A Fault-Aware Cache Management Policy for CMPs Operating at Ultra-Low Voltages

Alexandra Ferrerón-Labari, Jesús Alastruey-Benedé, Darío Suárez Gracia, Teresa Monreal-Arnal, Pablo Ibáñez-Marín, Víctor Viñals-Yúfera

Grupo de Arquitectura de Computadores
Departamento de Informática e Ingeniería de Sistemas
Instituto Universitario de Investigación de Ingeniería de Aragón
Universidad de Zaragoza
How to Cope with Errors: Natural Redundancy

• Natural replication of blocks in inclusive cache hierarchies:
  
  – Exploit the memory hierarchy organization

  – → BDOT-FA
    a fault-aware cache mgt. policy
System Overview

• 8-core tiled CMP

Ultraspred III Plus, in-order, 1 instr/cycle, single-threaded, 1GHz at V_{dd} 0.5 V

Private, 64 KB data and inst. caches, 4-way, 64 B block size, LRU, 2-cycle hit access time

Shared, 1 bank/tile, 1 MB/bank, 16-way, 64 B block size, Pseudo-LRU, Inclusive, 8-cycle hit access time

MESI, directory-based, full-map distributed among LLC banks

Mesh, 2 Virtual Networks (requests & replies), 16-byte flit size, 2-stage routers, 1-cycle latency hop

2 MC, DDR3 1333 MHz, 2 channels, 8 Gb/channel, 8 banks, 8 KB page size, open page policy
System Overview

- 8-core tiled CMP

L1 caches built with robust cells (fault-free) [KH09];
e.g., 8T SRAM cells [CMN+08]

LLC cache built with regular 6T cells (prone to errors)


SRAM Failure Model [ZKG+10]

- 6 6T SRAM cells, 32 nm, target $V_{\text{nth}} = 0.5$ V

Not considered
(~ 0% available entries at 0.5 V)

<table>
<thead>
<tr>
<th>Relative Area</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
</tr>
</thead>
<tbody>
<tr>
<td>% non-faulty</td>
<td>0.0</td>
<td>9.9</td>
<td>27.8</td>
<td>35.8</td>
<td>50.6</td>
<td>59.9</td>
</tr>
</tbody>
</table>

System model
- Random fault maps for LLC
- Monte Carlo simulations:
  ensure error < 5 % with confidence $(1 - \alpha) = 0.95$

++ Area
++ Power
++ Reliability

Experimental Set-up

• Full-system simulation:
  Simics + GEMS + DRAMSim2 + McPAT

• Workloads
  – 20 multiprogrammed mixes:
    Random combinations of the 29 SPEC CPU 2006 applications
  – 4 parallel applications:
    PARSEC applications (sim-large, LLCMPKI $\geq 1.0$)
    canneal, ferret, streamcluster, vips
### Coping with SRAM Errors

- **Block Disabling (BD) [S89]:**
  - disable cache entries with faults

  ![LLC Tag vs LLC Data Diagram](chart)

  - Simple and cheap (1 bit)
  - Performance degrades very fast as the number of faulty entries increases

---

Performance of BD: LLC MPKI

SPEC CPU 2006 multiprogrammed mixes

Normalized MPKI

With respect to an “unrealistically” fault-free LLC (“Robust”)
BD and Inclusion Victims

Inclusive hierarchy → all blocks in L1 must be mapped in LLC as well

Only 2 operative ways in set i
BD and Inclusion Victims

Only 2 operative ways in set i
The associativity of LLC limits the associativity of L1

Unused fault-free L1 entries → Performance degrades!
Avoid Inclusion Victims Increase?

