MEMORY HIERARCHY BASICS

B649
Parallel Architectures and Programming
BASICS
Why Do We Need Caches?
Overview

![Diagram showing CPU, Cache, Memory, I/O bus, I/O devices with details on register reference, cache reference, memory reference, disk memory reference, size and speed.](image-url)
<table>
<thead>
<tr>
<th>Cache Terms</th>
<th>Associative Terms</th>
<th>Allocation Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache</td>
<td>fully associative</td>
<td>write allocate</td>
</tr>
<tr>
<td>virtual memory</td>
<td>dirty bit</td>
<td>unified cache</td>
</tr>
<tr>
<td>memory stall cycles</td>
<td>block offset</td>
<td>misses per instruction</td>
</tr>
<tr>
<td>direct mapped</td>
<td>write back</td>
<td>block</td>
</tr>
<tr>
<td>valid bit</td>
<td>data cache</td>
<td>locality</td>
</tr>
<tr>
<td>block address</td>
<td>hit time</td>
<td>address trace</td>
</tr>
<tr>
<td>write through</td>
<td>cache miss</td>
<td>set</td>
</tr>
<tr>
<td>instruction cache</td>
<td>page fault</td>
<td>random replacement</td>
</tr>
<tr>
<td>average memory access time</td>
<td>miss rate</td>
<td>index field</td>
</tr>
<tr>
<td>cache hit</td>
<td>n-way set associative</td>
<td>no-write allocate</td>
</tr>
<tr>
<td>page</td>
<td>least-recently used</td>
<td>write buffer</td>
</tr>
<tr>
<td>miss penalty</td>
<td>tag field</td>
<td>write stall</td>
</tr>
</tbody>
</table>
Terminology

cache
virtual memory
memory stall cycles
direct mapped
valid bit
block address
write through
instruction cache
average memory access time
cache hit
page
miss penalty

fully associative
dirty bit
block offset
write back
data cache
hit time
cache miss
page fault
miss rate
n-way set associative
least-recently used
tag field

write allocate
unified cache
misses per instruction
block
locality
address trace
set
random replacement
index field
no-write allocate
write buffer
write stall
Four Memory-Hierarchy Questions

• Where can a block be placed in the upper level?
  ★ block placement

• How is a block found if it is in the upper level?
  ★ block identification

• Which block should be replaced on a miss?
  ★ block replacement

• What happens on a write?
  ★ write strategy
Where Can a Block Be Placed in a Cache?

- Only one place for each block
  - direct mapped
    \[(\text{Block address}) \mod (\text{Number of blocks in cache})\]

- Anywhere in the cache
  - fully associative

- Restricted set of places
  - set associative
    \[(\text{Block address}) \mod (\text{Number of sets in cache})\]
How is a Block Found if it is in Cache?

• “Tags” in each cache block gives the block address
  ★ all possible tags searched in parallel (associative memory)
  ★ valid bit tells whether a tag match valid

Fields in a memory address

<table>
<thead>
<tr>
<th>Block address</th>
<th>Tag</th>
<th>Index</th>
<th>Block offset</th>
</tr>
</thead>
</table>

• No “index” field in fully associative caches
Which Block Should be Replaced on a Miss?

- Random
  - easy to implement

- Least-recently used (LRU)
  - idea: rely on the past to predict the future
  - replace the block unused for the longest time

- First in, First out (FIFO)
  - approximates LRU \((\text{oldest}, \text{rather than least recently used})\)
  - simpler to implement
What Happens on a Write?

• Write strategy
  ★ write through
    ★ write to cache block and to the block in the lower-level memory
  ★ write back
    ★ write only to cache block, update the lower-level memory when block replaced

• Block allocation strategy
  ★ write allocate
    ★ allocate a block on cache miss
  ★ no-write allocate
    ★ do not allocate, no affect on cache
CACHE PERFORMANCE
Defining Performance

• Miss rate is attractive, but misleading

• Bottom line: CPU time
  ★ assume in-order execution for now

★ include hit clock cycles in memory cycles or execution cycles?
Defining Performance

• Miss rate is attractive, but misleading

\[
\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}
\]

• Bottom line: CPU time
  ★ assume in-order execution for now

★ include hit clock cycles in memory cycles or execution cycles?
Defining Performance

• Miss rate is attractive, but misleading

\[
\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}
\]

• Bottom line: CPU time
  ✴ assume in-order execution for now

\[
\text{CPU time} = (\text{CPU execution clock cycles} + \text{Memory stall clock cycles}) \times \text{Clock cycle time}
\]
  ✴ include hit clock cycles in memory cycles or execution cycles?
Example

