ECE 4750 Computer Architecture
Topic 6: Cache Microarchitecture

Christopher Batten
School of Electrical and Computer Engineering
Cornell University

http://www.csl.cornell.edu/courses/ece4750
slide revision: 2013-10-01-22-23
Agenda

Single-Bank Cache Microarchitecture

Multi-Bank Cache Microarchitecture

Basic Optimizations

Cache Examples
Direct Mapped Cache Microarchitecture
Set Associative Cache Microarchitecture

Tag Check

Data Access

Tag Check

Data Access

addr

tag

idx

off

27b

1b

2b

Way 0

Set 0

Set 1

v

tag

Way 1

Set 0

Set 1

word enable

1b

ram enable

hit?

rd/wr

wdata

Tag Check

Data Access

rdata

Tag Check

Data Access
Fully Associative Cache Microarchitecture
Synchronous SRAMs

This assumes combinational SRAMS. What if SRAMs are synchronous? Pipeline all cache accesses?
Direct-Mapped Parallel Read Hit Path

**Diagram Description:**
- **Addr:** The address input (addr) is processed to extract the tag, index, and offset.
- **Tag Check:** The tag is checked against the cache tags.
- **Read Access:** If the tag matches, the corresponding data is accessed.
- **Direct-Mapped:** Each set (Set 0 to Set 3) contains one block that is directly mapped.
- **Parallel Read:** Multiple requests can be processed simultaneously.
- **Hit Decision:** If all data is found, a hit is signaled (hit?).

**Key Elements:**
- **Tag:** The tag is compared with the cache tags.
- **Index:** The index is used to select the appropriate set.
- **Offset:** The offset is used to retrieve the correct data within the block.
- **Word Enable:** Control signal for access.
- **Rdata:** The read data output.
Direct Mapped Pipelined Write Hit Path
Set-Associative Parallel Read Hit Path

- Tag Check
- Read Access

Way 0
Way 1
Processor-Cache Interaction

Single-Bank Cache uArch

Multi-Bank Cache uArch

Basic Optimizations

Cache Examples

ECE 4750

T06: Cache Microarchitecture

10 / 36
Zero-Cycle Hit Latency with Tightly Coupled Interface

Fetch (F)  Decode & Reg Read (D)  Execute (X)  Memory (M)  Writeback (W)
Two-Cycle Hit Latency with Val/Rdy Interface
## Agenda

- Single-Bank Cache Microarchitecture
- Multi-Bank Cache Microarchitecture
- Basic Optimizations
- Cache Examples
Multicore PARC w/ Shared Unified I/D $
Multicore PARC w/ Shared Single-Bank I$ and D$
Multicore PARC w/ Shared Multi-Bank I$ and D$
Multicore PARC w/ Private I$ and D$
Multicore PARC w/ Private I$ and Shared D$
Agenda

- Single-Bank Cache Microarchitecture
- Multi-Bank Cache Microarchitecture
- Basic Optimizations
- Cache Examples
Reduce Average Memory Access Time

Avg Mem Access Time = Hit Time + (Miss Rate × Miss Penalty)

- Reduce hit time
  - Small and simple caches

- Reduce miss rate
  - Large block size
  - Large cache size
  - High associativity
  - Compiler optimizations

- Reduce miss penalty
  - Multi-level cache hierarchy
  - Prioritize reads
Reduce Hit Time: Small & Simple Caches
Reduce Miss Rate: Large Block Size

- Less tag overhead
- Exploit fast burst transfers from DRAM
- Exploit fast burst transfers over wide on-chip busses
- Can waste bandwidth if data is not used
- Fewer blocks → more conflicts
Reduce Miss Rate: Large Cache Size

Empirical Rule of Thumb:
If cache size is doubled, miss rate usually drops by about $\sqrt{2}$
Reduce Miss Rate: High Associativity

Empirical Rule of Thumb:
Direct-mapped cache of size N has about the same miss rate as a two-way set-associative cache of size N/2
Reduce Miss Rate: Compiler Optimizations

- Restructuring code affects the data block access sequence
  - Group data accesses together to improve spatial locality
  - Re-order data accesses to improve temporal locality

- Prevent data from entering the cache
  - Useful for variables that will only be accessed once before eviction
  - Needs mechanism for software to tell hardware not to cache data
    (“no-allocate” instruction hits or page table bits)

- Kill data that will never be used again
  - Streaming data exploits spatial locality but not temporal locality
  - Replace into dead-cache locations
Loop Interchange

```
for(j=0; j < N; j++) {
    for(i=0; i < M; i++) {
        x[i][j] = 2 * x[i][j];
    }
}
```

```
for(i=0; i < M; i++) {
    for(j=0; j < N; j++) {
        x[i][j] = 2 * x[i][j];
    }
}
```

