Opportunistic Memory Systems in Presence of Hardware Variability

Mark William Gottscho

Committee:
Puneet Gupta (chair)
Lara Dolecek
Mani Srivastava
Glenn Reinman

Ph.D. Defense
UCLA Electrical Engineering
Friday, May 12, 2017
Memory is Essential

3.3M photos

2.3M Google searches per second

400 hours of video uploaded per minute
Hardware Variability in Memory

Hardware variability is particularly problematic for memories:

1. Smallest and densest device/circuit features
2. Large fraction of the chip area budget
3. Must permit instability in order to be rewritable

Memories are particularly susceptible to:

1. Manufacturing defects
2. Parametric variations
3. The operating environment

Memory wall often limits:

1. Energy efficiency
2. System resiliency

[Hennessy & Patterson ‘12]
Better-Than-Worst-Case Design

Introduction

Underdesigned and Opportunistic Computing machines

[Gupta et al. TCAD’13]
Opportunistic Memory Systems exploit & cope with hardware variations within and across individual chips for improved energy efficiency and resiliency.
Overview of My Dissertation

**Part 1: Opportunistically Exploiting Memory Variability**

1. ViPZonE: Saving Energy in DRAM Main Memory  
   *with Power Variation-Aware Memory Management*

2. DPCS: Saving Energy in SRAM Caches  
   *with Dynamic Power/Capacity Scaling*

3. X-Mem: Case Studies on Memory Performance Variability  
   *with the new Extensible Memory Characterization Tool*

**Part 2: Opportunistically Coping with Memory Errors**

4. Performability: Exploring the Impact of Corrected Memory Errors  
   *by quantifying and analytically modeling their performance effects*

5. SDECC: Recovering from Detected-but-Uncorrectable Memory Errors  
   *with Software-Defined Error-Correcting Codes*

6. ViFFTo: Improving Reliability of Embedded Scratchpad Memories  
   *with Virtualization-Free Fault Tolerance*
Agenda

• Introduction

• Part 1: Exploiting Variability
  – ViPZonE
  – DPCS
  – X-Mem

• Part 2: Coping with Errors
  – Performability
  – SDECC
  – ViFFT0

• Conclusion and Directions for Future Work
ViPZonE: Saving Energy in DRAM Main Memory using Power Variation-Aware Memory Management

Collaborators:
- Dr. Luis A. D. Bathen (UC Irvine)
- Prof. Nikil Dutt (UC Irvine)
- Prof. Alex Nicolau (UC Irvine)
- Prof. Puneet Gupta (UCLA)

Publications:
- Gottscho et al., ESL’12
- Bathen et al., CODES+ISSS’12
- Dutt et al., ASP-DAC’13
- Gottscho et al., TC’15
- Wanner et al., it’15
Part 1: Opportunistically Exploiting Memory Variability

Summary of ViPZonE

[Gottscho ESL’12, Bathen CODES+ISSS’12, Dutt ASP-DAC’13, Gottscho TC’15, Wanner it’15]

ViPZonE-enabled apps tell OS how to allocate virtual pages w/ special variant of malloc() in modified standard C library

ViPZonE-enabled glibc tells OS how to allocate virtual pages w/ special variant of mmap() syscall

Kernel’s physical page allocator attempts to map allocated virtual page in particular memory device

Legacy app does not exploit power variability!

ViPZonE app consolidates pages onto low power zones!
Part 1: Opportunistically Exploiting Memory Variability

Summary of ViPZonE

- Up to 27.8% energy savings on Intel Sandy Bridge/DDR3 testbed desktop
- No more than 4.8% performance degradation

Use ViPZonE when high memory-level parallelism or bandwidth is not needed

Physical zoning inherently trades off benefits of striping for resource consolidation and exploitation of device variations

Opportunistically save energy in today’s systems with no hardware changes

Through smart management of physical memory variation signatures
DPCS: Saving Energy in SRAM Caches with Dynamic Power/Capacity Scaling

Collaborators:
Dr. Abbas Banaiyan-Mofrad (UC Irvine)
Prof. Nikil Dutt (UC Irvine)
Prof. Alex Nicolau (UC Irvine)
Prof. Puneet Gupta (UCLA)

Publications:
Gottscho et al., DAC’14
Dutt et al., DAC’14
Gottscho et al., TACO’15
Wanner et al., it’15

Chapter 3
Part 1: Opportunistically Exploiting Memory Variability

Summary of DPCS

[Gottscho DAC’14, Dutt DAC’14, Gottscho TACO’15, Wanner it’15]

- Pre-characterize SRAM faults using BIST
- Encode min non-faulty VDD on per-block basis
  - Store in modified tag array with 2 extra bits per block

- High performance mode
  - Full VDD & cache capacity
- Low power mode
  - Reduced VDD, disabled faulty blocks

![Diagram showing SRAM Data Array and Col. Decode]
Part 1: Opportunistically Exploiting Memory Variability

Summary of DPCS

- Up to 79% total cache energy savings
- Up to 26% total system energy savings
- Average 2.24% performance overhead
- 6% total cache area overhead

Power vs. capacity tuning

Useful energy efficiency knob, complements DVFS

Fault Inclusion Property

Exploit it for efficient storage of fault maps

Opportunistic approach to energy-efficient caches

Leverage variability without harming reliability or performance
# Part 1: Opportunistically Exploiting Memory Variability

## X-Mem: A New Tool for Case Studies on Memory Performance Variability

### Collaborators:
- Dr. Sriram Govindan (Microsoft)
- Dr. Bikash Sharma (Microsoft)
- Dr. Mohammed Shoaib (Microsoft Research)
- Prof. Puneet Gupta (UCLA)

### Publications:
- Gottscho et al., ISPASS’16
Summary of X-Mem

Part 1: Opportunistically Exploiting Memory Variability

Summary of X-Mem

New flexible tool for characterizing memory systems

Surpasses capabilities of all prior tools

Key Features
(A) Diverse access patterns
(B) Cross-platform
(C) Flexible metrics
(D) Extensible

Three case studies
Explored efficacy of opportunistic variation-aware DRAM latency tuning

Agenda

• Introduction
• Part 1: Exploiting Variability
  – ViPZonE
  – DPCS
  – X-Mem
• Part 2: Coping with Errors
  – Performability
  – SDECC
  – ViFFTo
• Conclusion and Directions for Future Work
Performability: The Impact of Corrected Memory Errors on Performance

Collaborators:
- Dr. Mohammed Shoaib (Microsoft Research)
- Dr. Sriram Govindan (Microsoft)
- Dr. Bikash Sharma (Microsoft)
- Dr. Di Wang (Microsoft Research)
- Prof. Puneet Gupta (UCLA)

Publications:
- Gottscho et al., CAL’16
How Fault Tolerance Impacts Cloud Application Performance

Error Logging

Checkpointing

Page Retirement

Mirroring

Sparing

ECC Encode/Decode

$H \epsilon' = 0$
X-Mem extended: controlled injections of correctable memory errors in production-spec cloud server

Corrected memory errors can have severe impact on application performance!
Part 2: Opportunistically Coping with Memory Errors

Batch applications on multiprocessors with broadcast error handling

$S$, total processors

$\lambda$  

$M$

$\langle t_0, \gamma(\tilde{y}_0) \rangle$

$t_0 < t_1 < \ldots < t_K$

Multi-threaded Batch Task (Work-Sharing)

SMI or CMCI  

Error Queue

SMI: Broadcast  

CMCI: Single-issue  

$\langle t_i, \tau_i(\tilde{x}_i) \rangle$
Part 2: Opportunistically Coping with Memory Errors

Summary of Performability

Recommendations
- **Integrate** performability models and empirical data into high-level TCO models
- **Reduce** the overhead of hardware error reporting via architecture/firmware/OS optimizations
- **Prevent** faults proactively using page retirement and variation-aware memory management
Part 2: Opportunistically Coping with Memory Errors

Agenda

• Introduction

• Part 1: Exploiting Variability
  – ViPZonE
  – DPCS
  – X-Mem

• Part 2: Coping with Errors
  – Performability
  – SDECC
  – ViFFTo

• Conclusion and Directions for Future Work
Part 2: Opportunistically Coping with Memory Errors

SDECC: Recovering from Detected-but-Uncorrectable Memory Errors using Software-Defined Error-Correcting Codes

Collaborators:
- Clayton Schoeny (UCLA)
- Prof. Lara Dolecek (UCLA)
- Prof. Puneet Gupta (UCLA)

Publications:
- Gottscho et al., SELSE’16
- Gottscho et al., DSN-W’16
- Gottscho et al., 2017 manuscript submitted and under peer review

Chapter 6
Part 2: Opportunistically Coping with Memory Errors

**SDECC Concept**

[Godtsho SELSE’16, Gottscho DSN-W’16, Gottscho ‘17]
Candidate Codewords

Example using SECDED
(concept applies generally)