• Assumptions
  ★ in-order execution
  ★ miss penalty = 200 cycles
  ★ cycles per instruction (w/o cache misses) = 1 cycle
  ★ average miss rate = 2%
  ★ average memory references per instruction = 1.5
  ★ cache misses per 1000 instructions = 30

• What is the impact of cache?
Impact of Cache

Assumptions
* in-order execution
* miss penalty = 200 cycles
* cycles per instruction (w/o cache misses) = 1 cycle
* average miss rate = 2%
* average memory references per instruction = 1.5
* cache misses per 1000 instructions = 30

CPU time = \( IC \times (CPI + \text{Memory stall cycle / instruction}) \times \text{Clock cycle time} \)

With cache  = \( IC \times (1.0 + (30/1000 \times 200)) \times \text{Clock cycle time} \)
            = \( IC \times 7.0 \times \text{Clock cycle time} \)

Without cache  = \( IC \times (1.0 + 200 \times 1.5) \times \text{Clock cycle time} \)
                = \( IC \times 301 \times \text{Clock cycle time} \)
Miss Penalty and Out-of-order Execution

• Some of the miss penalty is hidden

• Miss latency
  ★ What is start and end of a memory operation (memory latency)?
  ★ What is the start of overlap with the processor (latency overlap)?
Miss Penalty and Out-of-order Execution

• Some of the miss penalty is hidden

Memory stall cycle / Instruction = (Misses / Instruction) × (Total miss latency – Overlapped miss latency)

• Miss latency
  ★ What is start and end of a memory operation (memory latency)?
  ★ What is the start of overlap with the processor (latency overlap)?
VIRTUAL MEMORY

... a system has been devised to make the core drum combination appear to the programmer as a single level store, the requisite transfers taking place automatically.

Killburn et al. [1962]
Virtual Memory

Virtual address

0
4K
8K
12K

Virtual memory

Physical address

0
4K
8K
12K
16K
20K
24K
28K

Physical main memory

Disk
# Virtual Memory vs Cache

<table>
<thead>
<tr>
<th>Parameter</th>
<th>First-level cache</th>
<th>Virtual memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block (page) size</td>
<td>16–128 bytes</td>
<td>4096–65,536 bytes</td>
</tr>
<tr>
<td>Hit time</td>
<td>1–3 clock cycles</td>
<td>100–200 clock cycles</td>
</tr>
<tr>
<td>Miss penalty</td>
<td>8–200 clock cycles</td>
<td>1,000,000–10,000,000 clock cycles</td>
</tr>
<tr>
<td>(access time)</td>
<td>(6–160 clock cycles)</td>
<td>(800,000–8,000,000 clock cycles)</td>
</tr>
<tr>
<td>(transfer time)</td>
<td>(2–40 clock cycles)</td>
<td>(200,000–2,000,000 clock cycles)</td>
</tr>
<tr>
<td>Miss rate</td>
<td>0.1–10%</td>
<td>0.00001–0.001%</td>
</tr>
<tr>
<td>Address mapping</td>
<td>25–45 bit physical address to 14–20 bit cache address</td>
<td>32–64 bit virtual address to 25–45 bit physical address</td>
</tr>
</tbody>
</table>
Why Virtual Memory?

- Ability to large programs with large data
  - earlier, users would need to use overlays manually

- Ability to share a machine among multiple processes, with protection
  - very early Windows systems did not have this protection!

- Ease of relocation
  - alternatively, would need to use a special relocation register, or do this is software
Virtual Memory: Terminology

- Cache block ⇒ page or segment
- Cache miss ⇒ page fault or address fault
- Address translation or memory mapping
  - virtual address to physical address
- Differences between VM and caches
  - replacement controlled by software in VM
  - processor address determines the size of VM
  - secondary storage shared with file system
- Two flavors of VM: paged and segmented
Segmented vs Paged

<table>
<thead>
<tr>
<th></th>
<th>Code</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paging</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Segmentation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2007 Elsevier, Inc. All rights reserved.
Four Memory Hierarchy Questions

• Where can a block be placed in memory?
  ★ fully associative
Four Memory Hierarchy Questions

• Where can a block be placed in memory?
  ★ fully associative

• How is a block found if it is in main memory?
  ★ using a page table
Four Memory Hierarchy Questions

• Where can a block be placed in memory?
  ★ fully associative

• How is a block found if it is in main memory?
  ★ using a page table

• Which block should be replaced on a virtual memory miss?
  ★ LRU policy, implemented with a *use bit* or *reference bit*
Four Memory Hierarchy Questions

