errors. If the count exceeds the threshold, indicating that the block has too many accumulated program errors, then the block is remapped to another location, which corrects both retention and program errors. In our evaluation, we set the threshold to 30 percent of the maximum number of errors that could be corrected by ECC, which is conservative. Figure 10 provides a flowchart of this hybrid FCR mechanism. Note that this hybrid FCR mechanism greatly reduces the additional erase operations present in remapping based FCR because it remaps a block (requires an erase operation) only when the number of accumulated program errors is high, which is rare due to the low program error rate.

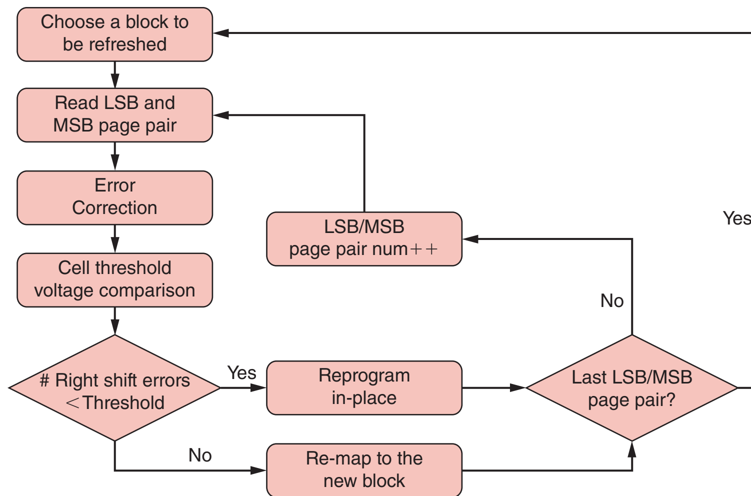


Figure 10: Hybrid FCR workflow: if reprogramming error count is less than a threshold, in-place reprogram the block; otherwise, remap to a new block
(Source: Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai, 2012[4])

Adaptive-Rate FCR

So far we assumed that FCR mechanisms, be it based on in-place reprogramming or remapping, are invoked periodically. This need not be the case. In fact, we observe that the rate of (retention) errors is very low during the beginning of flash lifetime, as shown in Figure 4. Until more than 1000 P/E cycles, the retention error rate is lower than the acceptable raw BER that can be corrected by the simplest BCH code (not shown, but described in detail in [4]), which is a commonly used ECC type in flash memories. Hence, at the beginning of its lifetime, flash memory does not need to be refreshed. Retention error rate increases as the number of P/E cycles increases. We leverage this key observation to reduce the number of unnecessary refresh operations.

The main idea of adaptive-rate FCR is to adapt the refresh rate to the number of P/E cycles a block has incurred. Initially, refresh rate for a block starts out at zero (no refresh). Once ECC becomes incapable of correcting retention errors, the block's refresh rate increases to tolerate the increased retention error rate. Hence, refresh rate is gradually increased over each flash block's lifetime to adapt to the increased P/E cycles. The whole lifetime of a flash block can be divided into intervals with different refresh rates ranging, for example, from no refresh (initially), yearly refresh, monthly refresh, weekly refresh, to daily refresh. The frequency of refresh operations at a given P/E cycle count is determined by the acceptable raw BER provided by the used ECC and the BER that corresponds to the P/E cycle count (which can be known by the controller[4]). Note that this mechanism requires keeping track of P/E cycles incurred for each block, but this information is already maintained to implement current wear-leveling algorithms.

"...refresh rate is gradually increased over each flash block's lifetime to adapt to the increased P/E cycles."

Additional Considerations

We briefly discuss some additional factors that affect the implementation and operation of the proposed FCR mechanisms.

Implementation Cost

"The FCR mechanisms do not require hardware changes."

The FCR mechanisms do not require hardware changes. They require changes in FTL software/firmware to implement the flowcharts shown in Figures 7 and 10. FCR can leverage the per-block validity and P/E cycle information that is already maintained in existing flash systems to implement wear leveling.

Power Supply Continuity

To perform a refresh, the flash memory must be powered. As FCR is proposed mainly for enterprise storage applications, these systems are typically continuously powered on. Our proposed techniques use daily, weekly, or monthly refresh and it is rare for a server to be powered off for such long periods.

Response Time Impact

"Unlike DRAM, where refresh is triggered frequently (for example, every 64 ms) to maintain correctness, the refresh period of FCR is at least a day..."

Refresh may interfere with normal flash operations and degrade the response time. To reduce this penalty, we can decrease the refresh priority, making it run in the background. The SSD can issue refresh operations whenever it is idle, and refresh operations can be interrupted to avoid the impact on the response time of normal operations. Unlike DRAM, where refresh is triggered frequently (for example, every 64 ms) to maintain correctness[13], the refresh period of FCR is at least a day, and the SSD can finish refresh operations within the refresh period. Recent work has shown that the response time overhead is within a few percent for daily refresh.[15] Note that our hybrid and adaptive FCR techniques have much lower overhead for refresh operations than periodic remapping based FCR.

Additional Erase Cycles

FCR introduces additional erase operations. Our evaluations take into account the impact of additional erase operations on flash lifetime and energy consumption.

Adapting to Variations in Retention Error Rate

Note that retention error rate is usually constant for a given refresh rate and P/E cycle combination. However, there are environmental factors, such as temperature, that can change this rate. For example, retention error rate would be dependent on temperature. To adapt to dynamic fluctuations in retention error rate, our hybrid FCR and adaptive-rate FCR mechanisms monitor the changes in the retention error rate at periodic intervals, and increase or decrease the refresh (that is, FCR) rate if the error rate in the previous interval is greater or less than a threshold. These mechanisms are similar in principle to what is employed in DRAM to adapt refresh rate to temperature changes.[13]

Evaluation of Flash Correct-and-Refresh

We evaluate FCR using Disksim[20] with SSD extensions[21]. All proposed techniques are simulated using various I/O traces from real workloads: *iozone*[22], *cello99*[23], *oltp*, *postmark*[24], *MSR-Cambridge*[25] and a *web search engine*[26]. We configure the simulated flash-based SSD with four channels. Each channel has eight flash chips. Each flash chip has 8,192 blocks containing 128 pages. The page size is 8 KB. The total storage capacity is 256 GB. The energy of flash read, program, and erase operations are collected from our experimental flash memory platform[2], and are used in the simulation infrastructure to obtain the overall energy consumption. The details of our experimental evaluation methodology, workloads, and our method for estimating lifetime are described in our previous work[4]. We present the major results showing the effect of our mechanisms on flash lifetime and energy consumption in this article. Much more detailed analyses of our individual techniques, analysis of sensitivity to refresh interval length, and results on individual workloads are provided in [4].

Effect on Flash Memory Lifetime

Figure 11 shows the lifetime improvement provided by three different versions of FCR compared to the baseline with no refresh. The adaptive-rate FCR mechanism is implemented on top of the hybrid FCR substrate. Flash lifetime is evaluated under various ECC configurations, ranging from weak 512-bit to strong 32-kb BCH codes (described and evaluated in detail in [4]). The refresh period of each periodic mechanism is chosen on a per-workload basis such that the lifetime provided for a workload by the mechanism is maximized (more analysis on the refresh period can be found in [4]). Adaptive-rate FCR, which adaptively and realistically chooses the refresh period, provides the highest lifetime improvement over the baseline as it corrects retention errors while avoiding unnecessary refreshes. The improvements are especially significant in read-intensive workloads since these workloads do not have high P/E cycles, causing the adaptive-rate FCR to keep the refresh rate very low. On average, adaptive-rate FCR provides 46.7x, 4.8x, and 1.5x higher flash lifetime compared to no-refresh (on the baseline system using 512-bit ECC), remapping-based FCR, and hybrid FCR, respectively. Note that the lifetime improvement provided by the much stronger 32-kb ECC is only four times that of the lifetime provided by the baseline 512-bit ECC, yet the implementation of the former, stronger ECC, requires 71 times the power consumption and 85 times the area of the latter, weaker ECC.[4] Contrast this with the 46.7x lifetime improvement provided by adaptive-rate FCR on the system with 512-bit ECC. Thus, improving lifetime via FCR is much more effective and efficient than doing so by increasing the strength of ECC. We conclude, based on these results, that adaptive-rate FCR implemented over the hybrid FCR mechanism is a promising mechanism for significant lifetime enhancement of flash memory at low cost.

"...adaptive-rate FCR provides 46.7x, 4.8x, and 1.5x higher flash lifetime compared to no-refresh"...

"...improving lifetime via FCR is much more effective and efficient than doing so by increasing the strength of ECC."

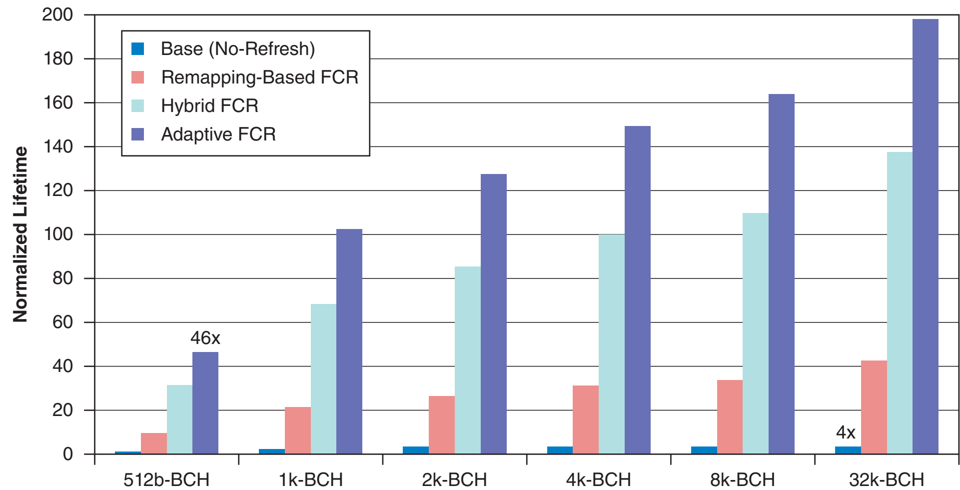


Figure 11: Lifetime improvement provided by different FCR techniques in comparison to systems employing varying strength ECC (BCH) codes. Data normalized to lifetime with no refresh on a system with 512-bit ECC
(Source: Onur Mutlu, 2012)

P/E Cycle and Energy Overhead Analysis

FCR techniques can introduce two main overheads: (1) additional P/E cycles due to remapping; (2) additional energy consumed by refresh operations. A detailed evaluation of the former is provided in [4]. Note that all P/E cycle overheads have already been accounted for in the collection of the flash lifetime results.

Figure 12 shows the additional flash energy consumption of remapping-based FCR and hybrid FCR averaged over all workloads compared to a system with no FCR. The refresh energy is estimated under the worst-case scenario that all data are to be refreshed. Even if we assume we must refresh the *entire* SSD each day, the energy overhead is only 7.8 percent and 5.5 percent for remapping-based FCR and hybrid FCR respectively. When the refresh interval is three weeks, the energy overhead is almost negligible (less than 0.4 percent). We also observe that hybrid FCR has less energy overhead than remapping based FCR mainly because hybrid FCR reduces the high-energy erase/remap operations by performing in-place reprogramming most of the time.

"When the refresh interval is three weeks, the energy overhead is almost negligible..."

We also evaluate the energy overhead of adaptive-rate FCR and find that it is only 1.5 percent (not shown in the figure). Recall that adaptive-rate FCR starts out with no refresh and gradually increases the refresh rate up to daily refresh as the P/E cycles accumulate. Yet its energy overhead is significantly lower than periodic daily refresh. We conclude that adaptive-rate FCR is the most superior of flash correct-and-refresh mechanisms in terms of both lifetime and energy consumption.

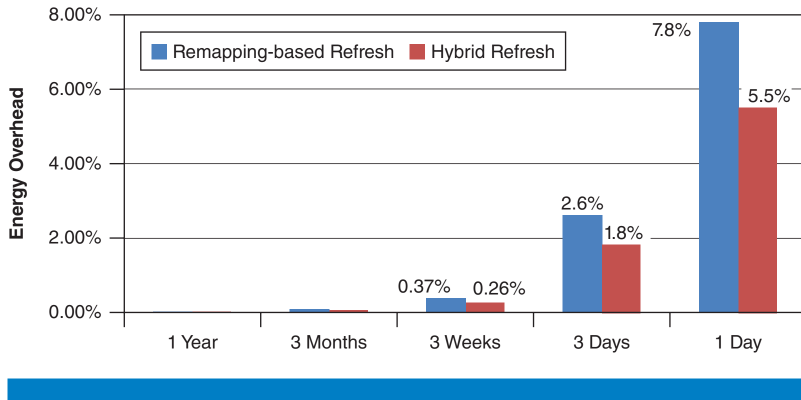


Figure 12: Energy increase of remapping-based and hybrid FCR vs. no refresh
(Source: Onur Mutlu, 2012)

Ongoing Work

In our comprehensive continued effort for enhancing flash memory scaling to smaller technology nodes, we have been characterizing the effects of different error mechanisms in flash memory, developing models to predict how they change over the lifetime of flash memory, and designing error tolerance mechanisms based on the developed characterization and models.

Recently, we have also experimentally investigated and characterized the threshold voltage distribution of different logical states in MLC NAND flash memory.[6] We have developed new models that can predict the shifts in the threshold voltage distribution based on the number of P/E cycles endured by flash memory cells. Our key results, presented in [6], show that 1) the threshold voltage distribution of flash cells that store the same value can be approximated, with reasonable accuracy, as a Gaussian distribution, 2) under ideal wear leveling, the flash cell can be modeled as an AWGN (Additive White Gaussian Noise) channel that takes the input (programmed) threshold voltage signal and outputs a threshold voltage signal with added Gaussian white noise, and 3) threshold voltage distribution of flash cells that store the same value gets distorted (shifts to right and widens around the mean value) as the number of P/E cycles increases. This distortion can be accurately modeled and predicted as an exponential function of the P/E cycles, with more than 95 percent accuracy. Such predictive models can aid the design of much more sophisticated error correction methods, such as LDPC codes[7], which are likely needed for reliable operation of future flash memories. We refer the reader to [6] for more detail.

"We have developed new models that can predict the shifts in the threshold voltage distribution based on the number of P/E cycles endured by flash memory cells."

We are currently investigating another increasingly more significant obstacle to continued MLC NAND flash scaling, which is the increasing cell-to-cell program interference due to increasing parasitic capacitances between the cells' floating gates. Accurate characterization and modeling of this phenomenon are needed to find effective techniques to combat this program interference. In our recent work[16], we leverage the *read retry*

mechanism found in some flash designs to obtain measured threshold voltage distributions results from state-of-the-art 2Y-nm (24- to 20-nm) MLC NAND flash chips. These results are then used to characterize the cell-to-cell program interference under various programming conditions. We show that program interference can be accurately modeled as additive noise following Gaussian-mixture distributions, which can be predicted with 96.8 percent accuracy using linear regression models. We use these models to develop and evaluate a read reference voltage prediction technique that reduces the raw flash bit error rate by 64 percent and increases the flash lifespan by 30 percent. We refer the reader to [16] for more detail.