- Hamming sphere
- 2-bit DUE with 4 equidistant candidate codewords
- 1-bit CE
- Each dotted edge is a single-bit flip between two $n$-bit strings
Part 2: Opportunistically Coping with Memory Errors

Analysis of Existing ECC Codes

[Gottscho ‘17]

<table>
<thead>
<tr>
<th>Class of Code</th>
<th>Type of Code</th>
<th>n</th>
<th>k</th>
<th>t</th>
<th>q</th>
<th># ways DUE</th>
<th>Avg. # C.C.</th>
<th>Baseline Prob. Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit SECDED</td>
<td>[Hsiao IBM Jour. ‘70]</td>
<td>39</td>
<td>32</td>
<td>1</td>
<td>2</td>
<td>741</td>
<td>12.04</td>
<td>8.50%</td>
</tr>
<tr>
<td>32-bit SECDED</td>
<td>[Davydov Trans. IT ‘91]</td>
<td>39</td>
<td>32</td>
<td>1</td>
<td>2</td>
<td>741</td>
<td>9.67</td>
<td>11.70%</td>
</tr>
<tr>
<td>64-bit SECDED</td>
<td>[Hsiao IBM Jour. ‘70]</td>
<td>72</td>
<td>64</td>
<td>1</td>
<td>2</td>
<td>2556</td>
<td>20.73</td>
<td>4.97%</td>
</tr>
<tr>
<td>64-bit SECDED</td>
<td>[Davydov Trans. IT ‘91]</td>
<td>72</td>
<td>64</td>
<td>1</td>
<td>2</td>
<td>2556</td>
<td>16.62</td>
<td>6.85%</td>
</tr>
<tr>
<td>32-bit DECTED</td>
<td>-</td>
<td>39</td>
<td>32</td>
<td>2</td>
<td>2</td>
<td>14190</td>
<td>4.12</td>
<td>28.20%</td>
</tr>
<tr>
<td>64-bit DECTED</td>
<td>-</td>
<td>79</td>
<td>64</td>
<td>2</td>
<td>2</td>
<td>79079</td>
<td>5.40</td>
<td>20.53%</td>
</tr>
<tr>
<td>128-bit SSCDSD (ChipKill-Correct)</td>
<td>[Kaneda Trans. Comp ‘82]</td>
<td>36</td>
<td>32</td>
<td>1</td>
<td>16</td>
<td>141750</td>
<td>3.38</td>
<td>39.88%</td>
</tr>
</tbody>
</table>
Computing Candidate Codewords

Part 2: Opportunistically Coping with Memory Errors

Algorithm
For each symbol-wise error position
  For each symbol-wise error value
    Perturb received string using current position/value
    ECC-decode the perturbed string
    If decoder produces a codeword
      Add codeword to list of candidates

Example using SECDED

<table>
<thead>
<tr>
<th>Original Codeword</th>
<th>Received String (2-bit DUE)</th>
<th>Candidate Codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000 0000 0000</td>
<td>0000 1000 1000 0000</td>
<td>1000 1000 1000 0001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0110 1000 1000 0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0001 1000 1000 0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000 0000 0000 0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Decoded bit flip

Perturbed bit flip

3-bit DUE, not a candidate!
Exploiting Data Side Information in Memory

Data types

- `uint32_t`, `double`, pointers, packed arrays, classes...

Object states

- Assertions, invalid pointers...

Data correlation

- Previously used for compression

[Yang MICRO'00, Alameldeen '04, Pekhimenko PACT'12]
Part 2: Opportunistically Coping with Memory Errors

Data Entropy-based Recovery Policy

[Mark Gottscho ‘17]

- Use entropy to determine most-correlated candidate codeword
  - High entropy detected → force a panic
  - Low entropy detected → heuristically recover

\( x_i \): Value of byte \( i \) in 64B cache line

**Entropy:**

\[
H(X) = - \sum_{i=1}^{64} P(x_i) \log_2 P(x_i)
\]
Architectural Support: SDECC for Main Memory

- Existing DRAM systems already have most of the required support for SDECC
  - ECC decoder
  - Error status registers
  - Error-reporting interrupts
- We only need to expose the corrupted cacheline to system software!
  - Extend functionality of existing error status registers and interrupt

No performance/energy overhead in common cases with no DUE!
Overall SDECC Approach

ECC decode

No errors or correctable errors (CEs)?

Yes

Success

No (DUE)

Probabilistic Success

Read Penalty Box

Compute candidate codewords (CCs)

Calculate cacheline sample entropy for each CC

Min. entropy above given threshold?

No

Force panic

Yes

Write back recovered CC to Penalty Box

Heuristically recover most likely CC
Results: DUE Recovery Breakdown

[Gottscho ‘17]

- Trace-based fault injection campaign
- 20 SPEC CPU2006 benchmarks
- RISC-V instruction set architecture
Part 2: Opportunistically Coping with Memory Errors

Results for Approximation-Tolerant Applications

[72,64,4]_2 Hsiao SECDED

<table>
<thead>
<tr>
<th></th>
<th>blackscholes</th>
<th>fft</th>
<th>inversek2j</th>
<th>jmeint</th>
<th>jpeg</th>
<th>sobel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>83.8</td>
<td>49.5</td>
<td>82.9</td>
<td>90.4</td>
<td>92.4</td>
<td>90.8</td>
</tr>
<tr>
<td>Forced Panic</td>
<td>9.6</td>
<td>38.6</td>
<td>11.4</td>
<td>4.9</td>
<td>4.6</td>
<td>6.0</td>
</tr>
<tr>
<td>MCE Total</td>
<td>6.4</td>
<td>11.8</td>
<td>5.5</td>
<td>4.5</td>
<td>2.8</td>
<td>3.1</td>
</tr>
</tbody>
</table>

Breakdown of MCE Total

<table>
<thead>
<tr>
<th></th>
<th>Benign</th>
<th>Crash</th>
<th>Hang</th>
<th>Tol. NSDC</th>
<th>Intol. NSDC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4.8</td>
<td>0.5</td>
<td>0.4</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>6.5</td>
<td>0.9</td>
<td>0.0</td>
<td>3.3</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>4.2</td>
<td>0.2</td>
<td>0.0</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>3.2</td>
<td>0.8</td>
<td>0.0</td>
<td>0.1</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>0.6</td>
<td>0.0</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

Original image (jpeg benchmark) 
Worst-case corrupted image (out of 1000) 
Pixel Delta
Pruning Candidates with Lightweight Hashes

What if we could prune the list of candidate codewords to improve chance of recovery?

Solution: lightweight hashes

- Compute small (4, 8, or 16-bit) universal hash of original cacheline, store in memory
- If-and-only-if DUE occurs:
  - Read out original hash
  - Compare it against computed candidate hashes
Lightweight Hash Implementation: ChipKill

[Gottscho '17]
Part 2: Opportunistically Coping with Memory Errors

**Overall SDECC Approach With Hashes**

[Gottscho ‘17]

1. **ECC decode**
2. **No errors or correctable errors (CEs)?**
   - Yes: **Success**
   - No (DUE): **Min. entropy above given threshold?**
     - No: **Force panic**
     - Yes: **Write back recovered CC to Penalty Box**

**Software**

- Read Penalty Box
- Compute candidate codewords (CCs)
- Calculate cacheline sample entropy for each CC

**Hardware**

- Filter using cacheline hash
- Hash outcome?
  - One or more CC match
    - **Heuristically recover most likely CC**
    - Min. entropy above given threshold?
      - No: **Force panic**
      - Yes: **Success**
  - No CC match

---

Mark Gottscho <mgottscho@ucla.edu> -- UCLA Electrical Engineering

Ph.D. Final Defense
May 12, 2017
Lightweight hashes can improve SDECC recovery rates by orders of magnitude

**Rates of Successful DUE Recovery**

<table>
<thead>
<tr>
<th></th>
<th>baseline</th>
<th>none</th>
<th>4-bit</th>
<th>8-bit</th>
<th>16-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECDED</td>
<td>5%</td>
<td>71.6%</td>
<td>87.8%</td>
<td>98.56%</td>
<td>N/A</td>
</tr>
<tr>
<td>ChipKill</td>
<td>39.9%</td>
<td>85.7%</td>
<td>98.05%</td>
<td>99.940%</td>
<td>99.9999%</td>
</tr>
</tbody>
</table>

SDECC Failure Rate With Hashes (Forced Panic or Induced MCE)

- [72,64,4]_2 SECDED (Hsiao)
- [36,32,4]_16 SSCDSD (ChipKill-correct)

- Close to DEC “almost for free”
- Close to Double-ChipKill “almost for free”
Summary of SDECC

[Gottscho SELSE’16, Gottscho DSN-W’16, Gottscho ‘17]