• Observation:
  BD limits associativity of the private levels
  BUT, from the coherence perspective, only directory inclusion is needed

• Idea:
  Allow blocks to be mapped in L1 by keeping only the tag in the LLC (with its coherence information) via resilient tag array
  (e.g., 8T SRAM cells – 2% area overhead)
BDOT: Block Disabling with Operational Tags

- Associativity seen by private levels is restored
- Requests to T entries: serviced by L1-to-L1 transfers or forwarded to main memory
- Replacement unchanged: no distinction between T and D entries
Performance of BDOT: LLC MPKI

SPEC CPU 2006 multiprogrammed mixes

LLC MPKI increases for all cell types but C2
++ off-chip traffic
++ overall energy consumption

Only improvements when many faulty entries in LLC

BD–NRU
BDOT–NRU

Normalized MPKI

C6 C5 C4 C3 C2

23.9% 30.6% 44.1% 56.7% 73.8% 90.4% 136.5%
BDOT Limitations

• BDOT only improves performance when there are many faulty entries in the LLC

• Observation:
  A block used more than once allocated to a T entry always misses in LLC data array, while BD would allocate a data entry for it

• Idea:
  “Useful” blocks should be allocated to D entries → Maximize on-chip hits and on-chip content
Cache Management Policy for BDOT

• **Maximize on-chip hits:**
  Allocate blocks more likely to be used in the near future to D entries → **Reuse** as future use detector:
  – Reuse locality in LLC [AIV+13]: Blocks accessed at least twice tend to be reused many times in the near future, and recently reused blocks are more useful than those reused earlier

• **Maximize on-chip content:**
  Give more priority (higher chances to be allocated to D entries) to blocks not present in L1
  – Blocks in L1 can be serviced through L1-to-L1 transfer

*ACM Trans. on Architecture and Code Optimization*, 2013
Reuse and L1 Presence Tracking

- NRR (Not-Recently Reused) [AIV+13]
  - Track Reuse: NR/R (reuse bit)
  - Track L1 presence: NC/C (L1 presence vector)

- 4 states:
  - NR-C: NonReused-Cached
  - NR-NC: NonReused-NonCached
  - R-C: Reused-Cached
  - R-NC: Reused-Cached

ACM Trans. on Architecture and Code Optimization, 2013
Reuse and L1 Presence Tracking

Invalid

1st L1 miss

NR-C

Blocks in L1

L1 eviction

NR-NC

L1 miss

R-C

Blocks in L1

L1 eviction (last sharer)

R-NC

Blocks likely to be used

R-NC

L1 miss

LLC Victim Selection

++priority of eviction

NR-NC

not being used (not cached in L1)

R-NC

protect reuse

NR-C

being used (cached in L1)

R-C
Where to insert blocks? (T or D)

At this point we do not have any reuse information → Blind allocation following NRR (or any other) replacement algorithm.

If a block entering a T branch shows reuse → it will trigger a burst of memory requests.

Promote block to D entry.
When to Promote Blocks?

As soon as reuse is detected → store data copy on the 2nd L1 miss

But these blocks are in L1→ Why waste storage in 2 copies?
When to Promote Blocks?

Invalid

1\textsuperscript{st} L1 miss

NR-C-D

L1 eviction

NR-NC-D

L1 miss

R-C-D

L1 eviction (last sharer)

R-NC-D

NR-C-T

L1 eviction

NR-NC-T

L1 miss

R-NC-T

Wait till last L1 replacement

R-C-T

L1 eviction (last sharer)

L1 miss
Promotions Necessarily Trigger Demotions

Demotion: D → T

Which block to demote?

L1 eviction (last sharer)

L1 miss

L1 eviction

L1 miss

cached in L1

not cached in L1

protect reuse

++priority of demotion

R-C-D

R-C-T

NR-C-D

NR-C-T

NR-NC-D

NR-NC-T

R-NC-D

R-NC-T

Invalid

1st L1 miss

Invalid

Invalid
BDOT-FA Cache Management Policy

Victim Selection (Insertion)
-Invalid
-1st L1 miss

NR-C-D → Demotion → L1 eviction

NR-C-T → Demotion → L1 eviction

Victim Selection (Demotion)

R-C-D → Demotion → L1 eviction (last sharer)

R-C-T → Demotion → L1 eviction (last sharer)

++priority of eviction
++priority of demotion

R-C-D
R-C-T

NR-C-D
NR-C-T

NR-NC-D
NR-NC-T

R-NC-D
R-NC-T

+保护重用

Demotion
not being used
(not cached in L1)

R-NC-T

R-NC-D

Victim Selection (Insertion)