"A key direction is to co-design the DRAM controller and DRAM, rethinking the DRAM interface and microarchitecture, such that DRAM scaling challenges are tolerated at the system level."

Finally, apart from investigating scaling challenges in flash memory, we are investigating techniques to enable better scaling of DRAM. Improving DRAM cell density by reducing the cell size, as has been done traditionally, is becoming significantly more difficult due to increased manufacturing complexity/cost and reduced cell reliability. We are examining alternative ways of enhancing the performance and energy-efficiency of DRAM while still maintaining low cost. A key direction is to co-design the DRAM controller and DRAM, rethinking the DRAM interface and microarchitecture, such that DRAM scaling challenges are tolerated at the system level. For example, we have recently proposed new techniques to reduce DRAM access latency at low cost by segmenting bitlines and creating a low-latency low-energy segment within a subbank[17], to increase DRAM parallelism and locality by enabling pipelined access of subbanks and enabling multiple row buffers to be concurrently active within a bank[18], to reduce the number of DRAM refreshes by taking advantage of variation in retention times of DRAM rows in a low-cost manner[13], and to accelerate bulk data copy and initialization operations by performing them solely in DRAM with only minor modifications to DRAM[19]. We have also experimentally characterized retention behavior of DRAM cells and rows for 248 commodity DRAM chips[27], with the goal of developing mechanisms that can dynamically profile retention times of different rows. We observed two significant phenomena: data pattern dependence, where the retention time of DRAM cells is significantly affected by the data stored in other DRAM cells, and variable retention time, where the retention time of some DRAM cells changes over time. We refer the reader to these respective works for further detail.

Conclusion

Reliability and energy efficiency challenges posed by technology scaling are a critical problem that jeopardizes both flash memory and DRAM capacity, cost, performance, lifetime, and efficiency. In this article, we have described our recent error analysis of flash memory and a new method to improve flash memory lifetime. We hope other works by us and other researchers in the field collectively enable the memory and microprocessor industry to develop cooperative techniques to enable scalable, efficient, and reliable flash memory (and DRAM) that continues to scale to smaller feature sizes.

References

[1] A. Maislos et al., "A New Era in Embedded Flash Memory," FMS 2011.

[2] Y. Cai, et al., "FPGA-Based Solid-State Drive Prototyping Platform," FCCM 2011.

[3] Y. Cai et al., "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization and Analysis," DATE 2012.

[4] Y. Cai et al., "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime," ICCD 2012.

[5] Y. Koh, "NAND Flash Scaling Beyond 20nm," IMW 2009.

[6] Y. Cai et al., "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis and Modeling," DATE 2013.

[7] R. G. Gallager, *Low-Density Parity Check Codes*. Cambridge: MIT Press, 1963.

[8] T. Hara, et al., "A 146-mm2 8-Gb multi-level NAND flash memory with 70-nm CMOS technology," JSSC, Vol. 41, pp. 161–169, 2006.

[9] Y. Li, et al., "A 16Gb 3-Bit Per Cell(X3) NAND Flash Memory on 56nm Technology With 8MB/s Write Rate," JSSC, Vol. 44, pp. 195–207, 2009.

[10] N. Shibata, et al., "A 70nm 16Gb 16-Level-Cell NAND flash Memory," JSSC, Vol. 43, pp. 929–937, 2008.

[11] K. D. Suh, et al., "A 3.3V 32Mb NAND Flash Memory with Incremental Step Pulse Program Scheme," JSSC, Vol. 30, No.11, pp. 1149–1156, 1995.

[12] M. Xu, et al., "Extended Arrhenius law of time-to-breakdown of ultrathin gate oxides," Applied Physics Letters, Vol. 82, pp. 2482–2484, 2003.

[13] J. Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

[14] K. Park et al., "A Zeroing Cell-to-Cell Interference Architecture with Temporary LSB Storing and Parallel MSB Program Scheme for MLC NAND Flash Memories," JSSC 2008.

[15] Y. Pan et al., "Quasi-Nonvolatile SSD: Trading Flash Memory Nonvolatility to Improve Storage System Performance for Enterprise Applications," HPCA 2012.

[16] Y. Cai et al. "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Application," Carnegie Mellon University SAFARI Technical Report, January 2013.

[17] D. Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.

[18] Y. Kim et al., "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," ISCA 2012.

[19] V. Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," Carnegie Mellon University SAFARI Technical Report, March 2013.

[20] J. Bucy et al., "DiskSim Simulation Environment Reference Manual," 2008.

[21] N. Agrawal et al., "Design Tradeoffs for SSD Performance," USENIX 2008.

[22] IOzone.org, "IOzone Filesystem Benchmark," http://iozone.org.

[23] Open Source software at HP Labs, http://tesla.hpl.hp.com/opensource.

[24] J. Katcher, "Postmark: a New File System Benchmark Technical Report," 1997.

[25] SNIA: IOTTA Repository, http://iotta.snia.org/tracetypes/3.

[26] UMass Trace: http://traces.cs.umass.edu/index.php/Storage/Storage.

[27] J. Liu et al., "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," To Appear in ISCA 2013.

Author Biographies

Yu Cai obtained his PhD degrees in Electrical and Computer Engineering from Carnegie Mellon University (2012). He received an MS degree in Electronic Engineering from Tsinghua University and a BS degree in Telecommunication Engineering in Beijing University of Posts and Telecommunication (BUPT). Currently, he is a staff engineer, SSD Architect working in the Flash Channel Department of LSI Corporation. Prior to LSI, he worked at Hong Kong Applied Science and Research Institute (ASTRI), Lucent Technology, and Microsoft Research Asia (MSRA). His research interests include data storage, reconfigurable computing, and wireless communication.

Gulay Yalcin is a PhD student at Universitat Politecnica de Catalunya and a researcher student in Bacelona Supercomputing Center. She holds a BS degree in Computer Engineering from Hacettepe University and an MS degree in Computer Engineering from TOBB University of Economics and Technology. Her research interests are reliability and energy minimization in computer architecture. For more information please see the web page at http://www.bscmsrc.eu/people/gulay-yalcin.

Onur Mutlu is the Dr. William D. and Nancy W. Strecker Early Career Professor at Carnegie Mellon University. He enjoys teaching and researching important and relevant problems in computer architecture and computer systems, including problems related to the design of memory systems, multi-core architectures, and scalable and efficient systems. He obtained his PhD and MS in ECE from the University of Texas at Austin (2006) and BS degrees in Computer Engineering and Psychology from the University of Michigan, Ann Arbor. Prior to Carnegie Mellon, he worked at Microsoft Research (2006-2009), Intel Corporation, and Advanced Micro Devices. He was a recent recipient of the IEEE Computer Society Young Computer Architect Award, CMU College of Engineering George Tallman Ladd Research Award, Intel Early Career Faculty Honor Award, Microsoft Gold Star Award, best paper awards at ASPLOS, VTS and ICCD, and a number of "computer architecture top pick" paper selections by the IEEE Micro magazine. For more information, please see his web page at http://www.ece.cmu.edu/~omutlu.

Erich F. Haratsch is Director of Engineering, Flash Channel Technology at LSI Corporation. In this role, he leads the development of advanced signal processing and error correction coding features for solid-state disk controllers. Prior to joining LSI, Haratsch was a Senior Member of Technical Staff at Agere Systems, where he developed signal processing architectures for magnetic recording in hard disk drives. He also developed equalizer and decoder architectures for Gigabit Ethernet over copper and optical communications at Bell Labs Research. Haratsch is the author of more than 30 peer-reviewed journal and conference papers, and holds 35 U.S. patents. He is a Senior Member of IEEE. Haratsch earned his MS and PhD from the Technical University of Munich, Germany.

Adrián Cristal is co-manager of the Computer Architecture for Parallel Paradigms research group at BSC. His interests include high-performance microarchitecture, multi- and many-core chip multiprocessors, transactional memory, programming models, and computer architectures for Big Data. He received a PhD from the Computer Architecture Department at the Polytechnic University of Catalonia (UPC), Spain, and he has a BS and an MS in computer science from the University of Buenos Aires, Argentina.

Osman Unsal received the BS, MS, and PhD degrees in Electrical and Computer Engineering from Istanbul Technical University (Turkey), Brown University (USA) and University of Massachusetts, Amherst (USA) respectively. Together with Dr. Adrian Cristal, he co-manages the Computer Architecture for Parallel Paradigms research group at BSC. His current research interests include many-core computer architecture, reliability, low-power computing, programming models and transactional memory.

Ken Mai received his BS, MS, and PhD degrees in electrical engineering from Stanford University in 1993, 1997, and 2005, respectively. He joined the Faculty of Carnegie Mellon University in 2005 as an Assistant Professor in the Electrical and Computer Engineering Department. His research interests are in high-performance circuit design, secure IC design, reconfigurable computing, and computer architecture. He was the recipient of an NSF CAREER award in 2007 and the George Tallman Ladd Research Award in 2008.

A CASE FOR NONUNIFORM FAULT TOLERANCE IN EMERGING MEMORIES

Contributor

Moinuddin K. Qureshi
Georgia Institute of Technology

As DRAM systems face scalability challenges, the architecture community has started investigating alternative technologies for main memory. These emerging memory technologies tend to suffer from the problem of limited write endurance. This problem is exacerbated because of the high variability in lifetime across different cells, resulting in weaker cells failing much earlier than nominal cells. Ensuring long lifetimes under high variability requires that the design can correct a large number of errors for any given memory line. Unfortunately, supporting high levels of error correction for all lines incurs significantly high overhead, often exceeding 10 percent of overall memory capacity. We propose to reduce the storage required for error correction by exploiting the observation that only a few lines require high levels of hard-error correction. Therefore, prior approaches that uniformly allocated a large number of error correction entries for all lines are inefficient, as most (more than 90 percent) of these entries remain unused. We propose Pay-As-You-Go (PAYG), an efficient hard-error resilient architecture that allocates error correction entries in proportion to the number of hard faults in the line. We describe a storage-efficient and low-latency organization for PAYG. Compared to uniform error correction, PAYG requires one third the storage overhead and yet provides 13 percent more lifetime.

Introduction

As DRAM-based memory systems get limited by power and scalability challenges, architects are turning their attention towards emerging memory technologies for building future systems. Phase Change Memory (PCM) has emerged as one of the most promising technologies suitable for incorporation into main memory.[3] While PCM has several desirable attributes such as improved scalability and nonvolatility, the physical properties of PCM dictates that only a limited number of writes can be performed to each cell. On average, PCM devices are expected to last for about 10 to the 7th and 10 to the 8th, writes per cell.[1] Once a cell reaches its end of life, it gets stuck in one of the states, manifesting itself as a hard error. The problem of limited lifetime is further exacerbated by the high variability in lifetime across different cells due to process variations. This means a small percentage of cells that have a significantly lower than average lifetime end up determining the overall lifetime of the system.

"…a small percentage of cells that have a significantly lower than average lifetime end up determining the overall lifetime of the system."

Ensuring reasonable system lifetime under high variability requires that the design provision large amounts of error correction for PCM lines. As we are concerned with lifetime failures that manifest themselves as hard errors, we focus only on hard-error correction in this article. Recent studies have proposed write-efficient error correction schemes such as *Error Correction*

Pointers (ECP)[5] and *SAFER*[6] to tolerate a large number of hard faults in memory lines. While our analysis is applicable to any hard-error correction scheme, we discuss ECP for our studies owing to its simplicity.

ECP corrects a failed bit in a memory line by recording the position of the bit in the line and its correct value. For example, a 64-byte (512-bit) line needs a 9-bit pointer plus 1 replacement bit resulting in a total of 10 bits for each ECP entry. Our evaluations show that correcting six errors per line can provide a lifetime of about 6.5 years for our baseline (the configuration is described in the section "Experimental Methodology"). Provisioning for 6 bits of error correction requires an overhead of 61 bits (60 bits of ECP plus one full bit to indicate that all ECP entries are used) per line, which translates to a total storage overhead of 12 percent. Note that this level of error correction would not be an optional feature in future PCM systems but rather something that would be essential to enable meaningful operation of the PCM array. Given that the memory market is low margin and highly cost-sensitive, it is important that the storage overhead of such necessary error correction be minimized, while retaining the desired levels of reliability. Thus, the 12 percent storage overhead of ECP may very well prove to be too high for wide-scale adoption of PCM.

To reduce the storage overhead of error correction, we begin by pointing to the inefficiency with the ECP approach that uniformly allocates six ECP entries per line. Our analysis shows that very few lines are weak, and more than 95 percent of the lines require no more than one ECP entry per line. Therefore, we would expect that with uniform ECP-6, the majority of the ECP entries would remain unused. Table 1 shows the distribution of lines that use a given number of ECP entries at different aging levels (age normalized to the lifetime under ECP-6, or 6.5 years). The average number of ECP entries used is also shown.

"Our analysis shows that very few lines are weak, and more than 95 percent of the lines require no more than one ECP entry per line."

Number Writes (Normalized Age)	**Number of ECP Entries Used per Line**				**Average Number of ECP Entries Used**
	0	**1**	**2**	**3–6**	
50%	99.02%	0.97%	0.00%	0.00%	0.010
90%	84.76%	14.02%	1.16%	0.07%	0.165
95%	79.63%	18.14%	2.06%	0.17%	0.228
100%	73.24%	22.82%	3.55%	0.40%	0.311

Table 1: Inefficiency of Uniform ECP-6. On average, only 0.3 out of six entries eventually gets used
(Source: Moinuddin K. Qureshi, 2013)

As the number of writes increase, the rate of faults increases, and hence more and more of the allocated ECP entries get used. However, even at the end of the expected system lifetime under ECP-6, less than 5 percent of the lines utilize more than one ECP entry. On average, only 0.3 entries out of the allocated six entries of ECP get used, indicating significant inefficiency with

uniform ECP. If we could allocate ECP entries only to lines that need those entries, we would reduce the required ECP entries by almost 20X. Ideally, we want to allocate more ECP entries to weak lines (lines with large number of errors) and fewer ECP entries to other lines. Unfortunately, uniform ECP allocates a large (and wasteful) number of ECP entries with each line a priori, being agnostic of the variability in lifetime of each line.

"…Pay-As-You-Go (PAYG, pronounced as "page"), an error correction architecture that allocates error correction entries in response to the number of errors in the given memory line."