• Reliability Benefits
  – Approximation-tolerant applications
    • Recover up to 92.4% of DUEs with [72,64,4]_2 SECDED
    • As low as 0.1% intolerable NSDC rate
  – Approximation-intolerant applications with 16-bit Lightweight Hash
    • Recover up to 99.9999% of DUEs with [36,32,4]_16 SSCDSD ChipKill-correct
    • MCE rate less than 0.2 ppm of DUEs

• Applications to several domains
  – Supercomputing: help reduce checkpoint frequency, saving time/energy
  – Approximation-tolerant IoT devices: support error correction at low cost
  – Real-time embedded systems: avoid missing deadlines when errors occur
Agenda

- Introduction
- Part 1: Exploiting Variability
  - ViPZonE
  - DPCS
  - X-Mem
- Part 2: Coping with Errors
  - Performability
  - SDECC
  - ViFFT0
- Conclusion and Directions for Future Work
ViFFTo: Virtualization-Free Fault Tolerance for Embedded Scratchpad Memories at Low Cost

Collaborators:
- Irina Alam (UCLA)
- Clayton Schoeny (UCLA)
- Prof. Lara Dolecek (UCLA)
- Prof. Puneet Gupta (UCLA)

Publications:
- Gottscho et al., 2017 manuscript submitted and under peer review
Part 2: Opportunistically Coping with Memory Errors

ViFFTo Approach

[Gottscho ‘17]
FaultLink: Guarding Against Hard Faults at Link-Time

[Gottscho ‘17]

Part 2: Opportunistically Coping with Memory Errors

Test chip data SPM

750 mV  700 mV  650 mV
Results: Hard Faults

[Gottscho ‘17]
SDELC: Guarding Against Soft Faults at Run-Time

Software-Defined Error-Localizing Codes (SDELCs)
- Based on novel Ultra-Lightweight Error-Localizing Codes (UL-ELCs)
  - Between parity & Hamming code
  - Detect & localize 1-bit errors to specific chunk

Software-Defined Recovery using Embedded C Library
- Application-driven data & instruction recovery policies
Results: Soft Faults

[Gottscho ‘17]

70% of single-bit errors can be recovered at less than half the cost of a standard Hamming code!
Summary of ViFFTo

[Gottscho ‘17]

• ViFFTo opportunistically copes with memory errors in low-cost IoT devices
  – FaultLink can reduce VDD by up to 440 mV
  – SDELc can recover 70-90% of single-bit soft faults

• Minimal or no hardware overheads required
  – Improve yield (cost), energy, and reliability of IoT devices
  – Safest for approximation-tolerant applications
Agenda

• Introduction
• Part 1: Exploiting Variability
  – ViPZonE
  – DPCS
  – X-Mem
• Part 2: Coping with Errors
  – Performability
  – SDECC
  – ViFFTo
• Conclusion and Directions for Future Work
Summary of Dissertation

• Addressing **energy efficiency** and **resiliency** of memories is essential

• Opportunistic memory systems can help solve this problem!
  
  • Part 1: ViPZonE, DPCS, X-Mem
    – Exploited hardware variability
  
  • Part 2: Performability, SDECC, ViFFTco
    – Coped with memory errors

Open-source code available at [https://github.com/nanocad-lab](https://github.com/nanocad-lab)
Data available at [http://nanocad.ee.ucla.edu/Main/DownloadForm](http://nanocad.ee.ucla.edu/Main/DownloadForm)
Directions for Future Work

• Short-term
  – Software-Defined ECC with fault models
  – Application-specific fault tolerance for hardware accelerators
  – Adapting techniques to emerging non-volatile memory devices

• Long-term
  – Joint abstractions for heterogeneity and variability
  – Checkerboard Architecture

• Vision
  – Demand for data + hardware specialization → Opportunistic Memory Systems
Acknowledgments

• Committee
  – Prof. Puneet Gupta (advisor)
  – Prof. Lara Dolecek
  – Prof. Mani Srivastava
  – Prof. Glenn Reinman

• UC Irvine
  – Prof. Nikil Dutt
  – Prof. Alexandru Nicolau
  – Dr. Luis A. D. Bathen
  – Dr. Abbas BanaiyanMofrad

• Microsoft
  – Dr. Mohammed Shoaib
  – Dr. Sriram Govindan
  – Dr. Bikash Sharma
  – Dr. Di Wang
  – Mike Andrewartha
  – Mark Santaniello
  – Dr. Jie Liu
  – Dr. Badriddine Khessib
  – Dr. Kushagra Vaid

• Qualcomm
  – Dr. Greg Wright

• UCLA doctoral students
  – Clayton Schoeny
  – Dr. Fred Sala
  – Salma Elmalaki
  – Dr. Lucas Wanner

• UCLA NanoCAD Lab
  – Irina Alam
  – Dr. Shaodi Wang
  – Dr. Liangzhen Lai
  – Yasmine Badr
  – Saptadeep Pal
  – Dr. Abde Ali Kagalwalla
  – Weiche Wang
  – Dr. Rani Ghaida
  – Dr. John Lee

• UCLA department staff
  – Deeona Columbia
  – Sandra Bryant
  – Mandy Smith
  – Ryo Arreola

• Funding
  – Qualcomm Innovation Fellowship
  – UCLA Dissertation Year Fellowship
  – US National Science Foundation Variability Expedition Grant No. CCF-1029783
  – UCLA Electrical Engineering Department PhD Fellowship


Questions?
BONUS SLIDES
1. Introduction
## Main Memory System

![Diagram of the main memory system](image)

### Breakdown of Main Memory

<table>
<thead>
<tr>
<th>Location</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>64b DATA</td>
<td>+8b ECC parity</td>
</tr>
</tbody>
</table>

**Memory Controller(s)**

- **Main Memory**
  - **L1D$**
  - **L1I$**
  - **L2$**
  - **L2$**

**Memory Controller(s)**

- **DIMM (Socketed)**
  - **8b**
  - **8b**
  - **8b**
  - **8b**

**Bank**

- (Independent, Lockstep within rank)

**Array**

- **Rows**
- **Columns**

**DRAM Chip**

- **Bank**

**Row Buffer**

- **64b DATA (+8b ECC parity)**

**Channels**

- **CMD, ADDR, CLK, RANK_SEL**

**Shared among all ranks and DRAMs in the channel**
Faults in the Memory Hierarchy

Ph.D. Final Defense
May 12, 2017
Mark Gottscho <mgottscho@ucla.edu> -- UCLA Electrical Engineering
How Much Hardware Variability is There?

Individual figures courtesy of Prof. Lucas Wanner’s UCLA PhD Defense, 2014

ITRS Projection of Power Variability

- Total Power
- Static (Sleep) Power

Measured Sleep Power in 10 Cortex M3 Processors

- 8x Variation

Measured Power Consumption of 5 Intel Core i5 CPUs [Balaji et al. HotPower’12]

- Power (W)
  - bzip2
  - 8.6
  - 8.2
  - 7.8
  - 7.4
  - 7

- Power (μW)
  - 220
  - 176
  - 132
  - 88
  - 44

- Power (mW)
  - 11.2
  - 10.6
  - 10
  - 9.4
  - 8.8
  - 8.2
  - 7.6

Measured Power Variations With Respect to Temperature

- SAM3U Sleep: 14x
- SAM3U Active: 44%

Ph.D. Final Defense
May 12, 2017
Mark Gottscho <mgottscho@ucla.edu> -- UCLA Electrical Engineering
2. ViPZonE
Motivation: Power Variability in Contemporary DRAMs

[Gottscho et al. ESL’12]

Significant power variations measured in off-the-shelf DDR3 memory

Systems could save energy by exploiting active memory power variability!
Related Work

[Bathen et al. CODES+ISSS’12, Gottscho et al. TC’15]

• **Power-aware memory systems**
  – Page allocation [Lebeck et al. ASPLOS’00]
  – Scheduler-based [Delaluz et al. DAC’02]
  – Page miss rates [Zhou et al. ASPLOS’04]
  – Adaptive architecture [Zheng et al. MICRO’08]
  – Independent DRAMs [Ahn et al. CAL’08]

• **Variation-aware circuits and systems**
  – Task scheduling [Wang et al. ICCAD’07]
  – Speed binning multicore processors [Sartori et al. ISQED’10]
  – Embedded sensing [Wanner et al. HotPower’10, DATE’11]
  – Quality adaptation [Pant et al. GLSVLSI’10]

No prior work on SW-based variation-aware memory management except VaMV [Bathen et al. DATE’12]
“ViPZonEs” have different power characteristics because they are directly mapped to DIMMs exhibiting variation.
DRAM Channel and Rank Interleaving

[Gottscho et al. TC’15]

- Assume DDR3 with:
  - 2 channels
  - 2 DIMMs per channel
  - 2 ranks per DIMM
  - All rank capacities equal

- Assume data mapping:
  - Data striped channels, DIMMs, and ranks @ cache line granularity
  - Stripe size < page size, e.g. 64B vs 4KB

