EFFICIENTLY ENABLING CONVENTIONAL BLOCK SIZES FOR VERY LARGE DIE-STACKED DRAM CACHES

MICRO 2011 @ Porte Alegre, Brazil

Gabriel H. Loh [1] and Mark D. Hill [2][1]

December 2011

[1] AMD Research
[2] University of Wisconsin-Madison

Hill’s work largely performed while on sabbatical at [1].
EXECUTIVE SUMMARY

- Good use of stacked DRAM is cache, but:
  - Tags in stacked DRAM believed too slow
  - On-chip tags too large (e.g., 96 MB for 1 GB stacked DRAM cache)

- Solution put tags in stacked DRAM, but:
  - Faster Hits: Schedule together tag & data stacked DRAM accesses
  - Faster Miss: On-chip MissMap bypasses stacked DRAM on misses

- Result (e.g., 1 GB stacked DRAM cache w/ 2 MB on-chip MissMap)
  - 29-67% faster than naïve tag+data in stacked DRAM
  - Within 88-97% of stacked DRAM cache w/ impractical on-chip tags
OUTLINE

- Motivation

- Fast Hits via Compound Access Scheduling

- Fast Misses via MissMap

- Experimental Results

- Related Work and Summary
CHIP STACKING IS HERE

Motivation

• Fast Hits via Compound Access Scheduling
• Fast Misses via MissMap
• Experimental Results
• Related Work and Summary

cores
DRAM layers
cores
DRAM layers
silicon interposer

“Vertical”

“Horizontal”

256 MB

Samsung @ ISSCC’11: “A 1.2V 12.8Gb/s 2Gb Mobile Wide-I/O DRAM with 4x128 I/Os Using TSV-Based Stacking”
HOW TO USE STACKED MEMORY?

- Complete Main Memory
  - Few GB too small for all but some embedded systems

- OS-Managed NUMA Memory
  - Page-size fragmentation an issue
  - Requires OS-HW cooperation (across companies)

- Cache w/ Conventional Block (Line) Size (e.g., 64B)
  - But on-chip tags for 1 GB cache is impractical 96 MB! (TAKE 1)

- Sector/subblock Cache
  - Tag w/ 2KB block (sector) + state bits w/ each 64B subblock
  - Tags+state fits on-chip, but fragmentation issues (see paper)
**TAG+DATA IN DRAM (CONVENTIONAL BLOCKS – TAKE 2)**

- Use 2K-Stacked-DRAM pages but replace 32 64B blocks with
  - 29 tags (48b) + 29 blocks

  ![DRAM Array Diagram](image)

  - 32 x 64-byte cachelines = 2048 bytes
  - 29 ways of data

- But previously dismissed as too slow

  ![Request latency and total bank occupancy](image)
**Motivation**
- Fast Hits via Compound Access Scheduling
- Fast Misses via MissMap

**Experimental Results**

- Ideal SRAM Tags
- CAS + MissMap
- Compound Access Sched.
- Tags in DRAM

**IMPRACTICAL IDEAL & OUR RESULT FORECAST**

Compound Access Scheduling + MissMap → Approximate impractical on-chip SRAM tags

Methods Later; Avg of Web-Index, SPECjbb05, TPC-C, & SPECweb05
OUTLINE

- Motivation
- Fast Hits via Compound Access Scheduling
- Fast Misses via MissMap
- Experimental Results
- Related Work and Summary
FASTER HITS \(\text{(CONVENTIONAL BLOCKS – TAKE 3)}\)

- Motivation
- Fast Hits via Compound Access Scheduling
- Fast Misses via MissMap
- Experimental Results
- Related Work and Summary

**Motivation**

- Fast Hits via Compound Access Scheduling
- Fast Misses via MissMap

**Experimental Results**

**Related Work and Summary**
**COMPOUND ACCESS SCHEDULING**

- Reserve the bank for data access; guarantee row buffer hit
  - Approximately trading an SRAM lookup for a row-buffer hit:

![Diagram showing tags, SRAM, ACT, and RD]