We propose Pay-As-You-Go (PAYG, pronounced as "page"), an error correction architecture that allocates error correction entries in response to the number of errors in the given memory line. To maintain low latency of error correction, PAYG splits the correction entries into two parts: first, a per-line Local Error Correction (LEC) that can correct up to one error per line and is sufficient for 95 percent of the lines; and second, a Global Error Correction (GEC) pool that contains tagged ECP entries and provides error correction entries for lines that have more errors than can be handled by the LEC.

We describe several versions of PAYG, each with varying effectiveness, storage overhead, and latency overhead. Our evaluations show that PAYG reduces the storage overhead of error correction by a factor of 3.1X compared to ECP-6 (19.5 bits per line vs. 61 bits per line) while still obtaining 13 percent longer lifetime. Thus, PAYG obtains the best of both worlds in that it achieves the lifetime corresponding to strong levels of error correction while maintaining the low storage overhead that is sufficient for most of the lines.

Background

The problem of limited write endurance is common to many of the emerging memory technologies. Without loss of generality, this article analyzes Phase Change Memory (PCM) as an example of emerging memory technology. PCM suffers from the limited endurance in that the memory cells cease to have the ability to store data after a certain number of writes. Such cells get stuck to one of the states and manifest themselves as hard errors.[5] Designing a robust PCM system that can last for several years requires carefully architecting the system to tolerate such errors.

Problem: Variability in Lifetime

ITRS[1] projections (and various other studies) indicate that PCM cells can be expected to have an average endurance in the range of 10^7–10^8 writes. While this range of endurance is much lower than the ~10^{15} endurance of DRAM, it is still sufficient to architect a system with several (more than five) years of lifetime. Unfortunately, the lifetime of PCM cells is not uniform, and process variability results in significant variations even within adjacent cells in the same die.[2][5] This causes certain cells to have much lower endurance than the average population. Such weak cells fail much earlier than the typical cell and can reduce the lifetime of the system significantly to the tune of a few weeks.

The variation in lifetime is typically expressed as normalized standard deviation (COV) around the mean. Previous studies on variability of PCM endurance have used COV values between 10–30 percent of the mean.[2][5][6] In our

analysis, we use a default COV value of 20 percent. With a COV of 20 percent, the cell failure probability at the very start is in the range of 10^{-6}. Given that a typical main memory system contains tens of billions of cells, even this small failure probability would result in several thousand cells having bit failures, which in turn would result in a drastic reduction in the overall system lifetime because of variability.

"…a typical main memory system contains tens of billions of cells, even this small failure probability would result in several thousand cells having bit failures,…"

Prior Work

The lifetime of a PCM system can be increased to a useful range if the system can tolerate errors. Hamming code-based error correction, which is typically employed in memory systems, can tolerate transient errors as well as hard errors. Unfortunately, such codes are write intensive and can further exacerbate the endurance problem in PCM. Fortunately, identifying the endurance-related write failures is easy as it can be done by simply performing a verify read after completing a write.[1] If the two values do not match then the nonmatching bit is likely to be a hard error.

Recent studies[5][6] have focused on developing write-efficient methods to provide error correction of hard faults, relying on this simple detection property of endurance-related failures.

One such proposal is Error Correcting Pointers (ECP).[5] ECP performs error correction by logging bit errors in a given line. For example, for a line of 64 bytes (512 bits), a 9-bit pointer is used to point to the failing bit and an additional bit to indicate the correct value. This scheme can correct one error and is referred to as ECP-1. The concept can be extended to correct multiple bits per line. Intelligent precedence rules allow correction of errors even in the ECP entries. A generalized scheme that can correct N errors per line is called ECP-N. A full bit per line indicates if all the ECP entries associated with the line are used. Thus, the storage overhead of ECP-N is $(10N + 1)$ bits per line.

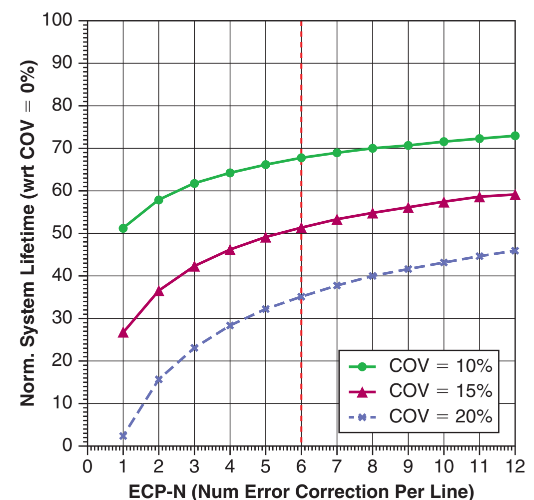


Figure 1: Normalized value of system lifetime (defined as the mean time of first uncorrectable error) as a function of the ECP strategy. The system lifetime is normalized with respect to a memory cell with 0% COV. Note that at COV of 20%, ECP-6 obtains 35% of theoretical maximum lifetime
(Source: Moinuddin K. Qureshi, 2013)

Need to Correct Several Errors per Line

Given that transient faults are rare, a typical memory system is designed to handle at most one or two transient faults per line. However, unlike transient faults, endurance-related hard errors accumulate over time. Therefore, we need to provide large amount of error correction per line in order to obtain reasonable system lifetime. Figure 1 shows the mean time to first uncorrectable error for our baseline system, where the number of ECP entries per line is varied from 1 to 12. All lifetime numbers are normalized to the case of zero variance. To show dependence of lifetime on variance, we show data for different COVs. For COV = 20 percent, ECP-6 obtains 35 percent of ideal

[1] If the system supports some amount of transient fault protection with each line, then we can identify the hard faults without performing the verify read. For example, the position of a bit that causes a failure with a transient fault protection mechanism can be tracked. Given that transient faults are rare, if the same bit position is causing frequent errors then that bit is likely to be a hard fault. Such a bit can then be corrected using a hard-error correction mechanism. This article assumes that an efficient means of detecting endurance-related failures exists and focuses only on correcting such failures.

"ECP-6 incurs storage of 61 bits per line, which translates to 12 percent storage overhead. Given that memory chips are extremely cost sensitive, such overhead may be too high for practical use."

lifetime. For our baseline, this translates to a lifetime of 6.5 years, which is in the desired range of 5–7 years for a typical server. ECP-6 incurs storage of 61 bits per line, which translates to 12 percent storage overhead. Given that memory chips are extremely cost sensitive, such overhead may be too high for practical use.

Inefficiency of Traditional Approach

For a memory of *N* lines, a PCM system would provision a total of 6*N* ECP entries to implement ECP-6. The problem with such an approach is that it results in significantly underutilized ECP entries. Because weak lines are few, only a few lines require high levels of error correction. Most of the other lines do not use the allocated ECP entries. Figure 2 shows the failure probability as the number of writes is increased (under COV = 20 percent), normalized to a system that has zero variance. The failure of line (or system) occurs when there is at least one uncorrectable error for a given amount of ECP. The expected time to failure is computed as the time at which the failure probability is 50 percent.

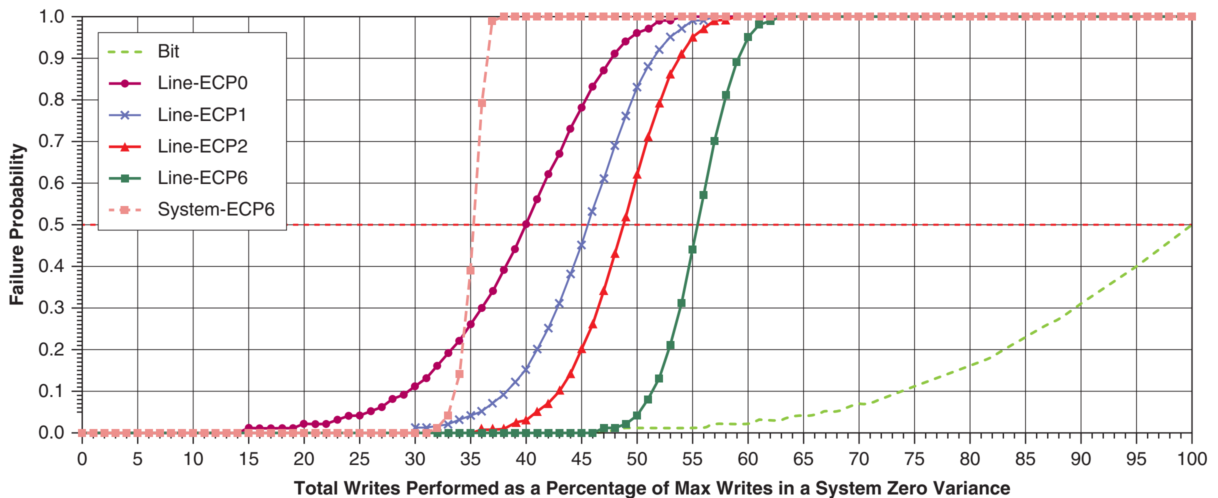


Figure 2: Failure probability vs. the percentage of writes assuming a COV = 20%, normalized to a system that has zero variance. "Bit" shows probability of failure of a single bit. "Line-ECPN" shows the probability of failure of a single line if N bits can be corrected. "System-ECP6" shows the probability that "at least one line fails out of all the lines" when each line has ECP-6. Observe that when the system failure is expected to occur under ECP-6, the probability of line failure with ECP-1 is approximately 3.5%
(Source: Moinuddin K. Qureshi, 2013)

Given that memory has millions of lines, the line failure probability must be very low (much less than 10^{-6}) to achieve a low system failure probability. When the system failure is expected to occur under ECP-6, the probability of line failure with ECP-1 is approximately 3.5 percent. This implies that fewer than 5 percent of the lines have more than one failed bit at the time of system failure, indicating significant inefficiency in the traditional approach that allocates six ECP entries for all lines. We note that ECP-1 is sufficient in the common case, and we need higher levels of ECP for very few lines. Ideally,

we would like to retain the robustness of ECP-6 while paying the hardware overhead of only ECP-1.

We base our solution on the insight that hard errors are quite different from transient faults. We need to allocate the storage for the error detection of transient faults up-front—before the error occurs. However, for hard errors, we can detect the error using a separate mechanism and allocate the error correction entry only when the error occurs. We discuss our experimental methodology before describing our proposal.

Experimental Methodology

The following section describes our experimental methodology.

Baseline Configuration

We assume a memory configuration that is designed with PCM banks each with 1 GB memory. Each bank has one write port and the write operation can be performed with a latency of 1 microsecond. The size of the line in the last-level cache is 64 bytes, which means there are 2^{24} lines in each bank. All operations on memory occur at line-size granularity. Given that each bank is a separate entity and can be written independently, we focus on determining the lifetime of one bank. We assume that each line has an endurance of 2^{25} writes. If endurance variance was 0 percent, we would expect the baseline to have a lifetime of 18 years.[2] ECP-6 obtains 35 percent of this lifetime, which translates to 6.5 years.

Assumptions

We are interested in evaluating the lifetime of memory, which is typically in the range of several years. Modeling a system for such a long time period inevitably involves making some simplifying assumptions. We make the following assumptions in order to evaluate memory lifetime:

- We assume the lifetime of each memory cell to follow a normal distribution without any correlation between neighboring cells. We assume a mean lifetime of 2^{25} writes[4] and a COV of 20 percent of the mean.
- We assume perfect wear-leveling to focus only on the impact of the error correction schemes. This implies that all the memory lines will receive the same number of writes.
- A write request to memory is converted into a sequence of write requests followed by a read request to detect hard faults. We assume that this technique can identify hard faults with 100 percent accuracy.

Figure of Merit

The endurance-limited lifetime of the system can be defined as the number of writes performed before encountering first uncorrectable error. Thus, for a given scheme, lifetime is determined by the first line that gets more errors than

[2] Each of the 224 lines can be written 225 times, for a total of 249 writes. With write latency of 1 microsecond, we can perform 106 writes/second or 244.8 writes per year, hence, a lifetime of 18 years, even under continuous write traffic.

can be corrected. ECP-6 obtains a lifetime of 6.5 years, which is in the range of 5–7 years of lifetime for a typical server. We want a lifetime in this range; hence all lifetime numbers in our evaluation are normalized to ECP-6. We define Normalized Lifetime (NL) as follows, and use this as the figure-of-merit in our evaluations:

$$NL = \frac{\text{Total Line Writes Before System Failure} \times 100\%}{\text{Total Line Writes Before System Failure With ECP 6}} \qquad (1)$$

Pay-As-You-Go Error Correction

"…PAYG, provides storage-efficient on-demand error correction while incurring negligible latency overhead."

We can architect an efficient and robust design by allocating error correction entries only on demand, as and when an error occurs. In fact, one can reduce the percentage of unused ECP entries to zero by having a fully associative structure where each entry contains one tagged ECP-1 unit. Unfortunately, such a design would incur intolerable latency as each memory access would need to search through hundreds of thousands of error-correction entries. Our proposed design, PAYG, provides storage-efficient on-demand error correction while incurring negligible latency overhead. In this section, we first start with a naive design for PAYG, identify its shortcomings, and then propose the robust design.

Architecture of Naive PAYG

When failure occurs under ECP-6, we observe that 73 percent of the lines have 0 errors (Table 1). Hence, error correction overhead could be decreased by ~4x, by allocating ECP-6 only for lines that have at least one error. This simplified architecture is called Naive-PAYG, and is shown in Figure 3.

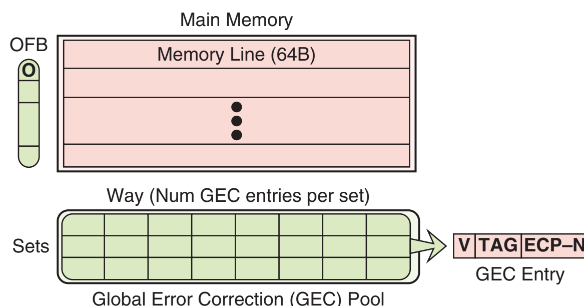


Figure 3: Architecture of Naive-PAYG (newly added structures are shaded)
(Source: Moinuddin K. Qureshi, 2013)

Each line contains an overflow bit (OFB) to indicate if the line has at least one failed bit.[3]

A Global Error Correction (GEC) pool provides error correction entries for such lines. Each GEC entry contains a valid bit, a tag (to identify the owner

[3] A stuck-at-zero OFB can be a single point of failure. Under COV = 20 percent, the probability that a bit will fail at first write is 0.3×10^{-6}. Given 16 million lines in memory, 4.8 lines are expected to have such a failure on average. We avoid this problem by using two-way replication for the OFB bit. We assume that OFB is set to 1, if any of the replicated bits is 1. The probability that both the replicated bits of OFB are stuck-at-zero is negligible (10^{-13}).

line), and one or more ECP entries (ECP-6 in our case). GEC is organized as a set-associative structure. Given that memory designs are highly optimized for a given array size, we want to use a line-size granularity for GEC as well. Therefore one set of GEC is sized such that it fits in 64 bytes, translating to seven GEC entries per set. We found that such a design is noncompetitive compared to even uniform ECP-6 because it suffers from three problems:

- The set associative organization needs a much larger number of entries (~8x) than a fully associative structure to reach the same level of effectiveness.
- Even with the filtering provided by the OFB, 25 percent of the lines can still incur a latency of two accesses (one for main memory and second for GEC), resulting in significant slowdowns.
- Most ECP entries remain unused as six ECP entries are allocated for lines with even one error.