Conventional interleaving is good for memory-level parallelism for within-page access patterns
Interleaving Disabled

- No striping of adjacent cache lines
- Single-page access = single-rank access
- Non-accessed ranks can enter low power states more often
  - BUT: reduced memory-level parallelism for access to adjacent cache lines

Disabling interleaving allows ViPZonE to work but could impact baseline performance
#include <stdlib.h> //Special ViPZonE GLIBC with ViPZonE Linux kernel

//...some code...

void foo(size_t arraySize) {
    int *data_ptr = NULL;

    /* Possible vip_malloc() flags:
     * One of: VIP_WRITE or VIP_READ
     * One of: VIP_HIGH_UTIL or VIP_LOW_UTIL
     * Programmer is responsible to decide
     */
    data_ptr = (int *) vip_malloc(sizeof(int)*arraySize, VIP_WRITE | VIP_HIGH_UTIL);

    //...some write-heavy operations...

vip_malloc() abstracts memory power variability in a user-friendly way
New apps can exploit ViPZonE, legacy apps work the same.
We are exploring other possible algorithms

Implementation: Lower OS Layer

Physical page allocator

[Batchen et al. CODES+ISSS’12, Gottscho et al. TC’15]

START: Receive allocation request with power parameters (write/read, high/low utilization)

Normal allocation? Expected usage?

yes

high

no

low

DMA32 required?

yes

Restrict possible DIMM zones to those < 4096 MB

no

no

DIMM zone list empty?

no

yes

Attempt allocation in lowest write/read power DIMM zone

Success?

yes

no

DIMM zone list empty?

no

yes

Attempt allocation in lowest write/read power DIMM zone with > THRESHOLD free space

Success?

yes

no

no

Remove this zone from consideration

no

yes

Remove this zone from consideration

no

yes

Zone list empty?

no

yes

Remove this zone from consideration

no

yes

Simplicity → Fast kernel 😊
Simulation Results: Promising Power Savings

[Bathen et al. CODES+ISSS’12]

- Simulations show that memory power savings could be up to ~20%
  - Using the 1GB DIMM variability data shown earlier

- Memory power savings could increase to ~30% if future DIMM variability increases to 100%
- Performance overhead was expected to be modest

**Average Power Savings (%)**
- Vanilla Linux vs. ViPZonE

**What-If Average Power Savings (%)**
- Vanilla Linux vs. ViPZonE

Detailed simulations indicate promising power savings
Measured Testbed Results

[Gottscho et al. TC’15]

### Fast2 Hardware Config

- **Execution Time (s)**
  - blackholes
  - bodytrack
  - canneal
  - facesim
  - fluidanimate
  - freqmine
  - raytrace
  - swaptions

#### Vanilla Interleaved
- 0
- 1
- 2
- 3
- 4

#### Vanilla
- 0
- 1
- 2
- 3
- 4

#### ViPZonE
- 0
- 1
- 2
- 3
- 4

### Slow2 Hardware Config

- **Execution Time (s)**
  - blackholes
  - bodytrack
  - canneal
  - facesim
  - fluidanimate
  - freqmine
  - raytrace
  - swaptions

#### Vanilla Interleaved
- 0
- 0.5
- 1
- 1.5
- 2

#### Vanilla
- 0
- 0.5
- 1
- 1.5
- 2

#### ViPZonE
- 0
- 0.5
- 1
- 1.5
- 2

### Good energy savings for non-bandwidth-intensive applications
Hypothetical Benefits for NVMs

[Gottscho et al. TC’15]

Idle power is the limiting factor for ViPZonE on current hardware
Benefits on testbed running PARSEC:

• Up to 25.1% memory power savings
• No more than 4.8% performance degradation
• Up to 27.8% memory energy savings
• Up to 50.7% hypothetical memory energy savings if NVMs used

Use when high memory-level parallelism or bandwidth not needed

\textit{Physical zoning inherently trades off benefits of striping for resource consolidation and exploitation of device variations}

Opportunistically save energy in today’s systems with no hardware changes

\textit{Through smart management of physical memory variation signatures}

• Up to 27.8% memory energy savings
• Up to 50.7% hypothetical memory energy savings if NVMs used
3. DPCS
Motivation: Increasing Process Variability Limits SRAM Voltage Scaling

[Gottscho et al. DAC’14]

Process variability $\uparrow$ $\rightarrow$ $V_T$ variations $\uparrow$ $\rightarrow$ SRAM $\sigma_{SNM}$ $\uparrow$

Limited min-VDD/yield, leakage-dominated caches, increasing portion of overall power
Related Work

- Rich body of work for fault-tolerant voltage-scalable (FTVS) cache memories in nanoscale era
  - Leakage reduction (famously: [Powell et al. ISLPED’00, Flautner et al. ISCA’02])
  - Fault tolerant circuits/architecture/ECC [Shirvani & McCluskey VLSI Test ‘99, Agarwal et al. TVLSI’05, Ansari et al. MICRO’09, Alameldeen et al. TC’11, etc.]
  - Memory power/performance scaling [Fan et al. ‘05, Deng et al. ASPLOS’11, David et al. ISCA’11, Deng et al. MICRO’12]

DPCS is the first FTVS scheme that efficiently leverages multiple voltage levels and power gating of disabled blocks, and supplements DVFS for logic
Question

[Gottscho et al. DAC’14, TACO’15]

How to optimize SRAM for the “best” system-level tradeoffs in energy, reliability, performance, & area?

There are many possible fault-tolerant cache design schemes that can be used!
Amdahl's Law Re-Formulated

\[ \text{PowerReduction}_{\text{FTVS, overall}} = \frac{1}{1 - \text{Fraction}_V^S + \text{Fraction}_{\text{FT, overhead}}^T + \frac{\text{Fraction}_V^S}{\text{PowerReduction}_V^S}} \] (1)

Save energy via

**simple & low-overhead**

fault-tolerant, voltage-scalable (FTVS)

SRAM cache architecture
Using Fault Tolerance to Achieve Lower min-VDD

Many fault-tolerant, voltage-scalable (FTVS) approaches lower min-VDD using sophisticated fault tolerance methods

Baseline Cache @ Nominal VDD – No Fault Tolerance

**PURPLE** = periphery @ full VDD
**BLUE** = SRAM cells (bright is higher VDD)
Many fault-tolerant, voltage-scalable (FTVS) approaches lower min-VDD using sophisticated fault tolerance methods [Gottscho et al. DAC’14]
Using Fault Tolerance to Achieve Lower min-VDD

Many fault-tolerant, voltage-scalable (FTVS) approaches lower min-VDD using sophisticated fault tolerance methods

ECC + Faulty Set Remapping Cache, Data Array @ 0.5 VDD

**PURPLE** = periphery @ full VDD

**BLUE** = SRAM cells (bright is higher VDD)

Min-VDD can be a misleading metric...
SRAM “Fault Inclusion Property”

NSF Variability Expedition “Red Cooper” test chips\(^1\) based on ARM Cortex M3

![Board Image with Red Cooper Test Chips]

\[\text{Gottscho et al. DAC'14, TACO'15}\]

We can now efficiently store multi-VDD fault maps with low overhead... Trade off cache capacity and power dynamically!
Architectural Mechanism

• No redundancy – just sacrifice faulty blocks as VDD scales
  • # good blocks fall off a “cliff” anyway
    • Redundancy can only do so much
  • Negligible area overhead

Simplicity is key to low overheads
Power/Capacity Scaling

[Gottscho et al. DAC’14, TACO’15]

• To adjust data array VDD
  – Temporarily stall accesses
  – Cache controller finds the blocks that will become faulty at next VDD using FM bits
    • Flush those blocks that are also Valid & Dirty
    • Then set Faulty bits, power gating them
  – Adjust VDD, wait for voltage to settle
  – Resume operations

• Two general types of runtime policies
  Static (SPCS)

---

Power gate cache blocks that are disabled for extra power savings
Static & Dynamic Power/Capacity Scaling Policies

• **Static (SPCS) Policy**: Choose single optimal VDD at design, test, or boot time
• **Dynamic Policy 1 (DPCS1)**: Based on access diversity $\leftrightarrow$ spatial locality
• **Dynamic Policy 2 (DPCS2)**: Based on average access time $\leftrightarrow$ temporal locality

**DPCS: Performance OK $\rightarrow$ lower VDD, etc.**
Adapt within and across applications
Evaluation Setup

1. 45nm SOI
2. 2 system/cache configurations for L1 & L2
3. 3 permitted VDD levels
4. SPEC CPU2006

Diagram:
- 45nm SOI
- 2 system/cache configurations for L1 & L2
- 3 permitted VDD levels
- SPEC CPU2006

[Reference: Gottscho et al. DAC’14, TACO’15]
Analytical Results

- BER vs VDD
- Probability vs VDD
- Yield vs VDD
- Normalized Static Power vs VDD
- Normalized Static Power vs Proportion of Usable Blocks
Simulation Results
Power vs. capacity tuning

Useful energy efficiency knob, complements DVFS

- SPCS: 62% (22%) total cache (system) energy savings
- DPCS: 79% (26%) total cache (system) energy savings
- DPCS: average 2.24% performance overhead
- 6% area overhead

Fault Inclusion Property

Exploit it for efficient storage of variation signatures

Opportunistic cache energy savings

Leverage variability without harming reliability or performance
Bonus Slides

4. X-Mem
Motivation: Memory is Important in Cloud Computing

- *Cloud subscribers* want to maximize app. performance
- *Cloud providers* want to minimize CapEx/OpEx given SLAs
- Needs pressure memory hierarchy: characterization is critical
- Memory benchmarking tools don’t meet key requirements
  - (A) Access pattern diversity
  - (B) Platform variability
  - (C) Metric flexibility
  - (D) Tool extensibility

We propose X-Mem, a new tool!