Victim Selection (Demotion)

not being used
(not cached in L1)

not being used
(not cached in L1)

being used
(cached in L1)

being used
(cached in L1)
Performance of BDOT with NRR: LLC MPKI

SPEC CPU 2006 multiprogrammed mixes

Minor improvement due to better replacement
Performance of BDOT-FA: LLC MPKI

SPEC CPU 2006 multiprogrammed mixes
Performance of BDOT-FA: System Speedup

SPEC CPU 2006 multiprogrammed mixes
Energy Consumption

SPEC CPU 2006 multiprogrammed mixes

Energy consumption includes the overhead of cache swaps

Assuming “Robust” has the same area/power as C2
BDOT-FA: Summary

• BD limits capacity and associativity of shared cache
  – Limits the associativity of the private caches as well!
  – Inclusion victims increase

• BDOT: Use the tags of the faulty entries → restore LLC associativity
  – Only improves performance when many faulty entries

• BDOT-FA: Fault-aware cache mgt. policy for BDOT
  – Smart allocation of blocks to entries T/D based on reuse and L1 presence
  – Performance speedup with respect to BD:
    • 2.1 – 13.1% (multiprogrammed)
  – Low overhead:
    • Based on BD, does not add additional storage overhead
    • Logic for cache swaps
A Fault-Aware Cache Management Policy for CMPs Operating at Ultra-Low Voltages

Alexandra Ferrerón-Labari, Jesús Alastruey-Benedé, Darío Suárez Gracia, Teresa Monreal-Arnal, Pablo Ibáñez-Marín, Víctor Viñals-Yúfera

Grupo de Arquitectura de Computadores
Departamento de Informática e Ingeniería de Sistemas
Instituto Universitario de Investigación de Ingeniería de Aragón
Universidad de Zaragoza
Back-up
BDOT-FA
Putting All Together: BDOT-FA

• Fault-aware cache mgt. policy on top of BDOT
• Insertion:
  – Blind: Rely on replacement algorithm
• Block reuse detection on T entry:
  – Trigger block promotion to D entry (cache swap)
  – Promotion implies demotion of a block from D to T entry: first blocks in L1, protect reuse
• No storage overhead:
  – Reuse bit from replacement policy
  – Presence vector from coherence protocol (no silent L1 evictions)
  – Cache swap logic
BDOT-FA: Parallel Workloads Performance

PARSEC applications

Normalized MPKI

- BD-NRU
- BDOT-NRU
- BDOT-NRR
- BDOT-NRR-FA

C6: 38.3, 113.7, 142.5, 160.7
C5: 83.7, 104.5, 75.1, 109.5
C4: 48.1, 136.9, 58.2, 155.4
C3: 31.4, 201.6, 69.4, 193.4
C2: 40.8, 104.6
BDOT-FA: Parallel Workloads Performance

PARSEC applications

Normalized Speedup

<table>
<thead>
<tr>
<th></th>
<th>BD-NRU</th>
<th>BDOT-NRU</th>
<th>BDOT-NRR</th>
<th>BDOT-NRR-FA</th>
</tr>
</thead>
<tbody>
<tr>
<td>C6</td>
<td>2.2</td>
<td>5.3</td>
<td>6.9</td>
<td>8.2</td>
</tr>
<tr>
<td>C5</td>
<td>2.9</td>
<td>6.9</td>
<td>8.2</td>
<td>10.7</td>
</tr>
<tr>
<td>C4</td>
<td>2.6</td>
<td>5.7</td>
<td>8.1</td>
<td>9.2</td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td>9.0</td>
<td>9.0</td>
</tr>
<tr>
<td>C2</td>
<td>6.4</td>
<td></td>
<td></td>
<td>6.4</td>
</tr>
</tbody>
</table>
BDOT-FA: Energy Consumption

PARSEC applications

- BD-NRU
- BDOT-NRR
- BDOT-NRU
- BDOT-NRR-FA

Normalized EPI

Symbols: Off–chip, On–chip

Values:
- C6: 18.8, 27.2, 21.6
- C5: 18.0, 27.4, 27.9
- C4: 20.2, 19.2, 20.1
- C3: 28.7, 29.6, 29.2, 19.1
- C2: 32.0, 31.2, 21.2

65.7
BDOT-FA: Parallel Workloads Performance

canneal

ferret

streamcluster

vips