We now describe efficient solutions to each of these problems, leading up to our final design.

Addressing Problem 1: Shortcoming of Set Associative Structure for GEC Pool

In a set-associative organization, each set has only a fixed number of ways, which means that the first set to exceed its allocation causes an uncorrectable failure. So, an important question in determining efficiency of the set-associative structure is to analyze the number of GEC entries occupied before one of the sets overflows. Given that most of the efficient wear-leveling algorithms[4][7] randomize the address space in PCM, we assume that failures occur at random lines in memory, and that any access pattern gets spread over the entire memory (due to remapping from wear leveling). Based on this randomized address space property, we can analyze the effective capacity utilization of a set-associative structure using an analogous buckets-and-balls problem, where a bucket represents one of the sets and a ball represents one of the occupied ways. If there are N buckets, each of which can hold B balls, then the collection can hold a maximum of NB balls. However, if balls are thrown at random, then how many balls can be thrown before one of the buckets overflows? Our Monte Carlo simulations indicate that a 7-way or 8-way GEC pool is only about 12 percent occupied when one of the sets overflows, indicating about 8x inefficiency with a set-associative structure.

Ideally, we want the efficiency of a fully associative structure (where all entries get used) and latency of set-associative structure (single low-latency index). To handle these contradictory requirements, we use a hash-table-with-chaining structure. It consists of two tables: first, the *Set Associative Table (SAT)* and second, the *Global Collision Table (GCT)*. SAT provides a single-index low-latency access to the GEC pool, while GCT provides flexibility in placement. Both SAT and GCT are structurally identical and differ only in the way they

"Ideally, we want the efficiency of a fully associative structure (where all entries get used) and latency of set-associative structure (single low-latency index)."

are indexed. Each GEC set (both in SAT and GCT) also contains a pointer (GCTPTR) that points to a location in the GCT.[4] The proposed GEC structure is shown in Figure 4.

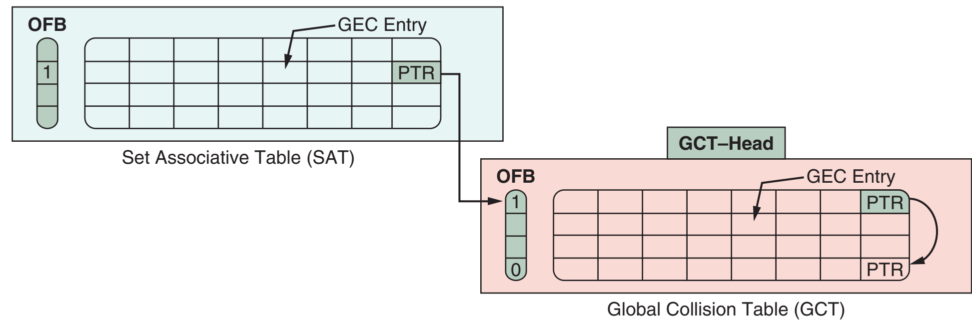


Figure 4: Architecture of scalable GEC pool (Set Associative Table + Global Collision Table) (Source: Moinuddin K. Qureshi, 2013)

Reading GEC Entries

For obtaining a GEC entry, SAT is accessed first in a set that is indexed by some bits of the line address. If there is no tag match in the set, then the GCTPTR of that set identifies the GCT set that must be checked. GCT can be indexed only in this manner. If there is a tag match in the GCT row, then GEC entries can be obtained. If there is no match, the GCTPTR in that set identifies the next GCT set that must be checked. The traversal continues until a GCT entry with matching tag (or a set with OFB = 0) is found.

Allocating GEC Entries

Initially, all GCT sets remain unallocated. These sets get allocated to a set of SAT only on overflow. To aid this allocation, a register called GCT-Head keeps track of the number of GCT entries that have been allocated. When one of the set of SAT or GCT overflows, the GCTPTR of that set is initialized to GCT-Head and the OFB associated with that set is set to 1. The newly allocated set of GCT provides as many GEC entries as the associativity of GCT. The GCTPTR of this newly allocated entry is marked invalid and OFB is set to 0 (to indicate end of traversal).

The GCT-Head is incremented after every GCT allocation. When the value of GCT-Head reaches the number of sets in GCT, it indicates an uncorrectable error.

"…our organization obtains an effective capacity of more than 70 percent of the allocated 1.5N entries, much higher than the 12 percent with a set-associative structure."

We use a GCT that has half as many sets as SAT. Table 2 shows the effective capacity if there are N sets in SAT and $0.5N$ sets in GCT, as the associativity of SAT is varied. For an 8-way SAT, our organization obtains an effective capacity of more than 70 percent of the allocated $1.5N$ entries, much higher than the 12 percent with a set-associative structure.

[4] We use two-way replication for GCTPTR for tolerating errors. We force the GCTPTR with a single stuck-at-bit to point to either location all-zeros or all-ones (both locations are reserved). On mismatch between the two copies of GCTPTR, the entry pointing to the reserved location is ignored. The probability of two bits stuck in GCTPTR is negligible (10^{-12}).

Associativity of SAT	1	2	4	8
Effective Capacity	1.19*N*	1.15*N*	1.11*N*	1.08*N*

Table 2: Effective capacity utilization (of 1.5N entries) with proposed (SAT+GCT) organization
(Source: Moinuddin K. Qureshi, 2013)

In the common case, we want the access to be satisfied by SAT and not the GCT, as GCT incurs higher latency due to multiple memory accesses. Our Monte Carlo simulations show that until about half the entries in SAT get occupied, the probability of single GCT access remains low (less than 1 percent). Thus, the proposed design has a good storage efficiency as well as low latency.

Addressing Problem 2: Local Error Correction for Low-Latency

One of the shortcomings of the naive design is that it accesses the GEC for a line with even one error. We can reduce latency and storage requirements for GEC by allocating a small amount of error correction with each line. For example, we observe that with ECP-1, the likelihood of failure is less than 4 percent even at the end of system lifetime. Therefore if we allocate ECP-1 with each line, we can reduce the GEC access rate as well as demand significantly. We propose to have such Local Error Correction (LEC) with each line. When the number of errors in the line exceeds what can be corrected by LEC, the OFB associated with that line is set and an entry from GEC is allocated. With ECP-1 in LEC, each GEC entry would need to store only ECP-5, which means the GEC can be an 8-way structure in a 64-byte space.

Addressing Problem 3: Fine-Grained On-Demand Allocation for Improved Efficiency

Another source of inefficiency in the naive design is that it allocates a large number of ECP entries for each assignment of a GEC entry. While this amortizes the tag overhead, it results in severe inefficiency, as most of the allocated ECP entries remain unused. The utilization of ECP entries can be increased by reducing the number of ECP entries in each GEC entry. For example, if each GEC entry contained only ECP-1, it would result in significant increase in utilization of ECP entries, even if it would mean relative increase in tag overhead. With ECP-1 in GEC entry, we can fit approximately 24 entries in the space of 64 bytes, therefore the associativity of GEC (SAT as well as GCT) would be 24. As there can be multiple tag hits in a given GEC set, we use the same precedence rule as used in the ECP proposal, that is, GEC entries are allocated from right to left, and younger entries have precedence over older entries. Our design restricts that all GEC entries of a given line must be placed in the same set. If a line needs more GEC entries and that set is full, then all ECP entries of the line are invalidated from the GEC set and relocated into a new set in GCT.

Proposed PAYG: Tying It All Together

PAYG obtains both high storage efficiency and low latency by leveraging the flexible structure for GEC, a hybrid LEC-GEC organization, and fine-grained allocation. Figure 5 shows the overall architecture of our proposed PAYG design.

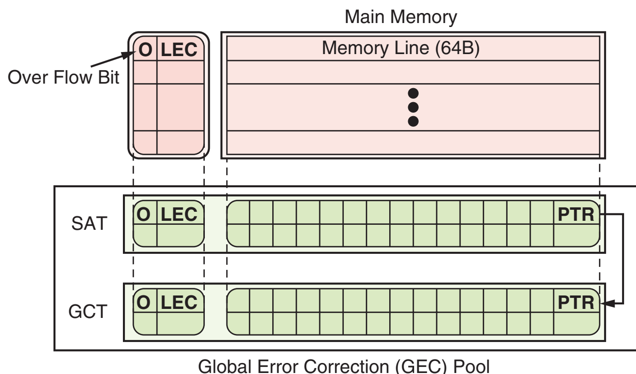


Figure 5: Proposed Architecture of PAYG
(Source: Moinuddin K. Qureshi, 2013)

"The LEC handles the common case of one-or-zero errors in a line for more than 95 percent of the lines. The GEC provides a storage-efficient low-latency on-demand allocation of ECP entries…"

The LEC handles the common case of one-or-zero errors in a line for more than 95 percent of the lines. The GEC provides a storage-efficient low-latency on-demand allocation of ECP entries for lines that have more than one error. Each GEC entry would contain only ECP-1 for high utilization of ECP entries. To reduce the array design overhead, we assume the same memory array for GEC (SAT and GCT) as the main memory, and provision the LEC + OFB for GEC as well, to maintain uniformity (this also allows the GEC size to be changed freely at runtime by the OS). An access to main memory with OFB = 0 is satisfied by single access. When OFB = 1, the GEC is accessed, one or more memory lines are read, matching GEC entries are obtained, ECP information is retrieved, and the line or lines get corrected.

Unlike uniform ECP-6, PAYG does not have to limit the maximum error correction allocated to a line. Thus, a weak line can use as many ECP entries as needed (limited only by the number of GE entries per line). This allows PAYG to outperform even ECP-6. The only real limiter of lifetime with PAYG is the number of GEC entries, as the likelihood of 24 or more errors per line is negligible for our system.

Results and Analysis

Our proposed design has three key components: the scalable structure for the GEC pool, Fine Grained Allocation (FGA), and Local Error Correction (LEC). In this section, we present the key results highlighting the importance of each of these components. We then analyze the storage and latency overheads, and also the impact of different variability scenarios on the effectiveness of our proposal.

Importance of Scalable GEC Pool

The key component of PAYG that provides scalability and efficiency is the architecture of the GEC pool. The first set of results we present are to emphasize the need for such a scalable structure. For this analysis, we assume a version of PAYG that has LEC implemented as ECP-1. The GEC does not have fine-grained allocation, which means each GEC entry contains ECP-5, and each set of GEC (in both SAT and GCT) contains 8 GEC entries. We call this configuration PAYG-NoFGA. Figure 6 compares the normalized lifetime of uniform ECP to that with PAYG-NoFGA. The left sets of bars are for ECP where the level of ECP is varied from 1 to 6. The middle sets of bars are for PAYG-NoFGA without GCT, where the number of sets in SAT is varied from 32K to 1024K. The right sets of bars are for PAYG-NoFGA with 128K sets in SAT and GCT sets vary from 2K to 64K.

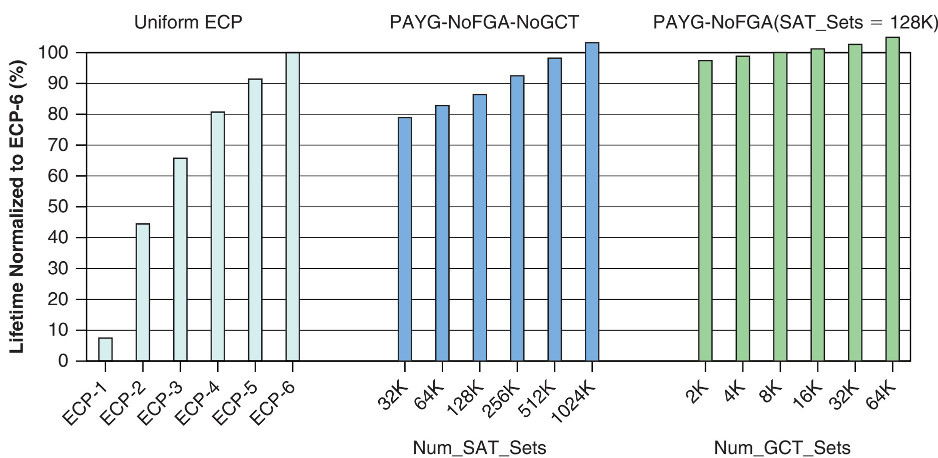


Figure 6: Lifetime of uniform ECP and PAYGNoFGA. Without GCT, PAYG-NoFGA needs 1024 sets (6.25% storage overhead) for lifetime comparable to ECP-6. With GCT, this reduces to (128K + 64K = 192K), 5x lower
(Source: Moinuddin K. Qureshi, 2013)

The first observation is that ECP-6 improves lifetime compared to ECP-1 by more than 10x. Unfortunately, ECP-6 incurs a storage overhead of 12 percent of memory capacity. The second observation is that PAYG-NoFGA needs a large number of sets (1 million) to achieve the lifetime as ECP-6, resulting in significantly high storage overhead (6.25 percent). However, the presence of GCT decreases storage requirement significantly. Combining 128K sets in SAT with 64K sets in GCT can provide a lifetime slightly higher than ECP-6 (this occurs because PAYG does not cap maximum error correction entries to six per line, so a few lines end up using ECP-7). The storage overhead of this combination would be 128K + 64K = 192K sets (1.2 percent overhead), which is 5x lower. Thus, a SAT-GCT based architecture is much more storage efficient than a simple set-associative structure. Unless specified otherwise, we will use 128K-set SAT combined with 64K-set GCT for the rest of the article.

"…SAT-GCT based architecture is much more storage efficient than a simple set-associative structure."

Importance of Fine-Grained Allocation

PAYG-NoFGA allocates five ECP entries with each GEC entry, most of which remain unused. FGA improves the utilization of ECP entries by reducing the number of ECP entries in each GEC entry. Table 3 shows the number of GEC entries that can be packed in one set (64 bytes), when the number of ECP entries in each GEC entry is varied from one to five. The tag size for our GEC structures is 7 bits, and we replicate the valid bit in GEC entry for fault tolerance. We also reserve 32 bits for GCTPTR (16 bits, 2-way replicated), which means only 480 bits per line are available for GEC entries. As the number of ECP entries per GEC entry decreases, the total number of GEC entries per each set increases.