**Project homepage**: nanocad-lab.github.io/X-Mem

**Source code**: github.com/Microsoft/X-Mem
Idea: Exploit Memory Process Variation for Higher Performance/Watt at Lower Cost
**Idea: DIMM Provisioning**

App1 is *insensitive to memory performance* on this system. Buy cheaper, lower performance DIMMs.

But *App2 is sensitive to memory performance* on this system. Buy higher performance DIMMs, which come at higher cost.
Related Work

- My own prior work showed up to 25% power variation across DDR3 DIMMs of same specs [Gottscho et al. ESL’12]
- ViPZonE exploited power variation for energy savings [Bathen et al. CODES+ISSS’12, Gottscho et al. TC’15]
- A recent study proposed variation-aware tuning of DRAM timings [Chandrakesar et al. DATE’14]
  - They found up to 25-35% latency and/or bandwidth improvements possible at DRAM level
  - Problems: Their approach is not scalable & system-level impact was not evaluated
  - Recently followed up by AL-DRAM [Lee et al. HPCA’15], which was done concurrently with this work

Question: How to evaluate efficacy of variation-aware DRAM performance tuning?
Develop a new software tool that can evaluate memory variation-aware solutions for improving energy efficiency and support other uses by the community.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>STREAM v5.10 [13]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>○</td>
<td>○</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>STREAM2 v0.1 [14]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>○</td>
<td>○</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Imbench3 [15]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>○</td>
<td>○</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TinyMemBench v0.3.9 [16]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>○</td>
<td>○</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>mlc v2.3 [17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>○</td>
<td>○</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>X-Mem v2.2.3 [18], [19]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>○</td>
<td>○</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
X-Mem Design

[Gottscho et al. ISPASS’16]

• Object-oriented C++
• Caches through DRAM
• (A) Access pattern diversity
• (B) Platform variability
• (C) Metric flexibility
• (D) Tool extensibility
• Open-source
• User-friendly CLI & documentation

Latest SW, documentation, data available @
https://nanocad-lab.github.io/X-Mem
X-Mem Feature: *(A) Access Pattern Diversity*

[Gottscho et al. ISPASS'16]

### 6 Degrees of Freedom

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Access granularity</td>
</tr>
<tr>
<td>2.</td>
<td>Access types</td>
</tr>
<tr>
<td>3.</td>
<td>Access patterns</td>
</tr>
<tr>
<td>4.</td>
<td>Parallelism</td>
</tr>
<tr>
<td>5.</td>
<td>Page sizes</td>
</tr>
<tr>
<td>6.</td>
<td>Topologies</td>
</tr>
</tbody>
</table>

- **(D) Tool Extensibility**: Developers can easily add specialized patterns through new benchmark kernel functions
X-Mem Feature:  
(B) Platform Abstractions  

[Gottscho et al. ISPASS’16]

<table>
<thead>
<tr>
<th></th>
<th>OS Support</th>
<th>Windows, GNU/Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Architectural support</td>
<td>x86, x86-64 with(out) AVX SIMD extensions, ARMv7 with(out) NEON SIMD extensions, ARMv8</td>
</tr>
</tbody>
</table>

- All OS and hardware-specific implementation details are abstracted via OOP techniques and preprocessor macros
  - Includes benchmark kernels, high-resolution timers, power measurement etc.
- Portable SCons-based build system using Python
- **(D) Tool Extensibility:** Ports to other OSes and architectures possible with relatively little effort. Enables apples-to-apples memory hierarchy comparisons.
**X-Mem Feature:**

**(C) Metric Flexibility**

[Gottscho et al. ISPASS’16]

- **Performance:** X-Mem measures real performance of the memory hierarchy as could be seen by an application
  - Average aggregate throughput
  - Average unloaded latency
  - Average loaded latency

- **Power**
  - Average and peak DRAM power
  - Simple software hooks for custom power measurement hardware

- **(D) Tool Extensibility:** shared-data throughput, percentile statistics, variance, data-aware power/performance bookkeeping for NVMs *etc.*
# Experimental Platform Details

[Gottscho et al. ISPASS’16]