What type of locality does this improve?
Loop Fusion

for(i=0; i < N; i++)
    a[i] = b[i] * c[i];

for(i=0; i < N; i++)
    d[i] = a[i] * c[i];

for(i=0; i < N; i++)
    a[i] = b[i] * c[i];
    d[i] = a[i] * c[i];

What type of locality does this improve?
Reduce Miss Penalty: Multi-Level Caches

```
Avg Mem Access Time =
Hit Time of L1 + ( Miss Rate of L1 × Miss Penalty of L1 )

Miss Penalty of L1 =
Hit Time of L2 + ( Miss Rate of L2 × Miss Penalty of L2 )
```

- Local miss rate = misses in cache / accesses to cache
- Global miss rate = misses in cache / processor memory accesses
- Misses per instruction = misses in cache / number of instructions

Local miss rate = misses in cache / accesses to cache
Global miss rate = misses in cache / processor memory accesses
Misses per instruction = misses in cache / number of instructions
Reduce Miss Penalty: Multi-Level Caches

- Use smaller L1 if there is also an L2
  - Trade increased L1 miss rate for reduced L1 hit time & L1 miss penalty
  - Reduces average access energy

- Use simpler write-through L1 with on-chip L2
  - Write-back L2 cache absorbs write traffic, doesn’t go off-chip
  - Simplifies processor pipeline
  - Simplifies on-chip coherence issues

- Inclusive Multilevel Cache
  - Inner cache holds copy of data in outer cache
  - External coherence is simpler

- Exclusive Multilevel Cache
  - Inner cache hold data not in outer cache
  - Swap lines between inner/outer cache on miss
Reduce Miss Penalty: Prioritize Reads

- Processor not stalled on writes, and read misses can go ahead of writes to main memory
- Write buffer may hold updated value of location needed by read miss
  - On read miss, wait for write buffer to be empty
  - Check write buffer addresses and bypass

Evicted dirty lines for writeback cache
OR
All writes in writethrough cache
### Cache Optimizations Impact on Average Memory Access Time

<table>
<thead>
<tr>
<th>Technique</th>
<th>Hit Time</th>
<th>Miss Rate</th>
<th>Miss Penalty</th>
<th>HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel read hit</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
</tr>
<tr>
<td>Pipelined write hit</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1</td>
</tr>
<tr>
<td>Smaller caches</td>
<td>— ++</td>
<td>—</td>
<td>—</td>
<td>0</td>
</tr>
<tr>
<td>Large block size</td>
<td>—</td>
<td>—</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>Large cache size</td>
<td>++</td>
<td>—</td>
<td>—</td>
<td>1</td>
</tr>
<tr>
<td>High associativity</td>
<td>++</td>
<td>—</td>
<td>—</td>
<td>1</td>
</tr>
<tr>
<td>Compiler optimizations</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>0</td>
</tr>
<tr>
<td>Multi-level cache</td>
<td>—</td>
<td>—</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Prioritize reads</td>
<td>—</td>
<td>1</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>
Agenda

Single-Bank Cache Microarchitecture

Multi-Bank Cache Microarchitecture

Basic Optimizations

Cache Examples
Intel Itanium-2 On-Chip Caches

Level 1: 16KB, 4-way s.a., 64B line, quad-port (2 load+2 store), single cycle latency

Level 2: 256KB, 4-way s.a., 128B line, quad-port (4 load or 4 store), five cycle latency

Level 3: 3MB, 12-way s.a., 128B line, single 32B port, twelve cycle latency
IBM Power-7 On-Chip Caches

32KB L1 I$/core
32KB L1 D$/core
3-cycle latency

256KB Unified L2$/core
8-cycle latency

32MB Unified Shared L3$
Embedded DRAM
25-cycle latency to local slice

February 9, 2010 CS152, Spring 2010
Power 7 On-Chip Caches [IBM 2009]
25
32KB L1 I$/core
32KB L1 D$/core...
256KB Unified L2$/core
8-cycle latency
32MB Unified Shared L3$
Embedded DRAM
25-cycle latency to local slice

ECE 4750 T06: Cache Microarchitecture 35 / 36
Acknowledgements

Some of these slides contain material developed and copyrighted by:

Arvind (MIT), Krste Asanović (MIT/UCB), Joel Emer (Intel/MIT)
James Hoe (CMU), John Kubiatowicz (UCB), David Patterson (UCB)

MIT material derived from course 6.823
UCB material derived from courses CS152 and CS252