Number of ECP in each GEC entry	1	2	3	4	5
Number of tag bits + valid bits	9	9	9	9	9
Number of bits for ECP	11	21	31	41	51
Size of 1 GEC entry (bits)	20	30	40	50	60
Number of GEC entries per set	24	16	12	9	8
Number of ECP entries per set	24	32	36	36	40

Table 3: Tradeoff between the number of ECP entries per GEC entry vs. ECP entries per set. Note that 24 GEC entries can be packed in one GEC set if each GEC entry contains ECP-1
(Source: Moinuddin K. Qureshi, 2013)

Figure 7 shows the normalized lifetime of PAYG as the number of ECP entries in GEC is varied from six to one. PAYG is implemented with LEC of ECP-1, SAT contains 128K sets, and GCT contains 64K sets. As the number of ECP entries in each GEC entry is reduced, there is a gradual increase in relative lifetime indicating that the effective utilization of ECP entries outweighs the relative increase in tag-store overhead.

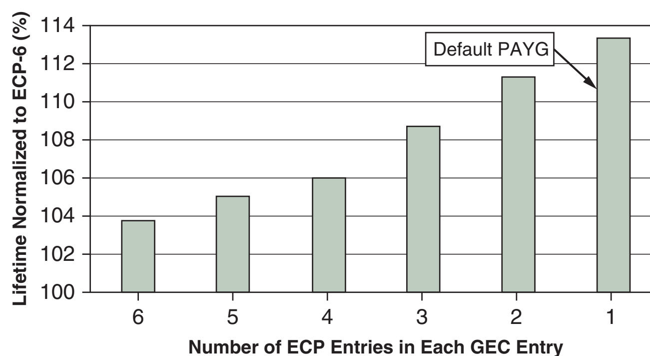


Figure 7: Effect of fine-grained allocation on effectiveness of PAYG. Note that having ECP-1 in GEC provides the highest lifetime and is the default PAYG configuration
(Source: Moinuddin K. Qureshi, 2013)

With only ECP-1 in each GEC entry, PAYG obtains a lifetime 13 percent higher than ECP-6, which is similar to that obtained with uniform ECP-8. Given the

efficiency of such fine-grained allocation, we assume that PAYG is implemented with ECP-1 in each GEC entry. The Default PAYG configuration used in our study is: 128K sets in SAT, 64K sets in GCT, LEC with ECP-1, and FGA with ECP-1 in each GEC entry. This configuration incurs a storage overhead of 3.8 percent of memory capacity and provides 13 percent more lifetime than uniform ECP-6.

"This configuration incurs a storage overhead of 3.8 percent of memory capacity and provides 13 percent more lifetime than uniform ECP-6."

Importance of Local Error Correction

The LEC provides the first line of defense for error correction in PAYG and is designed to handle the common case of zero or one failure per line. Figure 8 shows the normalized lifetime with PAYG as the level of ECP in LEC is varied from zero to six. Note that each ECP in LEC accounts for storage of approximately 2 percent of overall memory capacity, so having higher levels of ECP in each LEC entry incurs significant storage overhead. As expected, the lifetime increases with increasing ECP in LEC. A version of PAYG that has LEC containing ECP-5 has storage similar to uniform ECP-6 and provides a lifetime improvement of 43 percent. Thus, PAYG can not only be used to obtain a given amount of lifetime for reduced storage but can also be used to enhance lifetime at a given storage budget.

For the PAYG configuration without LEC (NoLEC), the given number of GEC entries are insufficient to handle the error rate, hence it obtains a lifetime lower than ECP-6. This can be avoided by simply increasing the number of GEC entries. The right set of bars in Figure 8 shows the lifetime of PAYG without LEC, when the GEC entries are doubled or quadrupled. We observe that simply doubling the entries (storage overhead of 2.4 percent) has lifetime equivalent to ECP-6, and when we double the GEC entries further to overhead of 4.8 percent, this combination can provide a lifetime significantly higher than with uniform ECP. However, the key problem of the PAYG configuration without LEC is the increased access latency. Because the line of defense of LEC is absent, all lines that have even a single error will experience increased latency because of GEC accesses.

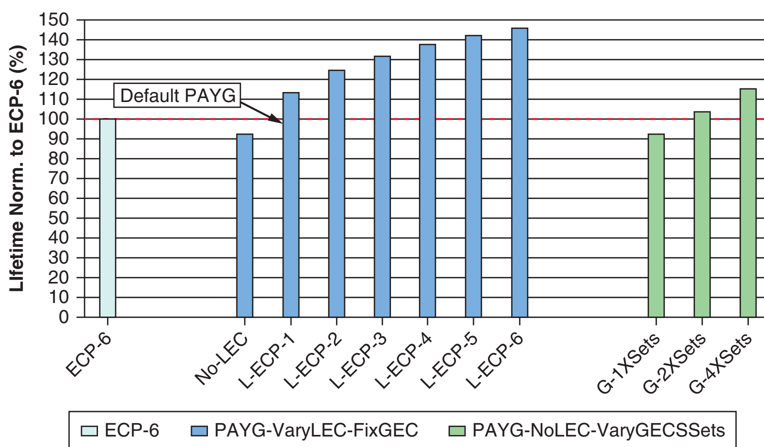


Figure 8: Lifetime impact of LEC: the middle set of bars vary ECP in each LEC entry from zero to six. To get lifetime comparable to ECP-6, we either need at least ECP-1 in LEC, or twice as many sets in GEC (Source: Moinuddin K. Qureshi, 2013)

"The storage overhead of PAYG is 3.13x lower than ECP-6."

Storage Overhead of PAYG

The storage overhead of PAYG consists of two parts: LEC and GEC. The overhead of LEC is incurred on a per-line basis, whereas the overhead of GEC gets amortized over all the lines. Table 4 computes the storage overhead of Default PAYG, given that the bank in our baseline contains $N = 2^{24}$ lines. The LEC incurs 13 bits/line (2-way replicated OFB bits + (1+10) bits for ECP-1). The storage overhead of PAYG is 3.13x lower than ECP-6. On average, PAYG needs 19.5 bits/line vs. 61 bits/line for ECP-6.

	PAYG
LEC (2 OFB + ECP-1)	13 bits/line
SAT (2^{17}) sets	2^{17} lines × 64 B = 8 MB
GCT (2^{16}) sets	2^{16} lines × 64 B = 4 MB
Total overhead of LEC	13 bits ($2^{24} + 2^{17} + 2^{16}$) = 26.9 MB
Total overhead of PAYG	26.9 MB + 8 MB + 4 MB = 38.9 MB
Total overhead of ECP-6	61 bits/line × 2^{24} = 122 MB
Ratio of (ECP-6/PAYG)	122 MB/38.9 MB = 3.13x

Table 4: Storage overhead of PAYG (PAYG obtains 13% more lifetime than ECP-6)
(Source: Moinuddin K. Qureshi, 2013)

Effective Latency with PAYG

Correcting an error with PAYG may require multiple accesses to memory. The main access simply gets broken down into multiple memory accesses (each of which takes deterministic time). The structures SAT and GEC are organized at a granularity of memory line, and we assume that an access to them incurs similar latency as access to main memory. When a GEC access occurs, the SAT is indexed and the memory line obtained is searched for a GECP entry with a matching tag. This incurs one extra memory access. If a match is not found, then the GCT is accessed, which incurs yet another memory access for each GCT access. However, this occurs rarely, given that GEC access happens only when the number of errors in a given line exceeds what can be corrected by the LEC. Figure 9 shows the percentage of demand accesses that require one extra access (satisfied by SAT) and two extra accesses (one for SAT and one for GCT). The probability of one extra access remains 5 percent or less throughout the expected lifetime under ECP-6 (6.5 years under continuous write traffic). Only after that does it increase significantly, reaching 17 percent at the end of lifetime with PAYG. In fact, for the first five years of system lifetime there is on average only 0.4 percent extra access per memory access, which means the performance impact is negligible (less than 0.4 percent) during the useful lifetime. The probability of two extra accesses remains very low throughout the lifetime.

"…for the first five years of system lifetime there is on average only 0.4 percent extra access per memory access…"

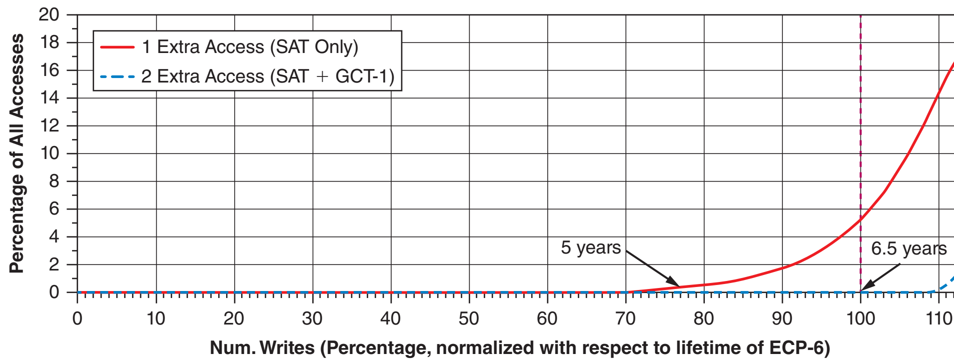


Figure 9: Extra accesses for each demand accesses with PAYG. Note that 100% of ECP lifetime is 6.5 years. PAYG incurs one extra access for less than 0.4% of memory accesses during the first five years of machine lifetime. The latency increases to noticeable range only after 6.5 years. The probability of three or more extra accesses as it remains negligible (< 0.01%) throughout the lifetime
(Source: Moinuddin K. Qureshi, 2013)

Summary

Emerging memory technologies suffer from the problem of limited write endurance. Such systems need high levels of error correction to ensure reasonable lifetime under high variability in device endurance. Uniformly allocating large amounts of error correction entries to all the lines results in most of them remaining unused. We can avoid the storage overhead of such unused entries by allocating the entries in proportion to the number of faults in the line. Based on this key insight, our article makes the following contributions:

- We propose Pay-As-You-Go (PAYG), an efficient hard-error–resilient architecture that allocates error correction entries on-demand, as and when errors occur.
- We propose a storage-efficient, low-latency organization for searching through large number of global error correction (GEC) entries.
- We reduce the latency for accessing error correction entries further by allocating a small amount of Local Error Correction (LEC) per line. Our analysis shows that one bit of LEC per line is sufficient to balance the tradeoff between storage overhead and latency impact.

PAYG can be implemented with any hard-error correction technique and is highly effective compared to line sparing. While we have evaluated the concept of nonuniform fault tolerance in the context of PCM systems, this concept is applicable to other memory technologies as well.

References

[1] Int'l Technology Roadmap for Semiconductors (ITRS). http://www.itrs.net/Links/2008ITRS/Home2008.htm.

[2] E. Ipek et al. "Dynamically replicated memory: building reliable systems from nanoscale resistive memories." ASPLOS-15, 2010.

[3] M. Qureshi et al. "Scalable high performance main memory system using phase-change memory technology." In ISCA-36, 2009.

[4] M. K. Qureshi et al. "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling." In MICRO-42, 2009.

[5] S. Schechter et al. "Use ECP, not ECC, for hard failures in resistive memories." In ISCA-2010.

[6] N. H. Seong et al. "SAFER: Stuck At Fault Error Recovery for Memories." In MICRO-2010.

[7] N. H. Seong et al. "Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping." In ISCA-37, 2010.

Author Biography

Moinuddin Qureshi is an Associate Professor in the School of Electrical and Computer Engineering at Georgia Institute of Technology. His research interests include computer architecture, scalable memory systems, fault-tolerant computing, and analytical modeling of computer systems. Prior to joining Georgia Tech, he was a research staff member at IBM T.J. Watson Research Center from 2007 to 2011. He was awarded the IBM outstanding technical achievement award for his studies on emerging memory technologies for server processors. He received his PhD (2007) and MS (2003), both in Electrical Engineering, from the University of Texas at Austin, and Bachelor of Electronics Engineering (2000) degree from University of Mumbai. He is a recipient of the NetApp Faculty Fellowship (2012) and Intel Early Career Faculty Award (2012). He can be reached at moin@ece.gatech.edu.

AN INTEGRATED SIMULATION INFRASTRUCTURE FOR THE ENTIRE MEMORY HIERARCHY: CACHE, DRAM, NONVOLATILE MEMORY, AND DISK

Contributors

Jim Stevens
University of Maryland

Paul Tschirhart
University of Maryland

Mu-Tien Chang
University of Maryland

Ishwar Bhati
University of Maryland

Peter Enns
University of Maryland

James Greensky
Intel Labs

Zeshan Chisti
Intel Labs

Shih-Lien Lu
Intel Labs

Bruce Jacob
University of Maryland

As computer systems evolve towards exascale and attempt to meet new application requirements such as big data, conventional memory technologies and architectures are no longer adequate in terms of bandwidth, power, capacity, or resilience. In order to understand these problems and analyze potential solutions, an accurate simulation environment that captures all of the complex interactions of the modern computer system is essential. In this article, we present an integrated simulation infrastructure for the entire memory hierarchy, including the processor cache, the DRAM main memory system, and nonvolatile memory, whether it is integrated as hybrid main memory or as a solid state drive. The memory simulations we present are integrated into a full system simulation, which enables studying the memory hierarchy with a faithful representation of a modern x86 multicore processor. The simulated hardware is capable of running unmodified operating systems and user software, which generates authentic memory access patterns for memory hierarchy studies. To demonstrate the capabilities of our infrastructure we include a series of experimental examples that utilize the cache, DRAM main memory, and nonvolatile memory modules.

Introduction

The rise of multicore systems has shifted the primary bottleneck of system performance from the processor to the memory hierarchy, accelerating the gap that had already existed between processor and memory performance (the memory wall). Previously, the memory wall problem was the result of the increasing frequencies of CPUs relative to the latency of the memory system, which meant that CPUs were losing more processing time waiting on memory accesses. However, as processor frequency improvements stalled and with the introduction of multicore systems, a more urgent problem was created since the current memory system cannot scale at the same rate as the number of cores. Therefore, in modern systems there is actually much less bandwidth and capacity per core than there was a few years ago. This trend can be seen in Figure 1. This problem, combined with the existing operating frequency problem, has led to the memory hierarchy becoming the dominant source of slowdown in the system. To address the increased need for capacity, systems are now relying more on solid state drives and other high performance storage systems, exacerbating the latency problem of the memory system due to the increased frequency of references to the slower storage system. Finally, since multicore systems are running threads in different address spaces with different access patterns, there is less locality of reference for the cache hierarchy to exploit. This implies that overcoming the multicore memory wall problem requires examining the entire memory hierarchy from the cache system down to the storage system.