- On a miss, unnecessarily holds bank open for the tag-check latency
  - Prevents tag lookup on another row in same bank
  - Effective penalty is minimal since $t_{RAS}$ must elapse before closing this row, so bank will be unavailable anyway
OUTLINE

- Motivation

- Fast Hits via Compound Access Scheduling

- Fast Misses via MissMap

- Experimental Results

- Related Work and Summary
FASTER MISSES (CONVENTIONAL BLOCKS – TAKE 4)

- Want to avoid delay & power of stacked DRAM access on miss

- Impractical on-chip tags answer
  - Q1 “Present:” Is block in stacked DRAM cache?
  - Q2 “Where:” Where in stacked DRAM cache (set/way)?

- New on-chip MissMap
  - Approximate impractical tags for practical cost
  - Answer Q1 “Present”
  - But NOT Q2 “Where”
MISSMAP

- On-chip structures to answer Q1: Is block in stacked DRAM cache?

MissMap Requirements
- Add block in miss; remove block on victimization
- No false negatives: If says, “not present” ➔ must be not present
- False positives allowed: If says, “present” ➔ may (rarely) miss

- Sounds like a Bloom Filter?

- But our implementation is precise – no false negatives or positives
  - Extreme subblocking with over-provisioning
MISSMAP IMPLEMENTATION

Key 1: Extreme Subblocking

- Fast Hits via Compound Access Scheduling
- Fast Misses via MissMap
- Experimental Results
- Related Work and Summary

MissMap entry

Tag + 16 bits tracks 1KB of memory (e.g.)

Installing a line in the DRAM $X$

Evicting a line from the DRAM $Y$

1KB memory segment

64B

tag bit vector

1KB memory segment

Tag+16 bits tracks 1KB of memory (e.g.)
MISSMAP IMPLEMENTATION

- **Key 2: Over-provisioning**
- **Key 3: Answer Q1 “Present” NOT Q2 “Where”**
  - 36b tag + 64b vector = 100b
  - NOT 36b tag + 5*64b vector = 356b (3.6x)

**Example:**
2MB MissMap
4KB pages
Each entry is ~12.5 bytes
(36b tag, 64b vector)
167,000 entries total
Best case, tracks ~640MB
OUTLINE

- Motivation

- Fast Hits via Compound Access Scheduling

- Fast Misses via MissMap

- Experimental Results

- Related Work and Summary
METHODOLOGY (SEE PAPER FOR DETAILS)

- Workloads (footprint)
  - Web-Index (2.98 GB) // SPECjbb05 (1.20 GB)
  - TPC-C (1.03 GB) // SPECweb05 (1.02 GB)

- Base Target System
  - 8 3.2 GHz cores with 1 IPC peak w/ 2-cycle 2-way 32KB I$ + D$
  - 10-cyc 8-way 2MB L2 for 2 cores + 24-cyc 16-way 8MB shared L3
  - Off-chip DRAM: DDR3-1600, 2 channels

- Enhanced Target System
  - 12-way 6MB shared L3 + 2MB MissMap
  - Stacked DRAM: 4 channels, 2x freq (~1/2 latency), 2x bus width

- gem5 simulation infrastructure (= Wisconsin GEMS + Michigan M5)
**KEY RESULT: COMPOUND SCHEDULING + MISSMAP WORK**

**Motivation**
- Fast Hits via Compound Access Scheduling
- Fast Misses via MissMap

**Experimental Results**

**Related Work and Summary**

**Compound Access Scheduling + MissMap**
Approximate impractical on-chip SRAM tags
2ND KEY RESULT: OFF-CHIP CONTENTION REDUCED

- For requests that miss, main memory is more responsive

![Graphs showing improved performance with increased cache size](image-url)

- Fewer requests → lower queuing delay
- Fewer requests → More row-buffer hits → lower DRAM latency
OTHER RESULTS IN PAPER

- Impact on all off-chip DRAM traffic (activate, read, write, precharge)
- Dynamic active memory footprint of the DRAM cache
- Additional traffic due to MissMap evictions
- Cacheline vs. MissMap lifetimes
- Sensitivity to how L3 is divided between data and the MissMap
- Sensitivity to MissMap segment size
- Performance against sub-blocked caches
OUTLINE