- Where can a block be placed in memory?
  - fully associative

- How is a block found if it is in main memory?
  - using a page table

- Which block should be replaced on a virtual memory miss?
  - LRU policy, implemented with a use bit or reference bit

- What happens on a write?
  - always write back
- Virtualized page tables
- Inverted page tables
  - hash tables, as many entries as the number of allocated pages
- Multilevel page tables
Multilevel Page Tables

Virtual address translation on Opteron
Translation Look-aside Buffers (TLBs)

TLB uses fully associative placement
Address Translation with Caches

Virtual address <64>

Virtual page number <51>

Page offset <13>

TLB tag compare address <43>

TLB index <8>

L1 cache index <7>

Block offset <6>

To CPU

TLB tag <43>

TLB data <28>

L1 tag compare address <28>

=?

Physical address <41>

L2 tag compare address <19>

L2 cache index <16>

Block offset <6>

To CPU

L2 cache tag <19>

L2 data <512>

To L1 cache or CPU

L1 cache tag <43>

L1 data <512>
SIX BASIC CACHE OPTIMIZATIONS
Six Ideas

- Reducing the miss rate
  - larger block size
  - larger cache size
  - higher associativity
- Reducing the miss penalty
  - multilevel caches
  - prioritize read misses over writes
- Reducing the time to hit in cache
  - avoid address translation when indexing the cache

Average memory access time = Hit time + Miss rate × Miss penalty
Type of Cache Misses

• Compulsory
  ★ caused by the very first access to the block

• Capacity
  ★ if the cache cannot contain all the needed blocks, misses (in addition to the compulsory misses) due to blocks being discarded and retrieved later

• Conflict
  ★ if the block placement is not fully associative, misses (in addition to the above two kinds) due to blocks being discarded and later retrieved due to too many blocks being mapped to the same set

• Coherency
Classical Approach: Reduce Miss Rate

SPEC2000 benchmarks
Optimizations

• Larger block size to reduce miss rate
Increasing Block Size

![Graph showing the relationship between block size and miss rate for different cache sizes (4K, 16K, 64K, 256K). The x-axis represents block size ranging from 16 to 256, and the y-axis represents miss rate ranging from 0% to 10%. The graph indicates that larger block sizes generally lead to lower miss rates.](image)
Example

• Memory access:
  ★ latency = 80 clock cycles
  ★ bandwidth: 16 bytes each 2 cycles, thereafter
  ★ thus, 16 bytes in 82 cycles, 32 bytes in 84 cycles, ...
Example

- Memory access:
  - latency = 80 clock cycles
  - bandwidth: 16 bytes each 2 cycles, thereafter
  - thus, 16 bytes in 82 cycles, 32 bytes in 84 cycles, ...

Average memory access time = Hit time + Miss rate × Miss penalty
For 16-byte block, 4K size cache = 1 + (8.57% × 82) = 8.027 clock cycles
For 256-byte block, 256K size cache = 1 + (0.49% × 112) = 1.549 clock cycles
## Average Memory Access Times

<table>
<thead>
<tr>
<th>Block size</th>
<th>Miss penalty</th>
<th>Cache size</th>
<th>4K</th>
<th>16K</th>
<th>64K</th>
<th>256K</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>82</td>
<td>8.027</td>
<td>4.231</td>
<td>2.673</td>
<td>1.894</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>84</td>
<td><strong>7.082</strong></td>
<td>3.411</td>
<td>2.134</td>
<td>1.588</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>88</td>
<td>7.160</td>
<td><strong>3.323</strong></td>
<td><strong>1.933</strong></td>
<td><strong>1.449</strong></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>96</td>
<td>8.469</td>
<td>3.659</td>
<td>1.979</td>
<td>1.470</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>112</td>
<td>11.651</td>
<td>4.685</td>
<td>2.288</td>
<td>1.549</td>
<td></td>
</tr>
</tbody>
</table>
Optimizations

- Larger block size to reduce miss rate
- Larger caches to reduce miss rate
Optimizations

• Larger block size to reduce miss rate
• Larger caches to reduce miss rate
• Higher associativity to reduce miss rate
Increasing Associativity

Lessons:

1. 8-way is as good as fully associative

2. 2:1 cache rule of thumb: direct mapped cache of size N has about the same miss rate as 2-way of size N/2
Example

Clock cycle time$_2$ = 1.36 × Clock cycle time$_1$
Clock cycle time$_4$ = 1.44 × Clock cycle time$_2$
Clock cycle time$_8$ = 1.52 × Clock cycle time$_4$
Hit time = 1 clock cycle
Miss penalty to L2 = 25 cycles
Example