"…overcoming the multicore memory wall problem requires examining the entire memory hierarchy…"

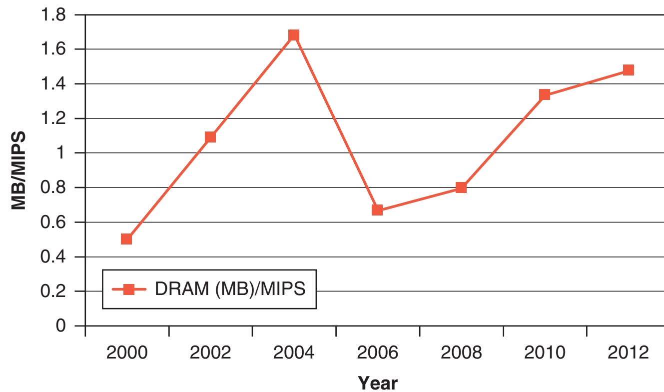


Figure 1: DRAM capacity (MB)/processor speed(MIPS) per core for a typical system
(Source: University of Maryland, 2013)

In addition to the strain on memory system capacity and bandwidth that has been introduced by multicore chips, memory system capacity is also limited by scaling problems at the device level. For DRAM, as the memory cells shrink, the charge that can be stored on the capacitor becomes very small and the pass transistor leakage increases, which reduces the retention time of the cell and requires more complex peripheral circuitry to detect the smaller charge. For flash memory, as the dielectric of the floating gate shrinks, the amount of damage during program-erase cycles that can be tolerated decreases and the cells wear out faster.[1] Additionally, since control circuitry has analog components that are difficult to scale down, as the DRAM and flash cell size decreases, the control circuitry takes up a larger percentage of the chip area relative to the memory array. Architects have attempted to address device scaling problems by adding more devices with technologies like FB-DIMM and Buffer on Board, as well as technologies in currently development like the Hybrid Memory Cube.[2] However, these solutions require additional hardware to be designed and added to the memory system, making them currently prohibitively expensive for most applications. New memory technologies have also been suggested that might eventually provide a solution to the capacity problem but these technologies are not yet competitive with existing technologies in terms of cost or capacity.[17] Meanwhile, software is not helping to alleviate the situation, because application working sets continue to increase in size. In recent years, big data applications such as bioinformatics and graph analytics have only accelerated the increasing demand for faster and more scalable storage systems. This has also contributed to the rapid adoption of solid state drives. However, much of the storage system's software and hardware infrastructure was constructed around assumptions of millisecond access latencies and, as a result, fails to efficiently utilize the new high performance storage solutions being implemented. In order to meet the new challenges posed by big data applications, the storage system needs to be reworked from the OS file system down to the hardware interfaces. Finally, as the

"…big data applications such as bioinformatics and graph analytics have only accelerated the increasing demand for faster and more scalable storage systems."

community pushes towards exascale computers, the power and resilience limitations of the current memory system are becoming more pronounced. [3] If an exascale-sized main memory system were constructed using today's technology, then just that component alone would consume the entire system power budget. Furthermore, given the current probability of failure in memory system components, as the number of components approach the numbers needed for exascale, the probability of a failure somewhere in the system approaches 1. This means that if an exascale computer were built with today's memory technology, not only would it use too much energy, it would also be breaking constantly. Therefore, to enable the push to exascale it is imperative that new, more energy-efficient and resilient memory technologies and architectures be developed.

"…the feedback between the various components of the system is vital to understanding performance."

In order to overcome these problems, new architectures and software need to be developed and evaluated. Since the new solutions will involve multiple aspects of the system, the feedback between the various components of the system is vital to understanding performance. For example, many researchers are studying how to integrate nonvolatile memory into the system as a first class citizen, which involves both the hardware and the software. Trace-based simulation has been used in the past to study these kinds of architecture problems. Unfortunately, trace-based simulation does not capture the feedback loops between software and hardware. One way to produce these feedback loops is to build a real-world prototype. However, due to the engineering effort required, real-world prototypes are impractical and costly for studying large design spaces. Full system simulation models those complex interactions and can provide valuable insights into the dynamic behavior of a variety of system designs. Previously, no full system simulator existed that could study all levels of the memory and storage hierarchy. In this article, we describe our simulation infrastructure that addresses this need by providing a full system simulator capable of modeling the entire processor and memory hierarchy, including the storage system.

Simulator Description

Our memory hierarchy simulation infrastructure is an extension of the MARSSx86 full system simulation environment[4] developed at SUNY Binghamton. We utilize MARSS to simulate the microprocessor and other non-memory hierarchy components of the system. The memory infrastructure builds on top of the prior MARSS memory hierarchy and incorporates detailed simulations of every level of the hierarchy including the cache, the main memory system, and the storage system. The cache simulator is an extended version of the existing cache simulation in MARSS that allows for heterogeneous technologies at different levels of the cache hierarchy. For traditional DRAM-based main memory systems, our simulation environment uses DRAMSim2, which is a detailed, cycle-accurate DRAM memory system simulator developed by our lab[13]. For nontraditional hybrid nonvolatile/DRAM memory systems our simulation environment uses two modules, HybridSim and NVDIMM, which simulate the memory controller and

nonvolatile DIMMs that would be used by such a system. The hybrid memory components can also be reconfigured to simulate solid state drives. Figure 2 shows the overall structure of our simulation environment, including its constituent modules and how they communicate with one another.

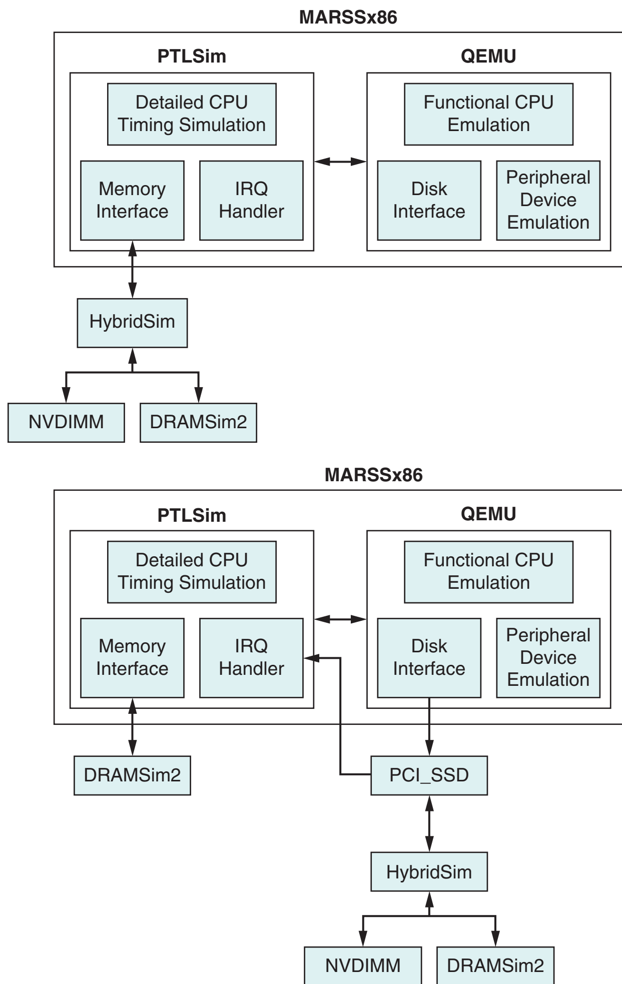


Figure 2: Block diagram of simulation environment for hybrid memory (top) and SSD (bottom)
(Source: University of Maryland, 2013)

MARSS

MARSS is designed to simulate a modern x86 computer system. MARSS utilizes PTLSim to simulate the internal details of the processor. PTLSim is capable of simulating a multicore processor with the full details of the pipeline, micro-op front end, reorder buffers, trace cache, and branch predictor. In addition, PTLSim also simulates a full cache hierarchy and can implement several cache coherency protocols. For the hardware that is not explicitly simulated, such as disks or the network card, MARSS uses the QEMU emulation environment. MARSS is able to boot full, unmodified operating systems, such as any Linux distribution, and then run unmodified benchmarks. We selected MARSS as the basis for our memory hierarchy simulation infrastructure because of its ability to simulate

"We selected MARSS as the basis for our memory hierarchy simulation infrastructure because of its ability to simulate both the user programs and the operating system functionality…"

both the user programs and the operating system functionality, while most other simulation environments are only capable of simulating user-level instructions. Therefore, in addition to being the most realistic simulation environment possible, MARSS can be used to study the behavior of the operating system, which we view as vital to solving the problems of future memory and storage systems.

Cache Simulation

While PTLSim already provides an SRAM-based cache simulation, studying other technologies is vital because of the power, bandwidth, and capacity problems that arise in the design of the memory hierarchy for future systems. Memory technologies such as SRAM, STT-RAM, and eDRAM have been considered for implementing on-die LLCs. Though they all have low read latency and high write endurance, they can be very different for other performance characteristics. For instance, SRAM is low density and has high leakage current, STT-RAM has high write latency and write energy consumption, and eDRAM requires refresh. Additionally, due to the very different inherent characteristics of each of the memory technologies, researchers have proposed various power and performance optimization techniques. Therefore, in order to make useful comparisons between SRAM, STT-RAM, and eDRAM LLCs, we expand MARSS with the following:

1. We integrate a refresh controller into MARSS to support eDRAM LLCs.
2. In addition to the parameterized cache access time, we expand MARSS with parameterized cache cycle time, tag access latency, and refresh period. Separating cycle time and tag access latency allows the user to evaluate pipelined caches and sequentially accessed caches (such as when data array access is skipped on a tag mismatch). We also modify MARSS to support asymmetric cache read and write latencies. This property is required to evaluate STT-RAM caches realistically.
3. We integrate dead line predictors to enable low power modes for SRAM and eDRAM caches.

"These changes allow our environment to investigate future cache designs incorporating new technologies and techniques."

These changes allow our environment to investigate future cache designs incorporating new technologies and techniques.

DRAM Main Memory Simulation

Since the DRAM-based main memory system has a large number of configuration and timing parameters, such as the command and data queues, address mappings, refresh timings, low power modes, activate and pre-charge periods, and so on, choice of one or another scheme could have drastically different power or performance implications.[14] Therefore, DRAMSim2, a cycle-accurate JEDEC DDRx memory system simulator, was developed.[12][13] It models the memory controller, memory channels, DRAM ranks, and banks. The DRAMSim2 timing behavior has been compared and validated against Verilog-based device models published by DRAM vendors.

Recently, JEDEC published the next generation DDR4 standard.[15] DDR4 devices could operate at double the speed of previous generation DDR3 chips,

and moreover DDR4 will have additional features enabling low power and high memory capacity. DDR4 devices will have banks separated into multiple bank-groups to facilitate higher bandwidths and greater bank-level parallelism. However, since banks within a bank-group share some peripheral circuitry, requests to banks of the same bank-group takes longer time than banks on different bank-groups. We modified the DRAMSim2 memory controller to incorporate these DDR4 specific changes.

Power dissipated due to DRAM represents a substantial portion of the total system power budget, as the main memory capacity and bandwidth increases to satisfy requirements of the current and future data-intensive applications. Therefore, to study the tradeoffs involved with switching to various DRAM low power modes, such as active, power-down, self-refresh, and deep power-down , requires accurate switching time as well as the current drawn during each mode. Furthermore, refresh command scheduling could also potentially affect the switching to low power modes. We have augmented DRAMSim2 with detailed low power modes and a range of refresh policies, allowing users to study the performance and power tradeoff when using different low power modes and refresh methods.

"We have augmented DRAMSim2 with detailed low power modes and a range of refresh policies,"

Nonvolatile Memory Simulation

Recently many designs have been proposed that utilize nonvolatile device based DIMMs to address the capacity issues of the main memory system. For DIMMs that are not made using DRAM parts, we use NVDIMM, which is capable of simulating DIMMs made from a wide variety of technologies. This is possible because most nonvolatile technologies share many common features and differ in only a few parameters. For instance, both flash and Phase Change Memory (PCM) feature asymmetric reads and writes. To allow for these differences, NVDIMM has a wide variety of options that can be used to shape the behavior of the system. Some technology-specific options include access latencies, device interface widths, address mapping policies, and wear leveling policies. For example, in flash a dynamic mapping scheme is used so that dirty pages can be set aside to be erased during idle cycles by a garbage collection process, enabling faster modifications of existing data. This scheme was chosen because the erase time for flash is prohibitively long even for basic storage applications. Early architectures for PCM, on the other hand, have been designed with a simpler static mapping scheme that does not require a garbage collection process because its erase is considerably faster than flash's.

In addition, other options have been included in NVDIMM to enable investigations into the effects of organization, scheduling, and timing. A good example of such a study is to determine how many devices of a given type can be included on a DIMM before the host interface channel (such as DDR3 or SATA) is saturated. By enabling both device and architecture level investigations, NVDIMM allows our memory hierarchy simulation infrastructure to study different methods for integrating nonvolatile memory into a computer system.

"…NVDIMM allows our memory hierarchy simulation infrastructure to study different methods for integrating nonvolatile memory into a computer system."

Nonvolatile Memory Integration

There are two primary ways to integrate nonvolatile memory into a computer system below the cache level, as illustrated in Figure 3. The first method is the traditional storage route, which uses the same software and hardware abstractions and protocols as hard disk drives. The second method is to tie the nonvolatile memory directly into the memory controller. Our memory hierarchy simulation infrastructure is designed in such a way that you can utilize a common set of modules to simulate both integration methods, which enables the ability to make fair comparisons between the two.

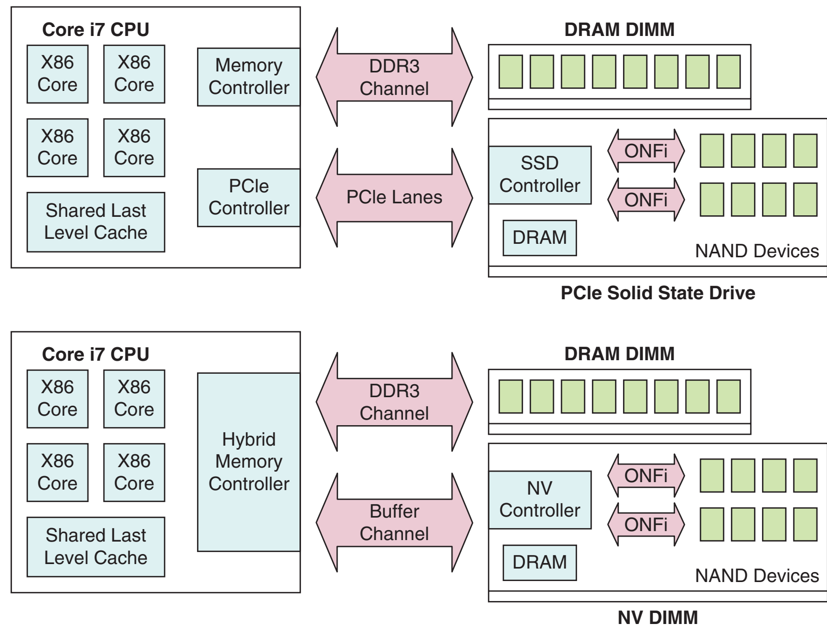


Figure 3: System design for SSD (top) and hybrid memory (bottom)
(Source: University of Maryland, 2013)

In both disk-like and memory-like integration methods, since the nonvolatile memory typically has long latencies, a faster memory such as DRAM or SRAM is utilized as a buffer or cache. We provide the HybridSim module to simulate this aspect of the system. HybridSim uses NVDIMM as its backing store and DRAMSim2 as its cache. HybridSim's features enable the study of a variety of cache replacement policies, prefetching policies, and hardware/software co-design (for example, having the memory controller and operating system work together to manage nonvolatile memory).