- Motivation

- Fast Hits via Compound Access Scheduling

- Fast Misses via MissMap

- Experimental Results

- Related Work and Summary
RELATED WORK

- Stacked DRAM as main memory
  - Mostly assumes all of main memory can be stacked
    \[ \text{Kgil} + \text{ASPLOS'06, Liu+ IEEE D&T'05, Loh ISCA'08, Woo+ HPCA'10} \]
- As a large cache
  - Mostly assumes tag-in-DRAM latency too costly
    \[ \text{Dong+ SC'10, Ghosh+ MICRO'07, Jiang+ HPCA'10,}
    \text{Loh MICRO'09, Zhao+ ICCD'07} \]
- Other stacked approaches (NVRAM, hybrid technologies, etc.)
  - \[ \text{Madan+ HPCA’09, Zhang/Li PACT’09} \]
- MissMap related
  - Subblocking \[ \text{Liptay IBMSysJ’68, Hill/Smith ISCA’84,}
    \text{Seznec ISCA’94, Rothman/Smith ICS’99} \]
  - “Density Vector” for prefetch suppression \[ \text{Lin+ ICCD’01} \]
  - Coherence optimization \[ \text{Moshovos+ HPCA’01, Cantin+ ISCA’05} \]
**EXECUTIVE SUMMARY**

- Good use of stacked DRAM is cache, but:
  - Tags in stacked DRAM believed too slow
  - On-chip tags too large (e.g., 96 MB for 1 GB stacked DRAM cache)

- Solution put tags in stacked DRAM, but:
  - **Faster Hits**: Schedule together tag & data stacked DRAM accesses
  - **Faster Miss**: On-chip MissMap bypasses stacked DRAM on misses

- Result (e.g., 1 GB stacked DRAM cache w/ 2 MB on-chip MissMap)
  - 29-67% faster than naïve tag+data in stacked DRAM
  - **Within 88-97% of stacked DRAM cache w/ impractical on-chip tags**
Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

©2011 Advanced Micro Devices, Inc. All rights reserved.
Ex. 70% of the time a 256MB cache held ~90,000 or fewer unique pages.
MISSMAP EVICTION TRAFFIC

<table>
<thead>
<tr>
<th>Workload Name</th>
<th>% Clean MissMap Evictions</th>
<th>Dirty Lines per Dirty Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>128MB</td>
</tr>
<tr>
<td>Web-Index</td>
<td>99.7%</td>
<td>0.02</td>
</tr>
<tr>
<td>SPECjbb05</td>
<td>55.7%</td>
<td>1.06</td>
</tr>
<tr>
<td>TPC-C</td>
<td>74.7%</td>
<td>3.80</td>
</tr>
<tr>
<td>SPECweb05</td>
<td>72.2%</td>
<td>1.61</td>
</tr>
</tbody>
</table>

- Many MissMap evictions correspond to clean pages (e.g., no writeback traffic from the L4)

- By the time a MissMap entry is evicted, most of its cachelines have are long past dead/evicted.
SENSITIVITY TO MISSMAP VS. DATA ALLOCATION OF L3

- 2MB MissMap + 6MB Data provides good performance
- 3MB MissMap + 5MB Data slightly better, but can hurt server workloads that are more sensitive to L3 capacity.
SENSITIVITY TO MISSMAP SEGMENT SIZE

- 4KB segment size works the best
- Our simulations make use of physical addresses, so consecutive virtual pages can be mapped to arbitrary physical pages
### COMPARISON TO SUB-BLOCKED CACHE

- Beyond 128MB, overhead is greater than MissMap
- At largest sizes (512MB, 1GB), sub-blocked cache delivers similar performance to our approach, but at substantially higher cost
BENCHMARK FOOTPRINTS

- TPC-C: ~80% of accesses served by hottest 128MB worth of pages
- SPECWeb05: ~80% accesses served by 256MB
- SPECjbb05: ~80% accesses served by 512MB
- Web-Index: huge active footprint