Clock cycle time\(_2\) = 1.36 \times \) Clock cycle time\(_1\)
Clock cycle time\(_4\) = 1.44 \times \) Clock cycle time\(_2\)
Clock cycle time\(_8\) = 1.52 \times \) Clock cycle time\(_4\)
Hit time = 1 clock cycle
Miss penalty to L2 = 25 cycles

Average memory access time\(_8\) = \text{Hit time}_8 + \text{Miss rate}_8 \times \text{Miss penalty}_8
\hspace{1cm} = 1.52 + \text{Miss rate}_8 \times 25
Average memory access time\(_4\) = 1.44 + \text{Miss rate}_4 \times 25
Average memory access time\(_2\) = 1.36 + \text{Miss rate}_2 \times 25
Average memory access time\(_1\) = 1.00 + \text{Miss rate}_1 \times 25
## Average Memory Access Times

<table>
<thead>
<tr>
<th>Cache size (KB)</th>
<th>One-way</th>
<th>Two-way</th>
<th>Four-way</th>
<th>Eight-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3.44</td>
<td>3.25</td>
<td>3.22</td>
<td>3.28</td>
</tr>
<tr>
<td>8</td>
<td>2.69</td>
<td>2.58</td>
<td>2.55</td>
<td>2.62</td>
</tr>
<tr>
<td>16</td>
<td>2.23</td>
<td>2.40</td>
<td>2.46</td>
<td>2.53</td>
</tr>
<tr>
<td>32</td>
<td>2.06</td>
<td>2.30</td>
<td>2.37</td>
<td>2.45</td>
</tr>
<tr>
<td>64</td>
<td>1.92</td>
<td>2.14</td>
<td>2.18</td>
<td>2.25</td>
</tr>
<tr>
<td>128</td>
<td>1.52</td>
<td>1.84</td>
<td>1.92</td>
<td>2.00</td>
</tr>
<tr>
<td>256</td>
<td>1.32</td>
<td>1.66</td>
<td>1.74</td>
<td>1.82</td>
</tr>
<tr>
<td>512</td>
<td>1.20</td>
<td>1.55</td>
<td>1.59</td>
<td>1.66</td>
</tr>
</tbody>
</table>
Optimizations

- Larger block size to reduce miss rate
- Larger caches to reduce miss rate
- Higher associativity to reduce miss rate
- Multilevel caches to reduce miss penalty
Computing Memory Access Time

Average memory access time = Hit time₁ + Miss rate₁ × Miss penalty₁

Miss penalty₁ = Hit time₂ + Miss rate₂ × Miss penalty₂

Average memory access time = Hit time₁ + Miss rate₁ × (Hit time₂ + Miss rate₂ × Miss penalty₂)
Computing Memory Access Time

Average memory access time = Hit time₁ + Miss rate₁ × Miss penalty₁

Miss penalty₁ = Hit time₂ + Miss rate₂ × Miss penalty₂

Average memory access time = Hit time₁ + Miss rate₁ × (Hit time₂ + Miss rate₂ × Miss penalty₂)

• Note: L2 misses are on leftovers from L1
Two Types of Miss Rates

• Local Miss Rate
  ★ number of misses / total number of memory accesses to this cache

• Global Miss Rate
  ★ number of misses / total number of memory accesses generated by the processor

• Local miss rate is large for second-level cache
  ★ Why?
Two Types of Miss Rates

• Local Miss Rate
  ★ number of misses / total number of memory accesses to this cache

• Global Miss Rate
  ★ number of misses / total number of memory accesses generated by the processor

• Local miss rate is large for second-level cache
  ★ Why?

Average memory stalls per instruction =
L1 misses per instruction × L2 hit time +
L2 misses per instruction × L2 miss penalty
Miss Rates vs Cache Sizes

- **Local miss rate**
- **Global miss rate**
- **Single cache miss rate**

The graph shows the relationship between miss rates and cache sizes. As cache size increases, the miss rate decreases. The local miss rate shows a significant drop at small cache sizes, while the global and single cache miss rates exhibit a more gradual decrease. The data points are marked at cache sizes of 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1024 KB, 2048 KB, and 4096 KB, with corresponding miss rates of 99%, 99%, 98%, 96%, 88%, 67%, 55%, 51%, 46%, 39%, and 34%, respectively.
Relative Execution Time by L2 Size

Second-level cache size (KB)