"HybridSim's features enable the study of a variety of cache replacement policies, prefetching policies, and hardware/software co-design…"

When HybridSim is simulating a memory-like integration method for nonvolatile memory, also known as a hybrid main memory, it interacts with the memory controller of the base MARSS system to capture addresses and bypass its simpler memory model. HybridSim then performs its caching functions and sends requests to DRAMSim2 or NVDIMM to implement requests. When the requests complete, HybridSim sends callbacks to the MARSS memory controller to indicate that a request is done and allow the processor to make progress at the appropriate clock cycle.

When HybridSim is simulating a disk-like integration method, it receives disk requests from MARSS and then later raises an I/O interrupt to indicate a request is complete. This works exactly like a modern solid state drive. We also provide an additional module called PCI_SSD to simulate the host interface for a modern SATA or PCIe SSD and to allow the user to configure various options including the number of lanes, half or full duplexing, an optional two-level interface (such as Intel Direct Media Interface to SATA), frequency, and protocol overhead. Our SSD simulation also ties in with our DRAMSim2 main memory simulation to perform direct memory access operations to DRAM before or after a disk request occurs. This process of disk simulation is also compatible with simulators for conventional hard disk drives like DiskSim[5] and HDD simulation could be achieved by simply modifying the PCI_SSD module.

"Our SSD simulation also ties in with our DRAMSim2 main memory simulation to perform direct memory access operations…"

Simulation Variability and Warm-Up

Full system simulation introduces some additional sources of complexity and nondeterminism that can lead to inaccurate results if they are not dealt with properly. In particular, just as in a real system, the OS introduces nondeterminism into the simulation as a result of timing variation (for example, interrupt arrival time) from run to run. This problem can be reduced by utilizing checkpoints of the system state, which MARSS enables using the QEMU snapshot mechanism. Another source of complexity is how to properly warm up the caches and other state (such as NVDIMM's address mapping) for novel memory hierarchy architectures. We provide a generic mechanism for warm-up utilizing state files that can be saved during a warm-up period or generated by scripts and then restored at the beginning of the region of interest. An example of this warm-up process can be seen in Figure 4.

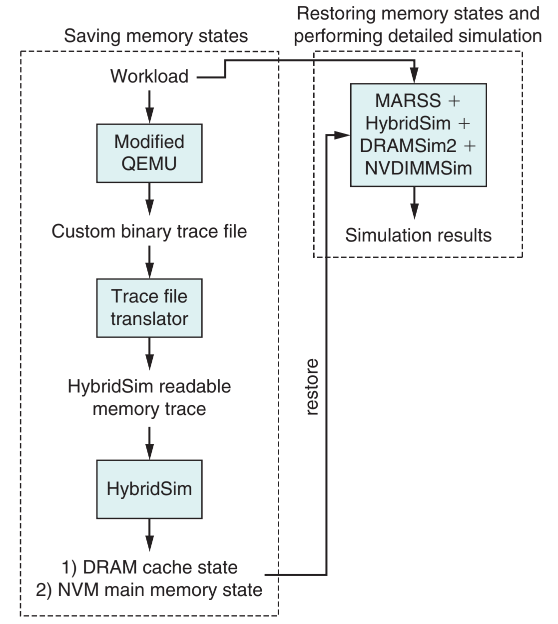


Figure 4: An example of a complete warm-up
(Source: University of Maryland, 2013)

Baseline Configuration

The baseline configuration for the following experiments is a quad-core, out-of-order system, with cache organization similar to the Intel® Core™ i7. The cache experiments below use this processor with a modified LLC to incorporate new memory technologies. The cache experiments also utilized the baseline DRAM main memory configuration. These baseline configurations are shown in Table 1.

Processor	4-core, issue width = 4, 2 GHz
L1I (private)	128 KB, 8-way, 64-B block size
L1D (private)	128 KB, 8-way, 64-B block size
L2 (private)	2 MB, 8-way, 64-B block size
L3 (shared) (if present)	8 MB, 16-way, 64-B block size
DRAM (if used as cache)	512 MB, 64-way, 4-KB page size
DRAM (if used as main memory)	1 GB, DDR3-1333
Nonvolatile main memory	8 GB, 4-KB page size, PCIe 3.0 16 Lane equivalent bandwidth

Table 1: Baseline Configuration
(Source: University of Maryland, 2013)

The DRAM examples also utilize the baseline processor and cache shown in Table 1.

For the hybrid and SSD experiments, an 8-GB NVM is considered, with a 512-MB DRAM cache in front of it. The nonvolatile DIMM organization has 1 channel, 64 dies per channel, 2 planes per die, 4,096 blocks per plane, 64 pages per block, and each page is 4 KB. All transfers between the NVM and the DRAM occur at the page granularity. The timing parameters for the nonvolatile memory are based on MLC flash numbers.[6] The DRAM cache, also in the form of a DIMM, is organized as 1 channel, 1 rank per channel, 8 banks per rank, 8,192 rows per bank, and 1,024 columns per row. All transfers between the DRAM and the L2 cache occur at the L2 cache line granularity (64 B). DRAM timing parameters are based on a Micron datasheet.[7] All devices are 8 bits wide.

For these experiments we use the GUPS benchmark and a random access micro-benchmark called mmap developed by our lab as well as selected benchmarks from the NAS benchmark suite, the SPEC benchmark suite, and the PARSEC benchmark suite.[8][9][10][11] These benchmarks were selected because they have a large working set size and are memory intensive.

Experiments

The following experiments demonstrate examples of the wide variety of studies that can be performed using the various modules of our environment. For the processor cache, we present energy and execution time data for last-level caches constructed using different memory technologies for a several benchmarks. To demonstrate the capabilities of the DRAM system portion of the simulator, we have included power and instructions-per-cycle data for similar sets of several benchmarks. Finally, we exhibit the features of the nonvolatile memory portions of our environment with data showing the effects of additional bandwidth, prefetching, working set size, and memory system traffic volume on system performance. Table 1 contains the baseline configuration details that are common to all of the experiments.

Caches

"…we compare the LLC energy consumption and system performance when using SRAM, STT-RAM, and eDRAM."

As a case study, we compare the LLC energy consumption and system performance when using SRAM, STT-RAM, and eDRAM. The LLC is a 32-nm, 32-MB, 16-way write-back cache that is partitioned into 16 banks and uses 64-byte blocks. It is also pipelined and sequentially accessed.

Figure 5 illustrates the normalized energy breakdown of LLCs based on SRAM, STT-RAM, and eDRAM. We include the results for "regular" implementations (without power-optimization) and "low power" implementations. For instance, "regular" SRAM uses high performance transistors to implement the entire cache without power gating; "regular" STT-RAM uses storage-class STT-RAM technology, which has a long retention time but requires high write energy; and "regular" eDRAM uses the conventional periodic refresh method. On

the other hand, low power SRAM uses dead line prediction[18], power gating, and low leakage CMOS for the memory cells[19] to reduce leakage power; low power STT-RAM uses device optimization techniques to reduce write energy by sacrificing data-retention time[17]; and low power eDRAM uses dead line prediction to reduce the number of refresh operations. The impact of different memory technologies and implementations on system performance is shown in Figure 6.

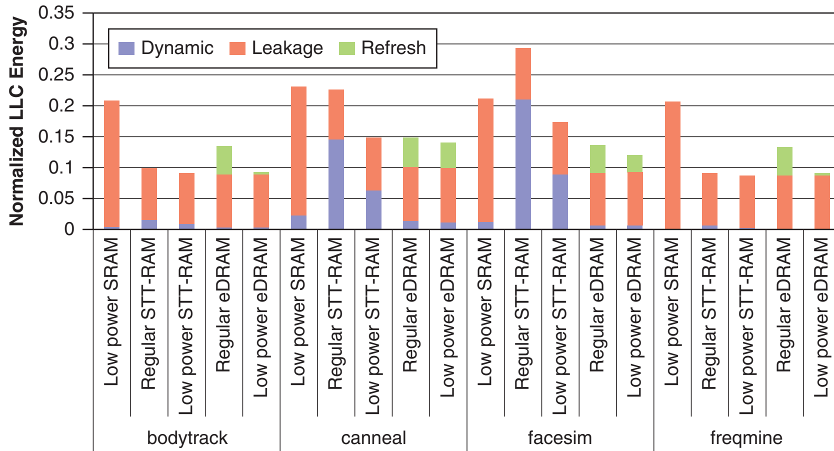


Figure 5: Normalized LLC energy breakdown with respect to various memory technologies. The results are normalized to regular SRAM (not shown). Note that regular SRAM dissipates 5x more power on average (Source: University of Maryland, 2013)

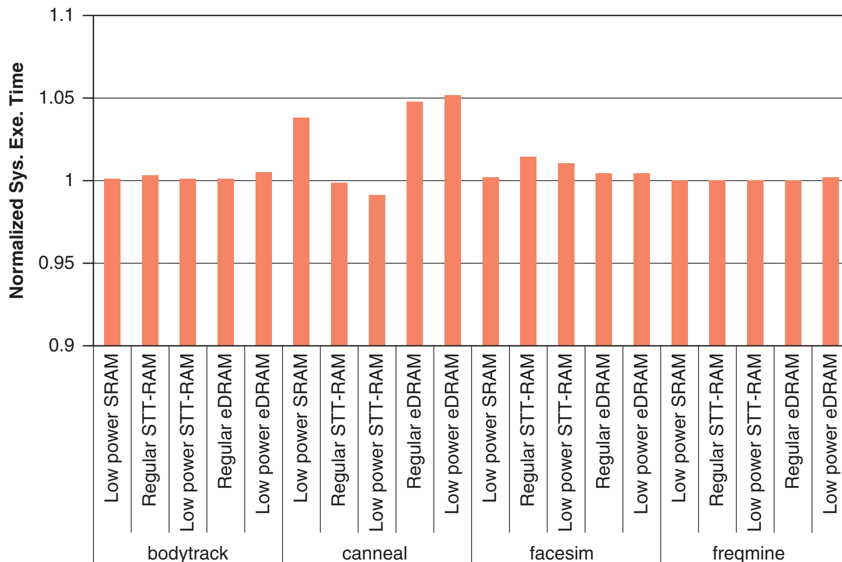


Figure 6: Normalized system execution time with respect to various memory technologies. The results are normalized to regular SRAM (not shown) (Source: University of Maryland, 2013)

DRAM

As an interesting case study of a DRAM-based main memory system, we show the impact of refresh when device size is increased from small 1-Gb to future big 32-Gb chips. We simulated few SPEC2006 benchmarks in region of interest (RoI) for 1 billion instructions, assuming both with and without refresh enabled. Figure 7 presents the energy contribution separated for each type of operation, that is: read and write, activate and pre-charge, background and refresh operations. The Y-axis representing energy is normalized to the corresponding 1-Gb device values for each benchmark. The background and refresh energy portion increases for higher density devices, because of the greater number of peripheral circuitry and cells to be refreshed as device size increases. Since with DRAM density, the number rows also increases, this leads to more frequent refresh commands to be scheduled, and therefore leads to a degradation of the memory performance and latency. Figure 8 shows the percentage degradation of system performance (IPC) and the average latency increase due to refresh operations as the size of DRAM devices vary.

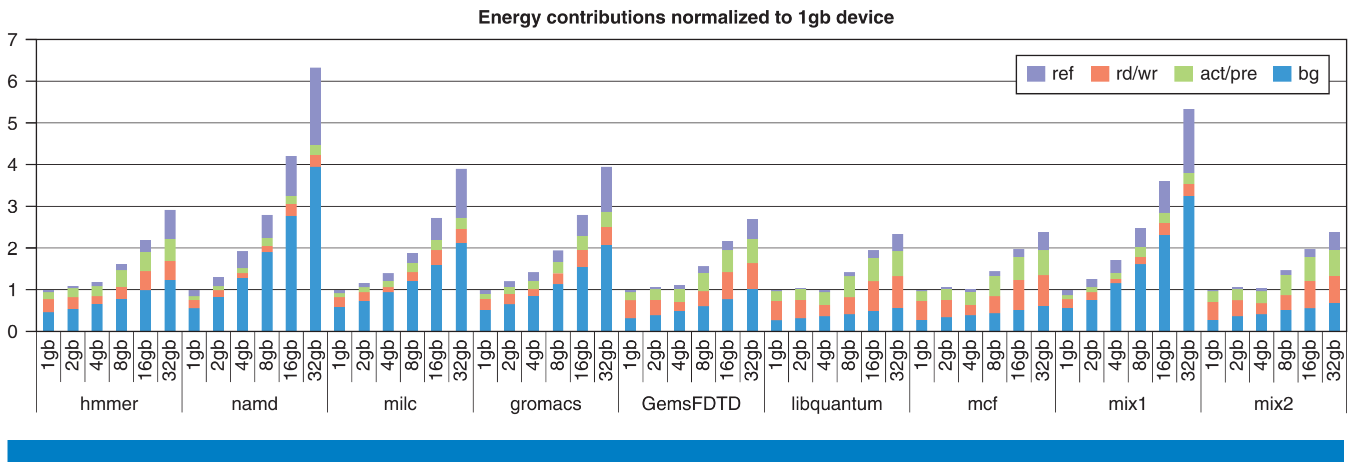


Figure 7: Energy contributions separated for each operation type normalized to the 1-Gb DRAM device size. Refresh and background energy consumption increases when DRAM density gets higher
(Source: University of Maryland, 2013)

