CACHE POLICIES AND INTERCONNECTS

Mahdi Nazm Bojnordi

Assistant Professor
School of Computing
University of Utah
Overview

- Upcoming deadline
  - Feb. 3rd: project group formation
  - Note: email me once you form a group

- This lecture
  - Cache replacement policies
  - Cache partitioning
  - Content aware optimizations
  - Cache interconnect optimizations
  - Encoding based optimizations
Recall: Cache Power Optimization

- Caches are power and performance critical components

- Performance
  - Bridging the CPU-Mem gap

- Static power
  - Large number of leaky cells

- Dynamic power
  - Access through long interconnects

Example: FX Processors

[source: AMD]
Basic Replacement Policies

- Least Recently Used (LRU)
- Least Frequently Used (LFU)
- Not Recently Used (NRU)
  
  - every block has a bit that is reset to 0 upon touch
  - a block with its bit set to 1 is evicted
  - if no block has a 1, make every bit 1

- Practical pseudo-LRU
Common Issues with Basic Policies

- Low hit rate due to cache pollution
  - streaming (no reuse)
    - A-B-C-D-E-F-G-H-I-...
  - thrashing (distant reuse)
    - A-B-C-A-B-C-A-B-C-...

- A large fraction of the cache is useless – blocks that have serviced their last hit and are on the slow walk from MRU to LRU
Basic Cache Policies

- **Insertion**
  - Where is incoming line placed in replacement list?

- **Promotion**
  - When a block is touched, it can be promoted up the priority list in one of many ways

- **Victim selection**
  - Which line to replace for incoming line? (not necessarily the tail of the list)

Simple changes to these policies can greatly improve cache performance for memory-intensive workloads
Inefficiency of Basic Policies

- About 60% of the cache blocks may be dead on arrival (DoA)

[Qureshi’07]
Adaptive Insertion Policies

- MIP: MRU insertion policy (baseline)
- LIP: LRU insertion policy

Traditional LRU places ‘i’ in MRU position.

LIP places ‘i’ in LRU position; with the first touch it becomes MRU.

[Qureshi’07]
Adaptive Insertion Policies

- LIP does not age older blocks
  - A, A, B, C, B, C, B, C, ...

- BIP: Bimodal Insertion Policy
  - Let $\varepsilon = \text{Bimodal throttle parameter}$
  - if $(\text{rand()} < \varepsilon)$
    - Insert at MRU position;
  - else
    - Insert at LRU position;

[Qureshi’07]
Adaptive Insertion Policies

- There are two types of workloads: LRU-friendly or BIP-friendly
- DIP: Dynamic Insertion Policy
  - Set Dueling

Read the paper for more details.

[Qureshi’07]
Adaptive Insertion Policies

- DIP reduces average MPKI by 21% and requires less than two bytes storage overhead

[Qureshi’07]
Re-Reference Interval Prediction

- Goal: high performing scan resistant policy
  - DIP is thrash-resistance
  - LFU is good for recurring scans
- Key idea: insert blocks near the end of the list than at the very end
- Implement with a multi-bit version of NRU
  - zero counter on touch, evict block with max counter, else increment every counter by one

Read the paper for more details.

[Jaleel’10]
A thread’s performance may be significantly reduced due to an unfair cache sharing.

Question: how to control cache sharing?

- Fair cache partitioning [Kim’04]
- Utility based cache partitioning [Qureshi’06]
Utility Based Cache Partitioning

Key idea: give more cache to the application that benefits more from cache

[Qureshi’06]
Utility Based Cache Partitioning

Three components:
- Utility Monitors (UMON) per core
- Partitioning Algorithm (PA)
- Replacement support to enforce partitions

[Qureshi’06]
Highly Associative Caches

- Last level caches have ~32 ways in multicores
  - Increased energy, latency, and area overheads

![Graph showing increase over 4-way (%) for Area, Hit Latency, Hit Energy for 16-way and 32-way caches.]

![Graph showing IPC improvement vs 4-way (%) for different benchmarks: rand0, ammp_m.]

[Sanchez’10]
Recall: Victim Caches

- Goal: to decrease conflict misses using a small FA cache

Can we reduce the hardware overheads?
The ZCache

- **Goal:** design a highly associative cache with a low number of ways
- Improves associativity by increasing number of replacement candidates
- Retains low energy/hit, latency and area of caches with few ways
- Skewed associative cache: each way has a different indexing function (in essence, \( W \) direct-mapped caches)

[Sanchez’10]
The ZCache

- When block A is brought in, it could replace one of four (say) blocks B, C, D, E; but B could be made to reside in one of three other locations (currently occupied by F, G, H); and F could be moved to one of three other locations.

Read the paper for more details.