Relative execution time

L2 hit = 8 clock cycles
L2 hit = 16 clock cycles

© 2007 Elsevier, Inc. All rights reserved.
Questions to Consider

• Size of L2 cache
• Set-associativity
• Multi-level inclusion
  ★ different block sizes cause problems
  ★ what if size of L2 is only slightly bigger than L1?
  ★ AMD Opteron follows exclusion property
    ★ swap L1 and L2 blocks on L1 miss
Optimizations

- Larger block size to reduce miss rate
- Larger caches to reduce miss rate
- Higher associativity to reduce miss rate
- Multilevel caches to reduce miss penalty
- Prioritizing read misses over writes to reduce miss penalty
Example

```
SW R3, 512(R0)    ; M[512] ← R3 (cache index 0)
LW R1, 1024(R0)   ; R1 ← M[1024] (cache index 0)
LW R2, 512(R0)    ; R2 ← M[512] (cache index 0)
```
Example

- RAW data hazard
- Write through cache
  - write buffer can cause a problem
  - either let write finish or check the buffer and let read miss proceed (preferred)
- Write back cache
  - either let writes of replaced dirty blocks to finish, or buffer writes and let read misses proceed after checking the buffer (preferred)

```
SW R3, 512(R0) ; M[512] ← R3 (cache index 0)
LW R1, 1024(R0) ; R1 ← M[1024] (cache index 0)
LW R2, 512(R0) ; R2 ← M[512] (cache index 0)
```
Optimizations

• Larger block size to reduce miss rate
• Larger caches to reduce miss rate
• Higher associativity to reduce miss rate
• Multilevel caches to reduce miss penalty
• Prioritizing read misses over writes to reduce miss penalty
• Avoiding address translation during indexing of the cache to reduce hit time
Virtual Caches
Virtual Caches

- Use virtual address in caches to avoid translation
Virtual Caches

• Use virtual address in caches to avoid translation

• Reasons against:
  ★ Page-level protection (checked at address translation time)
  ★ Process switches might force flushes; or expand the cache address tag with process-identifier (PID) tag
Miss Rate of Virtually Addressed Caches

- Cache size: 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, 512K, 1024K
- Miss rates: 0.6%, 1.1%, 1.8%, 2.7%, 3.4%, 3.9%, 4.1%, 4.3%, 4.3%, 4.3%
- Types: Purge, PIDs, Uniprocess
Virtual Caches

• Use virtual address in caches to avoid translation

• Reasons against:
  ★ Page-level protection (checked at address translation time)
  ★ Process switches might force flushes; or expand the cache address tag with process-identifier (PID) tag
  ★ Multiple virtual addresses might map to same physical address (aliases or synonyms)
    * need special hardware to ensure unique copy (e.g., Opteron has 64KB instruction cache, 2-way set associative, 4KB page size)
    * page coloring simplifies the hardware (e.g., Sun OS required all aliases to be identical in last 18 bits)
  ★ I/O often uses physical addresses
Best of Both Worlds

• Idea: Use part of page offset to index the cache
  ★ recall that page offset is identical in both virtual and physical addresses

• Virtually indexed, physically tagged

• Limitation: direct-mapped cache size may not be bigger than page size

★ increase associativity to get larger caches (e.g., IBM 3033 has 16-way set-associative cache to get over the limit of 4KB page size)
6 Optimizations: Recap

- Larger block size to reduce miss rate
- Larger caches to reduce miss rate
- Higher associativity to reduce miss rate
- Multilevel caches to reduce miss penalty
- Prioritizing read misses over writes to reduce miss penalty
- Avoiding address translation during indexing of the cache to reduce hit time
Watch Out for OS Performance

<table>
<thead>
<tr>
<th>Workload</th>
<th>% in applications</th>
<th>% in OS</th>
<th>% time due to application misses</th>
<th>Time</th>
<th>% time due directly to OS misses</th>
<th>% time OS misses and application conflicts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Misses</td>
<td></td>
<td>Inherent application misses</td>
<td>OS conflicts with applications</td>
</tr>
<tr>
<td>Pmake</td>
<td>47%</td>
<td>53%</td>
<td>14.1%</td>
<td></td>
<td></td>
<td>4.8%</td>
</tr>
<tr>
<td>Multipgm</td>
<td>53%</td>
<td>47%</td>
<td>21.6%</td>
<td></td>
<td></td>
<td>3.4%</td>
</tr>
<tr>
<td>Oracle</td>
<td>73%</td>
<td>27%</td>
<td>25.7%</td>
<td></td>
<td></td>
<td>10.2%</td>
</tr>
</tbody>
</table>
NEXT: MORE ON MEMORY HIERARCHY