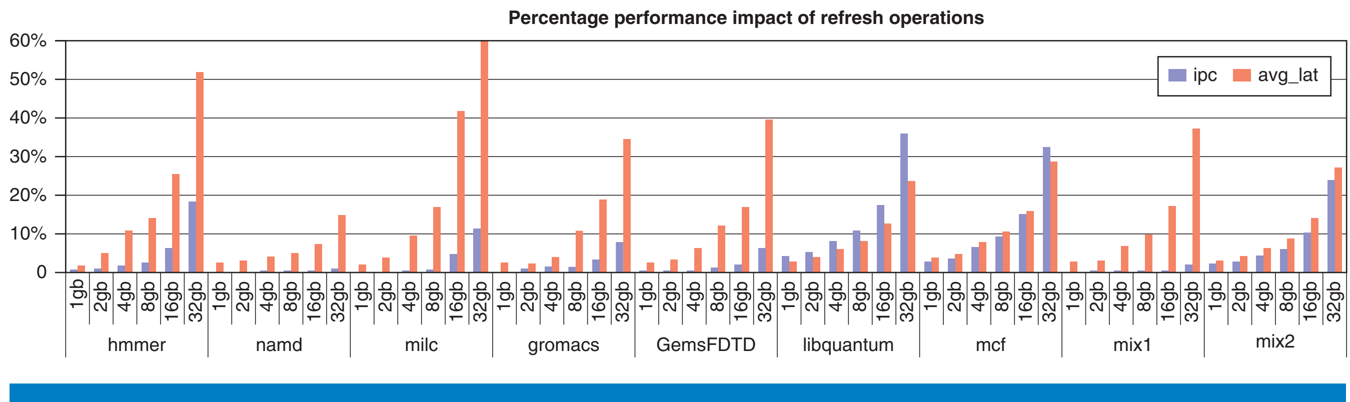


Figure 8: Percentage of refresh penalty measured using Instructions Per Cycle (IPC) of the entire system and average latency of the memory system. For higher density devices, the performance penalty increases sharply
(Source: University of Maryland, 2013)

Nonvolatile Memory

An important type of study for future memory systems is to understand how the system reacts to changing the working set size and volume of accesses. This is especially important in hybrid main memory systems because nonvolatile memory latencies can be significantly slower than traditional SRAM and DRAM. The Giga-Updates Per Second (GUPS) implementation from Sandia National Laboratories is an ideal benchmark to study such access patterns since it takes the working set size and number of accesses as parameters, unlike many other benchmarks that assume a constant pattern for memory accesses. GUPS creates a large table and then performs a series of updates on pseudorandom locations within that table. In this experiment we chose table sizes of 256 MB, 512 MB and 1 GB. The DRAM cache in our test system was 512 MB. Our choice of table sizes allows us to see the effect on system performance when the table fits in the DRAM cache easily, when the table is approximately the same size as the DRAM cache to cause some swapping between the DRAM cache and the nonvolatile backing store, and when the table is two times the size of the DRAM cache to cause a significant number of DRAM cache misses. We also vary the number of updates from 1000 to 5000 in increments of 1000 to show the effect of different volumes of memory traffic on system performance. Finally, we included data for systems that incorporate the nonvolatile memory as both a hybrid memory and as a traditional SSD. From the results in Figure 9, we can see that for the SSD configuration as the table grows larger than the 512 MB DRAM and more accesses must go to the slower flash swap space, system performance suffers as would be expected. However, for the hybrid memory version, performance is not dependent on the table size. This is because Linux sees the 8 GB backing store as the main memory address space and allocates the entire table inside this space. Initially, this table is not present in the DRAM cache because it has been accessed yet. When the table size is twice the size of the DRAM, the performance of the Hybrid implementation becomes much better than the SSD implementation. This is because the SSD has more overhead for its accesses to the swap space than the Hybrid has for its accesses to the flash.

"…the SSD has more overhead for its accesses to the swap space than the Hybrid has for its accesses to the flash."

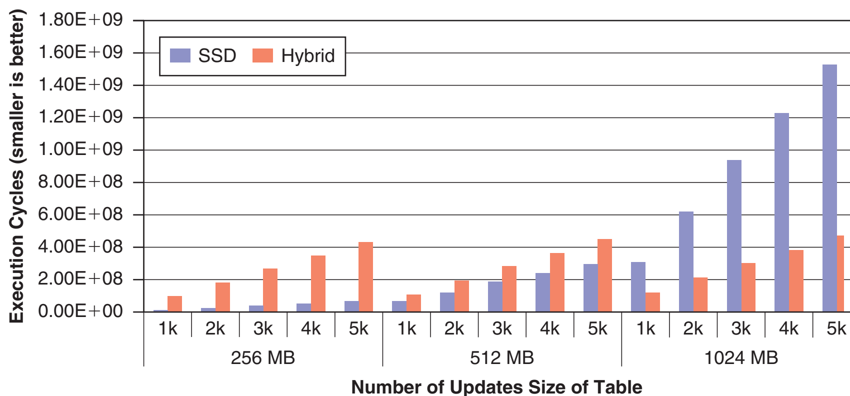


Figure 9: Execution time of GUPS when table size and number of updates are varied (smaller is better)
(Source: University of Maryland, 2013)

"Optimizing the performance of the nonvolatile backing store is another important area of study for future memory systems."

Optimizing the performance of the nonvolatile backing store is another important area of study for future memory systems. One area of potential performance gain is the interface of the nonvolatile devices used to create the backing store. To show the effect of improving the bandwidth provided by these interfaces, we utilize an in-house micro-benchmark called MMAP. MMAP works by first defining a large memory mapped file that is opened with the mmap() system call in Linux and then it accesses this file randomly. This benchmark is well suited to bandwidth studies because it is single threaded and therefore provides a clear picture of the effect of a minor change without much noise from other system threads. Additionally, since MMAP is designed to force misses to the DRAM cache as often as possible, which causes only one 64-byte access within each 4-KB page, it maximally stresses the host interface and device channels in the backing store. This is a worst-case scenario for the memory system because it generates a large volume of random accesses that are not fully utilized by the cache. This is the reason for the low observed IPC. For this experiment, we vary the clock rate of the interface of a device (the amount of time it takes to transmit 8 bits of data) from 0.05 ns to 10 ns. In addition, we also utilize a basic sequential prefetching algorithm to generate more accesses and place greater pressure on the devices. We vary the number of additional pages that are prefetched by our algorithm from 4 to 8 to 16. As was the case in the previous example, we also include data for both a hybrid-style integration of the nonvolatile memory and an SSD-style integration. In Figure 10, we can see that both faster device interfaces and larger prefetching windows help to improve the system performance. We do not use the prefetching in HybridSim for the SSD version of the experiment because prefetching is performed by the operating system for disk accesses. It is also important to note that there is less nondeterminism in these results than in the previous example because this example is single threaded, which eliminates nondeterminism introduced by the OS scheduler when it has to schedule multiple threads. There is still some minor nondeterminism in this experiment's results, but that is what one would expect from a real system.

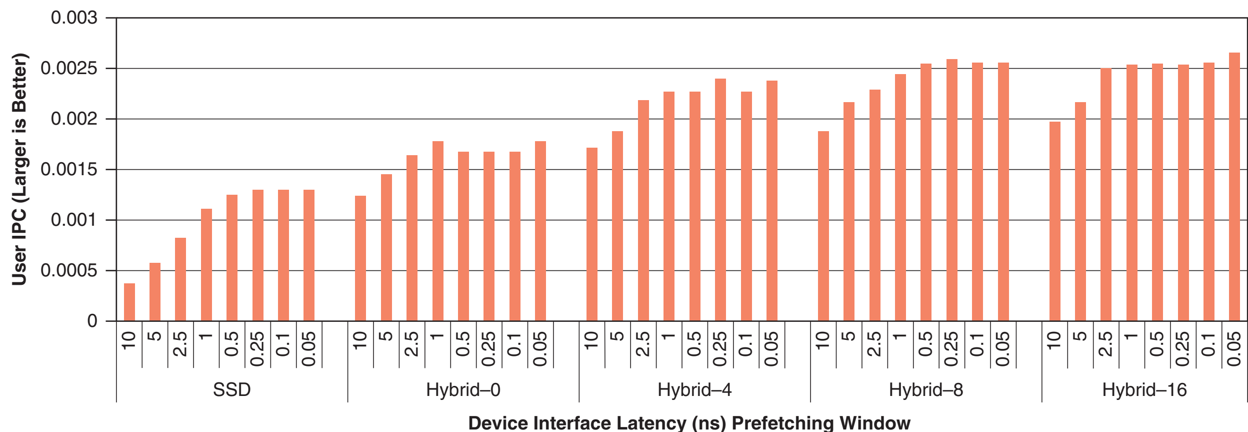


Figure 10: Performance of MMAP with varying bandwidth and prefetching window size (Source: University of Maryland, 2013)

Conclusion

In this work, we have introduced a complete memory hierarchy simulation environment that is capable of accurately simulating the processor cache, the DRAM main memory system, and nonvolatile memory, whether it is implemented as a hybrid memory or as an SSD. We have shown the utility of this infrastructure for solving future memory hierarchy design problems by presenting example experiments that demonstrated multiple last-level cache cell technologies, DRAM refresh schemes, and nonvolatile memory integration methods.

References

[1] Laura M. Grupp, John D. Davis, and Steven Swanson. 2012. "The bleak future of NAND flash memory." In Proceedings of the 10th USENIX conference on File and Storage Technologies (FAST'12). USENIX Association, Berkeley, CA, USA, 2–2.

[2] E. Cooper-Balis, P. Rosenfeld, and B. Jacob, "Buffer On Board memory systems," in Proceedings of the 39th Annual International Symposium on Computer Architecture, ser. ISCA '12, 2012.

[3] US Department of Energy Office of Science. "The Opportunities and Challenges of Exascale Computing." Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee. Fall 2010.

[4] A. Patel et al., "MARSSx86: A Full System Simulator for x86 CPUs," in Design Automation Conference 2011 (DAC'11), 2011.

[5] J. Bucy et. al. "The DiskSim Simulation Environment Version 4.0 Reference Manual." Carnegie Mellon University Parallel Data Laboratory Technical Report CMU-PDL-08–101. May 2008.

[6] Micron Technology. "128Gb SLC Flash Datasheet." 2012. [Online]. Available: http://www.micron.com/parts/nand-flash/mass-storage/mt29f128g08akcabh2-10

[7] Micron Technology. "4Gb DDR3 SDRAM Datasheet." 2009. [Online]. Available: http://www.micron.com/parts/dram/ddr3-sdram/mt41j256m16re-125

[8] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. "The PARSEC benchmark suite: characterization and architectural implications." In Proceedings of the 17th international conference on Parallel architectures and compilation techniques (PACT '08). ACM, New York, NY, USA, 72–81.

[9] NASA Advanced Supercomputing Division. "NAS Parallel Benchmarks." 2012. [Online]. Available: http://www.nas.nasa.gov/publications/npb.html

[10] Standard Performance Evaluation Corporation. "SPEC2006 CPU Benchmarks." 2012. [Online]. Available: http://www.spec.org/cpu2006/

[11] Sandia National Labs. "RandomAccess GUPS (Giga Updates Per Second)." 2012. [Online]. Available: http://www.sandia.gov/~sjplimp/algorithms.html

[12] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Katie Baynes, Aamer Jaleel, and Bruce Jacob. "DRAMsim: A memory-system simulator." SIGARCH Computer Architecture News, Vol. 33, No. 4, pp. 100–107. September 2005.

[13] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," Computer Architecture Letters, Vol. 10, No. 1, pp. 16–19, Jan.-June 2011.

[14] S. Srinivasan, L. Zhao, B. Ganesh, B. Jacob, M. Espig, and R. Iyer. "CMP memory modeling: How much does accuracy matter?" Proc. Fifth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), pp. 24–33. Austin TX, June 2009.

[15] JEDEC, "DDR4 SDRAM Standard," 2012. [Online]. Available: http://www.jedec.org/sites/default/files/docs/JESD79-4.pdf

[16] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in Proceedings of the 36th Annual International Symposium on Computer Architecture, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 24–33.

[17] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan. "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on, Vol., no., pp. 50–61, 12–16 Feb. 2011.

[18] Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. "Cache decay: exploiting generational behavior to reduce cache leakage power." In Proceedings of the 28th annual international symposium on Computer architecture (ISCA '01). ACM, New York, NY, USA, 240–251.

[19] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," Proceedings of the IEEE, Vol. 91, No. 2, pp. 305–327, Feb. 2003.

Author Biographies

Jim Stevens received a BS degree in computer engineering from the University of Kansas in 2006, an MS degree in computer science from the University of Arkansas, Fayetteville, in 2009, and is currently pursuing a PhD in computer science at the University of Maryland, College Park. His research interests include memory controller design and adapting operating systems for nonvolatile memories.

Paul Tschirhart received his BS degree in computer engineering from the University of Virginia in 2007. He is currently pursuing a PhD in computer and electrical engineering at the University of Maryland, College Park. His research interests include memory controller design, memory system architecture, and SSD design.

Mu-Tien Chang received the BS and the MS in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2006 and 2008, respectively. He is currently pursuing a PhD in electrical and computer engineering at the University of Maryland, College Park. His research interests include memory circuit and processor cache design.

Ishwar Bhati received B.Tech. in electronics and communication engineering from Indian Institute of Technology, Guwahati, India, in 2005. He worked in the VLSI/ASIC industry as design and verification engineer for five years. He is currently pursuing a PhD in electrical and computer engineering at the University of Maryland, College Park. His research interests include energy-efficient memory systems and high performance computing.

Peter Enns is currently pursuing a PhD in linguistics at the University of Maryland with a concentration in computational linguistics and natural language processing. He received a BS in computer engineering from the University of Maryland in 2011 with honors (summa cum laude). While he was an undergrad, Peter studied nonvolatile memory systems with Dr. Bruce Jacob in Maryland's Memory Systems Research Lab.

James Greensky is currently a software engineer in the Memory Architecture Lab (MAL) within Intel Labs. James received his BS and MS degrees in computer science and is currently pursuing a PhD in the area of computer architecture from the University of Minnesota.

Zeshan Chishti received the BSc (Hons) degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 2001, and a PhD in computer engineering from Purdue University in 2007. He is a Research Scientist at Intel Labs, Hillsboro, Oregon. His research interests include microarchitecture, energy-efficient memory systems, and cache hierarchies for chip multiprocessors.

Shih-Lien Lu received his BS in EECS from UC Berkeley, and MS and PhD both in CSE from UCLA. He is a principal researcher and leads the memory architecture team at Intel Labs. From 1984 to 1991 he was on the MOSIS project at USC/ISI, which provides research and education community VLSI fabrication services. He was on the faculty of the ECE Department at the Oregon State University from 1991 to 2001. His research interests include computer microarchitecture, memory circuits, and VLSI systems design.

Bruce Jacob received the AB degree in mathematics from Harvard University in 1988 and the MS and PhD degrees in CSE from the University of Michigan in Ann Arbor in 1995 and 1997, respectively. He also worked for two successful startup companies: Boston Technology and Priority Call Management; at Priority Call Management he was the initial system architect and chief engineer. He is a professor of electrical and computer engineering at the University of Maryland in College Park, and he is currently visiting at the University of Siena, Italy, where he is working on memory issues for many-core systems. He is a recipient of a US National Science Foundation CAREER award for his work on DRAM, and he is the lead author of an absurdly large tome on the topic of memory systems. His research interests include memory systems, operating systems, and designing electric guitars.