<table>
<thead>
<tr>
<th>System Name</th>
<th>ISA</th>
<th>CPU</th>
<th>No. Cores</th>
<th>CPU Freq.</th>
<th>L1$</th>
<th>L2$</th>
<th>L3$</th>
<th>$ Blk.</th>
<th>Process</th>
<th>OS</th>
<th>NUMA</th>
<th>ECC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop</td>
<td>x86-64 w/ AVX</td>
<td>Intel Core i7-3820 (Sandy Bridge-E)</td>
<td>4</td>
<td>3.6 GHz*, 1.2 GHz</td>
<td>split, private, 32 KiB, 8-way</td>
<td>private, 256 KiB, 8-way</td>
<td>shared, 10 MiB, 20-way</td>
<td>64 B</td>
<td>32 nm</td>
<td>Linux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>x86-64 w/ AVX2</td>
<td>Dual Intel Xeon E5-2600 v3 series (Haswell-EP)</td>
<td>12 per CPU</td>
<td>2.4 GHz</td>
<td>split, private, 32 KiB, 8-way</td>
<td>private, 256 KiB, 8-way</td>
<td>shared, 30 MiB, 20-way</td>
<td>64 B</td>
<td>22 nm</td>
<td>Win.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Microserver</td>
<td>x86-64</td>
<td>Intel Atom S1240 (Centerton)</td>
<td>2</td>
<td>1.6 GHz</td>
<td>split, private, 24 KiB 6-way data, 32 KiB 8-way inst.</td>
<td>private, 512 KiB, 8-way</td>
<td>-</td>
<td>64 B</td>
<td>32 nm</td>
<td>Win.</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>PandaBoard (ES)</td>
<td>ARMv7-A w/ NEON</td>
<td>TI OMAP 4460 (ARM Cortex-A9)</td>
<td>2</td>
<td>1.2 GHz</td>
<td>split, private, 32 KiB, 4-way</td>
<td>shared, 1 MiB</td>
<td>-</td>
<td>32 B</td>
<td>45 nm</td>
<td>Linux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AzureVM</td>
<td>x86-64</td>
<td>AMD Opteron 4171 HE</td>
<td>4</td>
<td>2.1 GHz</td>
<td>split, private, 64 KiB, 2-way</td>
<td>private, 512 KiB, 16-way</td>
<td>shared, 6 MiB, 48-way</td>
<td>64 B</td>
<td>45 nm</td>
<td>Linux</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>AmazonVM</td>
<td>x86-64 w/ AVX2</td>
<td>Intel Xeon E5-2666 v3 (Haswell-EP)</td>
<td>4</td>
<td>2.9 GHz</td>
<td>split, private, 32 KiB, 8-way</td>
<td>private, 256 KiB, 8-way</td>
<td>shared, 25 MiB, 20-way</td>
<td>64 B</td>
<td>22 nm</td>
<td>Linux</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ARMServer</td>
<td>ARMv7-A</td>
<td>Marvell Armada 370 (ARM Cortex-A9)</td>
<td>4</td>
<td>1.2 GHz</td>
<td>split, private, 32 KiB, 4-way / 8-way (1/D)</td>
<td>private, 256 KiB, 4-way</td>
<td>-</td>
<td>32 B</td>
<td>?</td>
<td>Linux</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Name</th>
<th>Config. Name</th>
<th>Memory Type</th>
<th>No. Channels</th>
<th>DPC</th>
<th>RPD</th>
<th>DIMM Capacity</th>
<th>Chan. MT/s</th>
<th>nCAS - clk (tCAS - ns)</th>
<th>nRCD - clk (tRCD - ns)</th>
<th>nRP - clk (tRP - ns)</th>
<th>nRAS - clk (tRAS - ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop*</td>
<td>1333 MT/s, Nominal Timings 4C</td>
<td>DDR3 U</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2 GiB</td>
<td>1333</td>
<td>9 (13.5 ns)</td>
<td>9 (13.5 ns)</td>
<td>11 (16.5 ns)</td>
<td>24 (36.0 ns)</td>
</tr>
<tr>
<td>Desktop</td>
<td>1333 MT/s, ~33% Slower Timings 4C</td>
<td>DDR3 U</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2 GiB</td>
<td>1333</td>
<td>12 (18.0 ns)</td>
<td>12 (18.0 ns)</td>
<td>15 (22.5 ns)</td>
<td>32 (48.0 ns)</td>
</tr>
<tr>
<td>Desktop*</td>
<td>800 MT/s, Nominal Timings 4C</td>
<td>DDR3 U</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2 GiB</td>
<td>800</td>
<td>7 (17.5 ns)</td>
<td>7 (17.5 ns)</td>
<td>8 (20.0 ns)</td>
<td>16 (40.0 ns)</td>
</tr>
<tr>
<td>Desktop</td>
<td>800 MT/s, ~33% Slower Timings 4C</td>
<td>DDR3 U</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2 GiB</td>
<td>800</td>
<td>10 (25.0 ns)</td>
<td>10 (25.0 ns)</td>
<td>11 (27.5 ns)</td>
<td>22 (55.0 ns)</td>
</tr>
<tr>
<td>Desktop*</td>
<td>1333 MT/s, Nominal Timings 1C</td>
<td>DDR3 U</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2 GiB</td>
<td>1333</td>
<td>12 (18.0 ns)</td>
<td>12 (18.0 ns)</td>
<td>15 (22.5 ns)</td>
<td>32 (48.0 ns)</td>
</tr>
<tr>
<td>Desktop</td>
<td>1333 MT/s, ~33% Slower Timings 1C</td>
<td>DDR3 U</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2 GiB</td>
<td>800</td>
<td>7 (17.5 ns)</td>
<td>7 (17.5 ns)</td>
<td>8 (20.0 ns)</td>
<td>16 (40.0 ns)</td>
</tr>
<tr>
<td>Server*</td>
<td>1333 MT/s, Nominal Timings</td>
<td>DDR3 R</td>
<td>4 per CPU</td>
<td>1</td>
<td>2</td>
<td>16 GiB</td>
<td>1333</td>
<td>9 (13.5 ns)</td>
<td>9 (13.5 ns)</td>
<td>9 (13.5 ns)</td>
<td>24 (36.0 ns)</td>
</tr>
<tr>
<td>Server</td>
<td>1333 MT/s, ~33% Slower Timings</td>
<td>DDR3 R</td>
<td>4 per CPU</td>
<td>1</td>
<td>2</td>
<td>16 GiB</td>
<td>1333</td>
<td>12 (18.0 ns)</td>
<td>12 (18.0 ns)</td>
<td>12 (18.0 ns)</td>
<td>32 (48.0 ns)</td>
</tr>
<tr>
<td>Server</td>
<td>1600 MT/s, Nominal Timings</td>
<td>DDR3 R</td>
<td>4 per CPU</td>
<td>1</td>
<td>2</td>
<td>16 GiB</td>
<td>1600</td>
<td>11 (17.5 ns)</td>
<td>11 (17.5 ns)</td>
<td>11 (17.5 ns)</td>
<td>29 (36.25 ns)</td>
</tr>
<tr>
<td>Server</td>
<td>1600 MT/s, ~33% Slower Timings</td>
<td>DDR3 R</td>
<td>4 per CPU</td>
<td>1</td>
<td>2</td>
<td>16 GiB</td>
<td>1600</td>
<td>15 (18.75 ns)</td>
<td>15 (18.75 ns)</td>
<td>15 (18.75 ns)</td>
<td>38 (47.5 ns)</td>
</tr>
<tr>
<td>Server</td>
<td>1867 MT/s, Nominal Timings</td>
<td>DDR3 R</td>
<td>4 per CPU</td>
<td>1</td>
<td>2</td>
<td>16 GiB</td>
<td>1867</td>
<td>13 (19.25 ns)</td>
<td>13 (19.25 ns)</td>
<td>13 (19.25 ns)</td>
<td>34 (36.42 ns)</td>
</tr>
<tr>
<td>Server</td>
<td>1867 MT/s, ~33% Slower Timings</td>
<td>DDR3 R</td>
<td>4 per CPU</td>
<td>1</td>
<td>2</td>
<td>16 GiB</td>
<td>1867</td>
<td>18 (19.28 ns)</td>
<td>18 (19.28 ns)</td>
<td>18 (19.28 ns)</td>
<td>46 (49.27 ns)</td>
</tr>
</tbody>
</table>
Case Study 1: Characterization of the Memory Hierarchy for Cloud Subscribers

[Gottscho et al. ISPASS’16]

- Cloud subscribers should measure and leverage:
  - Cache micro-architecture
  - System-level memory management

- Understanding these enables improved application performance:
  - Workload partitioning among threads?
  - Working set size per thread?
  - Data access patterns?
  - When, where, and how to allocate memory?

**Graph:**
- **X-axis:** Average Main Memory Total Latency (ns/access)
- **Y-axis:** Main Memory Total
- **Legend:**
  - 1333 MT/s 4C Bound
  - CPU Node 0, Memory Node 0
  - CPU Node 1, Memory Node 0
  - CPU Node 0, Memory Node 1
  - CPU Node 1, Memory Node 1
  - Local Memory (Large Pages)
  - Remote Memory (Large Pages)
  - 2x 8 GT/s QPI Bound

**Legend Note:**
X-Mem can uncover performance effects that only manifest at a system level.
Case Study 1: Characterization of the Memory Hierarchy for Cloud Subscribers

[Gottscho et al. ISPASS ’16]

Desktop Platform Insights:
Memory Hierarchy Landscape

X-Mem can quantify various aspects of performance for cache and memory architectures.
Case Study 1: Characterization of the Memory Hierarchy for Cloud Subscribers

[Gottscho et al. ISPASS’16]

Desktop Platform Insights:
L1 Data Cache Architecture

X-Mem can reveal hidden details of cache and memory micro-architectures
Case Study 2: Cross-Platform Insights for Cloud Subscribers

[Gottscho et al. ISPASS’16]

- Cloud subscribers can use X-Mem to directly compare memory performance:
  - x86 vs. ARM instruction set
  - Virtual vs. physical machines
  - Wimpy vs. brawny hardware
- This capability enables subscribers to:
  - Choose a target cloud platform that best suits workload characteristics

X-Mem can perform apples-to-apples comparisons between diverse platforms
Case Study 2: Cross-Platform Insights for Cloud Subscribers

[Gottscho et al. ISPASS’16]

Cross-Platform Insights:
Main Memory Loaded Latency

X-Mem can perform apples-to-apples comparisons between diverse platforms
Case Study 3: Impact of Variation-Aware Tuning of Platform Configurations for Cloud Providers

[Gottscho et al. ISPASS’16]

- Cloud providers can use X-Mem to evaluate the sensitivity of system-level performance to memory configurations:
  - Number of DRAM channels
  - DRAM timing parameters
  - Analyze throughput, unloaded and loaded latency, different access patterns, etc.

- This capability enables providers to:
  - Optimally configure their platforms for different types of workloads
  - Maximize performance/$, minimize TCO, etc.

X-Mem can facilitate studies of platform configurations and impact of variation-aware tuning.
Case Study 3: Impact of Variation-Aware Tuning of Platform Configurations for Cloud Providers

[Gottscho et al. ISPASS’16]

Desktop @ 3.6 GHz Platform
Case Study 3: Impact of Variation-Aware Tuning of Platform Configurations for Cloud Providers

[Gottscho et al. ISPASS’16]

Remote access: Up to 45% slower

<table>
<thead>
<tr>
<th>Platforms</th>
<th>Mem. Channel Frequency</th>
<th>Timings</th>
<th>1867 MT/s</th>
<th>1867 MT/s</th>
<th>1600 MT/s</th>
<th>1600 MT/s</th>
<th>1333 MT/s</th>
<th>1333 MT/s</th>
<th>800 MT/s</th>
<th>800 MT/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server (NUMA Local, Lrg. Pgs.)</td>
<td>91.43</td>
<td>91.54</td>
<td>91.66</td>
<td>95.74</td>
<td>91.99</td>
<td>97.61</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Server (NUMA Remote, Lrg. Pgs.)</td>
<td>126.51</td>
<td>128.54</td>
<td>129.62</td>
<td>139.25</td>
<td>133.59</td>
<td>141.69</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Desktop 4C @ 3.6 GHz</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>73.33</td>
<td>81.91</td>
<td>97.21</td>
<td>110.89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desktop 1C @ 3.6 GHz</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>72.38</td>
<td>80.94</td>
<td>97.36</td>
<td>109.56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desktop 4C @ 1.2 GHz</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>109.65</td>
<td>118.25</td>
<td>131.86</td>
<td>145.76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desktop 1C @ 1.2 GHz</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>108.44</td>
<td>117.09</td>
<td>131.85</td>
<td>144.46</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure: Sensitivity of unloaded latency (ns/access) w.r.t. CPU & DDR3 frequency, DRAM timing, # DDR3 channels

CPU underclocked 3X: 50% higher DRAM lat.