[Sanchez’10]
Content Aware Optimizations
Dynamic Zero Compression

More than 70% of the bits in data cache accesses are 0s

Example of a small cache

<table>
<thead>
<tr>
<th></th>
<th>Read (pJ)</th>
<th>Read (%)</th>
<th>Write (pJ)</th>
<th>Write (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total</strong></td>
<td>44.4</td>
<td>100.0</td>
<td>99.1</td>
<td>100.0</td>
</tr>
<tr>
<td><strong>Decoder</strong></td>
<td>5.5</td>
<td>12.4</td>
<td>5.5</td>
<td>5.5</td>
</tr>
<tr>
<td><strong>word lines</strong></td>
<td>1.1</td>
<td>2.5</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td><strong>Tag bitlines</strong></td>
<td>3.0</td>
<td>6.2</td>
<td>3.0</td>
<td>3.0</td>
</tr>
<tr>
<td><strong>and sense-amp</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data bitlines</strong></td>
<td>14.5</td>
<td>32.7</td>
<td>69.2</td>
<td>69.9</td>
</tr>
<tr>
<td><strong>and sense-amp</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>I/O buses</strong></td>
<td>12.1</td>
<td>27.3</td>
<td>12.1</td>
<td>12.2</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td>8.4</td>
<td>18.9</td>
<td>8.4</td>
<td>8.5</td>
</tr>
</tbody>
</table>

[Villa’00]
Dynamic Zero Compression

- Zero Indicator Bit; one bit per grouping of bits; set if bits are zeros; controls wordline gating

Address-controlled

Data-Controlled

[Villa’00]
Dynamic Zero Compression

Data cache bitline swing reduction

[Villa’00]
Dynamic Zero Compression

- Data cache energy savings

![Bar chart showing energy savings for various applications and benchmarks.](chart.png)

[Villa’00]
Cache Interconnect Optimizations
Fewer subarrays gives increased area efficiency, but larger delay due to longer wordlines/bitlines

[Aniruddha’09]
H-tree is clearly the dominant component of energy consumption.
Heterogeneous Interconnects

- A global wire management at the microarchitecture level
- A heterogeneous interconnect that is comprised of wires with varying latency, bandwidth, and energy characteristics

[Balasubramonian’05]
Better energy-efficiency for a dynamically scheduled partitioned architecture
  - ED² is reduced by 11%

A low-latency low-bandwidth network can be effectively used to hide wire latencies and improve performance

A high-bandwidth low-energy network and an instruction assignment heuristic are effective at reducing contention cycles and total processor energy.

[Balasubramonian’05]
Non-Uniform Cache Architecture

- NUCA optimizes energy and time based on the proximity of the cache blocks to the cache controller.

2MB @ 130nm
Bank Access time = 3 cycles
Interconnect delay = 8 cycles

16MB @ 50nm
Bank Access time = 3 cycles
Interconnect delay = 44 cycles

[Kim’04]
Non-Uniform Cache Architecture

**S-NUCA-1**
- Use private per-bank channel
- Each bank has its distinct access latency
- Statically decide data location for its given address
- Average access latency = 34.2 cycles
- Wire overhead = 20.9% → an issue

[Kim’04]
S-NUCA-2

- Use a 2D switched network to alleviate wire area overhead
- Average access latency = 24.2 cycles
- Wire overhead = 5.9%
Non-Uniform Cache Architecture

- **Dynamic NUCA**
  - Data can dynamically migrate
  - Move frequently used cache lines closer to CPU

KIM’04
Non-Uniform Cache Architecture

- Fair mapping
  - Average access time across all bank sets are equal

![Diagram of cache architecture]
Non-Uniform Cache Architecture

- Shared mapping
  - Sharing the closest banks for farther banks

![Diagram showing 8 bank sets and ways 0 to 3 with arrows indicating mapping]
Encoding Based Optimizations
Bus invert coding transfers either the data or its complement to minimize the number of bit flips on the bus.

\[ P_{\text{switching}} = \alpha C V_{DD}^2 f \]
Time-Based Data Transfer

- The percentage of processor energy expended on an 8MB cache when running a set of parallel applications on a Sun Niagara-like multicore processor

![Relative CPU Energy](chart)

[Bojnordi’13]
Communication over the long, capacitive H-tree interconnect is the dominant source of energy consumption (80% on average) in the L2 cache.
Key idea: represent information by the number of clock cycles between two consecutive pulses to reduce interconnect activity factor.

Time-Based Data Transfer

Example: transmitting the value 5

- **Time Based Data Transfer**
- **Parallel Data Transfer**
- **Serial Data Transfer**

[Bojnordi'13]
Cache blocks are partitioned into small, contiguous chunks.

[Bojnordi'13]
Time-Based Data Transfer

a) Transmitter

- Counter: 4 0 1 2 0 1 2 3 4 5 6 7 0 1 2
- Reset Strobe
- Data Strobe

3 cycles
2 cycles

Wire Delay

b) Receiver

- Counter: 5 6 7 0 1 2 3 4 5 6 0 1 2 0 1
- Reset Strobe
- Data Strobe

3 cycles
2 cycles

[Bojnordi’13]
L2 cache energy is reduced by 1.8x at the cost of less than 2% increase in the execution time.

[Bojnordi’13]