Benchmark | Config. | 1T  | 2T  | 3T  | 4T  |
-----------|---------|-----|-----|-----|-----|
| canneal   | 1333 MT/s 4C* | 9.74 | 9.02 | 8.83 | 8.89 |
| canneal   | 800 MT/s 1C   | 9.90 | 9.29 | 8.38 | 7.83 |
| streamcluster | 1333 MT/s 4C* | 11.14 | 11.53 | 11.82 | 12.24 |
| streamcluster | 800 MT/s 1C   | 8.10 | 5.93 | 2.63 | 1.24 |

Figure: Impact of 33% slower DRAM timings

X-Mem shows that variation-aware DRAM perf. tuning makes sense only when BW bottlenecks are removed

Benchmarks are memory BW starved; relative impact of DRAM timings is LESS w/ more threads

CPU & DDR3 frequency, DRAM timing, # DDR3 channels

Remote access: Up to 45% slower

# channels: no impact

DRAM timings 33% slower $\Rightarrow$ up to 12% slower overall

Memory has enough BW; benchmarks appear latency-bound
Summary

Gottscho et al. ISPASS’16

New flexible tool for characterizing memory systems

**Surpasses capabilities of all prior tools**

Several key features enable broad usability

(A) Access pattern diversity, (B) Platform variability,

(C) Metric flexibility, (D) Tool extensibility

Characterization is critical to opportunistic memory systems

Data-driven exploitation of performance and power variability
Bonus Slides

5. Performability
Motivation & Related Work

• Datacenters are growing in size
  – Prolific demand for memory
  – Increasing DRAM error rates observed in the field
    [Li et al. ’10, Schroeder et al. CACM’11, Sridharan & Liberty ‘12, Hwang et al. ASPLOS’12,
    DeBardeleben et al. SELSE’14, Meza et al. DSN’15]

• Memory errors cause significant loss of availability and higher TCO [Meza et al. DSN’15, Nikolaou et al.
  MICRO’15]

• …Even corrected errors do! [Meza et al. DSN’15]
  – Why? Apparently, performance impacts caused by “avalanches” of errors

Need controlled analysis of memory errors to answer the field studies’ call for action
Memory is Important in Cloud

• Main memory in cloud:
  – Impacts providers: capital and operational expenditures
  – Impacts subscribers: application performance

• Deep understanding of memory system can help minimize cost-to-benefit ratio for both

• DRAM faults and fault-tolerance techniques affect:
  – Performance and availability of servers
  – Total cost of ownership (TCO) of datacenter

How to optimally provision, manage, and retire DRAM to minimize datacenter TCO while satisfying performance and availability SLAs?
DRAM Fault Models

Fault Classification

Granularity
- Socket
- Channel
- Rank
- Chip
- Bank
- Row
- Column
- Multi-bit
- Single-bit

Time
- Permanent
- Intermittent
- Single-events

Space
- Within-pages
- Neighboring pages
- Across pages
DRAM Fault Tolerance Techniques Available on Current Cloud Servers (Haswell-EP)

Abstractions

Active SW Mgmt.

Prevention
- Page retirement (PFA)

Recovery
- Clean page fault (pseudo-checkpoint) – req. kernel mods

HW/FW Mgmt.

Reactive
- Rank sparing
- Channel mirroring
- Bit-steering/device tag

Active
- Demand scrubbing
- Patrol scrubbing

Detection Only
- Address/command parity
- Data parity

Detection and Correction
- SECDED ECC
- ChipKill/SDDC ECC

When and how to use these techniques?
Approach: How to Study Memory Resiliency in the Real World?

- Faults are considered rare events
  - Reliability engineering is a challenge

Four approaches to study resiliency

<table>
<thead>
<tr>
<th>Method</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field data</td>
<td>Ground truth, big-data statistics</td>
<td>Insight, post-facto analysis</td>
</tr>
<tr>
<td>Accelerated testing</td>
<td>Accuracy, hardware-in-the-loop</td>
<td>Cost, design space</td>
</tr>
<tr>
<td>Simulated modeling</td>
<td>Transparency, control</td>
<td>Time, scalability</td>
</tr>
<tr>
<td>Fault injection</td>
<td>Tractable, pre-deployment</td>
<td>Accuracy, assumptions</td>
</tr>
</tbody>
</table>

Unfortunate tetrahedron: Choose 1 face
Measuring Performance Impact of Injected Memory Errors

**Steps:**
1. Virtual → physical address
2. Inject faults

**Steps:**
1. Trigger pings faulty memory
2. Reference app measures performance
Performance of Batch Applications with SMI Errors

Average Slowdown (%)

# errors/sec - # benchmark copies - benchmark name
Application Thread-Servicing Time

• Modeled impact of corrected errors on general application performance
• Model combinations
  • System: uniprocessor vs. multiprocessor
  • Error-reporting scheme: broadcast vs. single-issue
  • Application type: batch (throughput-oriented) vs. interactive (latency-oriented)
• Models are built on derived model of application thread-servicing time (ATST) in presence of errors

Application thread-servicing time (ATST)

\[ t_{\text{service}} = \gamma + \tau_1 + \tau_2 + \tau_3 \]
Framework

Input

Fault Parameters → Fault Models → Fault Tolerance Techniques

Availability Model

Availability Metrics

Performance Metrics

Worst case (errors occur) → Performance Models

Common case (no errors occur) → Benchmarks

TCO Model and Opt.

Collect Fault Data From Field and Vendors

Closing the loop

Memory Provisioning Decisions

Goal for cloud provider

Project focus

Memory Provisioning Decisions

Ph.D. Final Defense
May 12, 2017
Mark Gottscho <mgottscho@ucla.edu> -- UCLA Electrical Engineering
Outlook

- Datacenter operations
  - Common/worst-case performance impact of memory errors
  - Optimal servicing of faulty hardware
  - Variation-aware memory provisioning

- System design
  - Efficient error-reporting architectures
  - Modeling impact of corrected errors on different applications

Understanding impact of corrected errors is useful for opportunistic memory provisioning
Motivation: Memory Errors are a Major Problem

- **System-level effects from embedded to HPC**
  - System crashes
  - Silent data corruption
- **DRAM reliability worsens** with density
  - Google: 70,000 FIT/Mb in commodity DRAM; 8% of modules affected per year; 4% of servers crash per year [Schroeder CACM’11]
  - Facebook: 2.5% of machines see DRAM errors per month [Meza DSN’15]
- **SRAM stops working** at low voltage
  - 6X fault rate measured from 600mV to 525mV [Gottscho TACO’15]
- **Flash wears out** with usage
  - NASA’s Opportunity Mars rover had to reformat its flash in 2014
- **STT-RAM is unpredictable**
  - Stochastic write & thermal instability [Zhao Microelec. Rel.’12]
- **Memory errors will continue to be a challenge!**

---

Ph.D. Final Defense
May 12, 2017
Mark Gottscho <mgottscho@ucla.edu> -- UCLA Electrical Engineering
Motivation & Related Work

Historically separate abstractions:

- Error-correcting codes (ECCs)
  - e.g., SECDED [Hsiao IBM Journal’70], DECTED, ChipKill [Dell IBM’97], SEC-DAEC [Dutta et al. VLSI Test’07], VS-ECC [Alameldeen et al. ISCA’11]

- System-level fault tolerance techniques
  - Checkpoint & recovery
  - Mirroring/sparing

Is there room for anything in between?
Number of Candidate Codewords

[Gottscho '17]

• Surprisingly small number of candidate codewords for any ECC that corrects $t$ symbol-wise errors and detects $t+1$ errors.
• We proved that the average number of candidate codewords is:

$$\mu(n, t, q) = \frac{\binom{2t+2}{t+1}W_q(2t + 2)}{\binom{n}{t+1}(q - 1)^{t+1}} + 1.$$

  - $W_q(2t + 2)$: number of min. weight codewords.
  - $\binom{2t+2}{t+1}$: number of DUEs distance of exactly $(t + 1)$ from each min. weight codeword.
  - $\binom{n}{t+1}(q - 1)^{t+1}$: number of ways to produce a min. weight DUE.
  - $+1$: for the original correct message.
Lightweight Hash Implementation: SECDED

Memories Transfer Block (586 bits)

Cacheline Hash (4 or 8 bits)
- Only decoded if DUE in a codeword
- Stored using extra x1 chip (no performance hit)

Burst Length (8 beats)

Cacheline Payload (512 bits)

Codeword (64-bit message + 8-bit parity)

DRAM I/O (4-bit)

Message (64-bit)

Parity (8-bit)

Custom DRAM Channel (73-bit)

Hash I/O (1-bit)
Data Recovery Policies Comparison

[<Gottscho '17>]

- Hsiao $[72,64,4]^2$
- DECTED $[79,64,6]^2$
- SSCDSD $[36,32,4]_16$

![Bar chart comparing data recovery policies](chart.png)
What if the Lightweight Hash has an Error?

[Gottscho ‘17]

• **Outcome 1 (likely):** hash does not match any candidate, fall back to normal SDECC

• **Outcome 2 (unlikely):** hash collides with wrong candidate, guaranteed miscorrection
  • 0.003% chance for 16-bit hash

**Fault Classification**

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Socket</td>
<td>• Permanent</td>
<td>• Within-pages</td>
</tr>
<tr>
<td>• Channel</td>
<td>• Intermittent</td>
<td>• Neighboring pages</td>
</tr>
<tr>
<td>• Rank</td>
<td>• Single-events</td>
<td>• Across pages</td>
</tr>
<tr>
<td>• Chip</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Bank</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Row</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Column</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Multi-bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Single-bit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory Controller

DIMM (Socketed)

Channels

Bank (Independent, Lockstep within rank)

Array

Row Buffer

Columns

Location

Memory Controller

DRAM Chip

64b DATA (+8b ECC parity)

Channel: CMD, ADDR, CLK, RANK_SEL
(Shared among all ranks and DRAMs in the channel)
Suppose 20% of all double-chip DUEs also have random 16-bit hash error

Then actual DUE recovery rate:
99.9999% → 97.2767%

Speedup: 15.6%

Avg. util: 97.6%

MTT ind. MCE: 5.5 Mhours
# System-Level Benefits

[Gottscho ‘17]

<table>
<thead>
<tr>
<th>scheme/hash size</th>
<th>opt. chkpt. intvl. [283]</th>
<th>speedup</th>
<th>util.</th>
<th>MTT ind. MCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>6.6 hours</td>
<td>-</td>
<td>84.4%</td>
<td>N/A</td>
</tr>
<tr>
<td>SDECC/none</td>
<td>18.4 hours</td>
<td>12.0%</td>
<td>94.5%</td>
<td>2.9 Khours</td>
</tr>
<tr>
<td>SDECC/4-bit</td>
<td>48.2 hours</td>
<td>16.1%</td>
<td>98.0%</td>
<td>48.0 Khours</td>
</tr>
<tr>
<td>SDECC/8-bit</td>
<td>272.9 hours</td>
<td>18.1%</td>
<td>99.7%</td>
<td>2.2 Mhours</td>
</tr>
<tr>
<td>SDECC/16-bit</td>
<td>N/A</td>
<td>18.5%</td>
<td>100%</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Bonus Slides

7. ViFFTto
Motivation

• Memory resiliency is a key challenge for embedded edge devices in IoT
• Conventional EDAC techniques are too costly and inefficient
• Many embedded systems lack “real” OS w/ virtual memory support

Opportunistic solution for coping with hard memory defects could reduce cost of IoT
SDELC Architecture

[Gottscho '17]
Results: Hard Faults

Test Chip 1

Test Chip 2

SHA packed in inst SPM
SDELC Instruction Recovery Insights

[Gottscho '17]

Relative Frequency

<table>
<thead>
<tr>
<th>Bit</th>
<th>31</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>20</th>
<th>19</th>
<th>15</th>
<th>14</th>
<th>12</th>
<th>11</th>
<th>7</th>
<th>6</th>
<th>0</th>
<th>-1</th>
<th>-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type-U</td>
<td>imm[31:12]</td>
<td>rd</td>
<td>opcode</td>
<td>parity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type-UJ</td>
<td>imm[20:10:11]</td>
<td>rd</td>
<td>opcode</td>
<td>parity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type-I</td>
<td>imm[11:0]</td>
<td>rs1</td>
<td>funct3</td>
<td>rd</td>
<td>opcode</td>
<td>parity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type-S</td>
<td>imm[11:5]</td>
<td>rs2</td>
<td>rs1</td>
<td>funct3</td>
<td>imm[4:0]</td>
<td>opcode</td>
<td>parity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type-R</td>
<td>funct7</td>
<td>rs2</td>
<td>rs1</td>
<td>funct3</td>
<td>rd</td>
<td>opcode</td>
<td>parity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type-R4</td>
<td>rs3</td>
<td>funct2</td>
<td>rs2</td>
<td>rs1</td>
<td>funct3</td>
<td>rd</td>
<td>opcode</td>
<td>parity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chunk

<table>
<thead>
<tr>
<th>Parity-Check</th>
<th>C1 (shared)</th>
<th>C2 (shared)</th>
<th>C3 (shared)</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C3</th>
<th>C2</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>00</td>
<td>11111</td>
<td>00000</td>
<td>111</td>
<td>1111</td>
<td>11111111</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>00000</td>
<td>11</td>
<td>00000</td>
<td>11111</td>
<td>000</td>
<td>1111</td>
<td>11111111</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>11111</td>
<td>00</td>
<td>00000</td>
<td>11111</td>
<td>111</td>
<td>0000</td>
<td>11111111</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Relative Frequency

Black line is geometric mean over all benchmarks

Avg. Inst. Recover. Rate

Position of Single-Bit Error in Instruction Codeword
8. Conclusion and Directions for Future Work
Many-Core Checkerboard Architecture for the Cloud

- Motivation: Datacenter-on-Chip
- Datacenter & cloud apps are often task-parallel or data-parallel
- But currently, they are deployed on commodity hardware
  - General-purpose
  - Also designed for other classes of applications
- Given the scale of datacenter services, it makes sense to consider customized architectures
- Question: Why do we need many-core processors to share a unified address space?
  - Corollary: How could we build better datacenter-specialized chips that allow for greater scale-out capabilities?
  - Not necessarily the wimpy node idea…
- Question: How could we build datacenter-specialized chips given the opportunities presented by emerging device and integration technology?
2D Checkerboard Architecture: High-Level

Heterogeneous memory tiles:
1) eDRAM
2) STT-RAM
3) 3D X-Point

Minimized data movement:
- On-chip dense scratchpads
- Local access by physically adjacent compute tiles only
- 4-way arbiter per memory tile
- No cache hierarchy
- No virtual memory
- Thread communication via core-to-core lightweight message passing, control migrations, locally-shared memory with up to 4 tiles

Heterogeneous compute tiles:
1) Performance CPU
2) Low power CPU
3) Accelerators
3) Field-programmable fabric
High performance tiles near edges for I/O accessibility, thermal footprint
Dielet Example

Dense inter-dielet wiring pitch
- high-BW, low-latency integration of disparate technologies
- Simplified dielet I/O
- Highly modular
- Heterogeneous compute and memory
- System-on-wafer
  - Less compromised than SoC
  - Denser, faster, lower energy than PCB
3D Checkerboard Architecture: High-Level

---

**Checkerboard Architecture**
Single-Floorplan, 3D-Stackable, Many-Core, Heterogeneous

- **Compute Tile**
- **Memory Tile**
- **6-port Memory Arbiter (A)**
- **Point-to-Point Memory Interface**
- **Downwards (-) and Upwards (+)** TSV-Based Cross-Layer Memory Interfaces

**Heterogeneous compute tiles:**
1) Performance CPU
2) Low power CPU
3) Accelerators
3) Field-programmable fabric

**Heterogeneous memory tiles:**
1) eDRAM
2) STT-RAM
3) 3D X-Point

---

Terraced 3D stacking with identical floorplans in each layer

Increased surface area for package heat dissipation, power delivery, & I/O

---

Ph.D. Final Defense
May 12, 2017
Mark Gottscho <mgottscho@ucla.edu> -- UCLA Electrical Engineering
Programming a Checkerboard Architecture

- Overlap-Clustered memory address space
  - Compute tiles: can only address adjacent memory tiles
  - Memory tiles: can only be accessed by adjacent compute tiles
  - *Remote memory access is forbidden*
    - Instead, HW-migrate lightweight threads as needed
  - *No virtual memory*
    - Instead, build relocatable programs – each memory tile has a base address offset
    - In-memory tile access control (e.g., forbid access from West compute tile)
  - *Allocate memory tiles, which come with adjacent compute tiles*
    - …instead of compute threads allocating memory, as is normally done

- System Benefits
  - Clusters have completely independent memory hierarchies
  - Lightweight or eliminated cache coherence – local tiles have shared nearby memory
  - Reduced global communication
  - Many-core scalability for running many task/data-parallel workloads
  - “Datacenter-on-chip” – well suited for isolated multi-core VMs in the cloud
Programming a Checkerboard Architecture: Example

VM 1 mapped to cluster 1

VM 2 mapped to cluster 2

Memory Tile

Compute Tile

VM 1 tries to access private VM 2 mem tile 1
-- In-mem ACL protection

Compute tile E needs to access memory tile 0
-- Lightweight HW thread migration/swap between E & C
-- Then access North mem tile from C