Topic 5: The Memory Hierarchy
Part A: Caches

These slides are mostly taken verbatim, or with minor changes, from those prepared by Mary Jane Irwin (www.cse.psu.edu/~mji) of The Pennsylvania State University

Key to the Slides

- The source of each slide is coded in the footer on the right side:
  - **Irwin CSE331** = slide by Mary Jane Irwin from the course CSE331 (Computer Organization and Design) at Pennsylvania State University.
  - **Irwin CSE431** = slide by Mary Jane Irwin from the course CSE431 (Computer Architecture) at Pennsylvania State University.
  - **Hegner UU** = slide by Stephen J. Hegner at Umeå University.
Review: Major Components of a Computer

- Processor
  - Control
  - Datapath
- Memory
  - Cache
  - Main Memory
  - Secondary Memory (Disk)
- Devices
  - Input
  - Output
The “Memory Wall”

- Processor vs DRAM speed disparity continues to grow

- Good memory hierarchy (cache) design is increasingly important to overall performance
The Memory Hierarchy Goal

- Fact: Large memories are slow and fast memories are small

- How do we create a memory that gives the illusion of being large, cheap and fast (most of the time)?
  - With hierarchy
  - With parallelism
A Typical Memory Hierarchy

- Take advantage of the **principle of locality** to present the user with as much memory as is available in the **cheapest** technology at the speed offered by the **fastest** technology.

- Nowadays most processors have three levels of on-chip cache.
Recall: the Oracle/(Sun) SPARC T5 (2013)

- Example of a modern high-end processor
  - 8 16 cores, up to 8 threads per core = 64 128 threads total
  - Scalability up to 4 8 sockets with no additional silicon necessary
  - Die size 403 478 mm²
  - Core size 15.4mm²
  - 2.85—3.0 3.6 GHz
  - 40 28 nm feature size
  - Caches:
    - L1: 16KB instruction, 16KB data, per core
    - L2: 128KB, per core, unified
    - L3: 4MB 8MB, 8 banks, 16-way set-associative, unified
  - Two Four memory controllers; up to 16 DDR3 DIMMs
  - On-chip encryption hardware
  - On-chip PCIe Generation 2 3
  - TDP (Total Power Dissipation) 240 ??? Watts maximum
Memory Hierarchy Technologies

- Caches use **SRAM** for speed and technology compatibility
  - Fast (typical access times of 0.5 to 2.5 nsec)
  - Low density (6 transistor cells), higher power, expensive ($2000 to $5000 per GB in 2008)
  - Static: content will last “forever” (as long as power is left on)

- Main memory uses **DRAM** for size (density)
  - Slower (typical access times of 50 to 70 nsec)
  - High density (1 transistor cells), lower power, cheaper ($20 to $75 per GB in 2008; less than $10 per GB in 2013 for an 8GB DDR3 DIMM)
  - Dynamic: needs to be “refreshed” regularly (~ every 8 ms)
    - Consumes 1% to 2% of the active cycles of the DRAM
  - Addresses divided into 2 halves (row and column)
    - **RAS** or *Row Access Strobe* triggering the row decoder
    - **CAS** or *Column Access Strobe* triggering the column selector

- Note: **SDRAM** = *Synchronous* DRAM.
  - Virtually all modern DRAM is synchronous.
The Memory Hierarchy: Why Does it Work?

- **Temporal Locality** (locality in time)
  - If a memory location is referenced then it will tend to be referenced again soon
  
  ⇒ Keep *most recently accessed* data items closer to the processor

- **Spatial Locality** (locality in space)
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon
  
  ⇒ Move blocks consisting of *contiguous words* closer to the processor

- Also, the following is important:
  - **Separate** instruction and data caches at L1
    - This effectively provides separate instruction and data memory without requiring two separate memory channels.
The Memory Hierarchy: Terminology

- **Block** (or line): the minimum unit of information that is present (or not) in a cache
- **Hit Rate**: the fraction of memory accesses found in a level of the memory hierarchy
  - **Hit Time**: Time to access that level which consists of
    - Time to access the block + Time to determine hit/miss
- **Miss Rate**: the fraction of memory accesses *not* found in a level of the memory hierarchy \[ \Rightarrow 1 - \text{(Hit Rate)} \]
  - **Miss Penalty**: Time to replace a block in that level with the corresponding block from a lower level which consists of
    - Time to access the block in the lower level + Time to transmit that block to the level that experienced the miss + Time to insert the block in that level + Time to pass the block to the requestor

Hit Time \(<<\) Miss Penalty
Characteristics of the Memory Hierarchy

- **Increasing distance from the processor in access time**

  - **Processor**
    - Inclusive—what is in L1$ is a subset of what is in L2$ is a subset of what is in MM that is a subset of is in SM.
    - 4-8 bytes (word)
    - 8-32 bytes (block)
    - 1 to 4 blocks
    - 1,024+ bytes (disk sector = page)

  - **Main Memory**
    - (Relative) size of the memory at each level

  - **Secondary Memory**

  - **Increasing** distance from the processor in access time
How is the Hierarchy Managed?

- registers ↔ memory
  - by compiler (programmer?)

- cache ↔ main memory
  - by the cache controller hardware

- main memory ↔ disks
  - by the operating system (virtual memory)
  - virtual to physical address mapping assisted by the hardware (TLB)
  - by the programmer (files)
Cache Basics

- Two questions to answer (in hardware):
  - Q1: How do we know if a data item is in the cache?
  - Q2: If it is, how do we find it?

- Direct mapped
  - Each memory block is mapped to exactly one block in the cache
    - lots of lower level blocks must share blocks in the cache
  - Address mapping (to answer Q2):
    (block address) modulo (# of blocks in the cache)
  - Have a tag associated with each cache block that contains the address information (the upper portion of the address) required to identify the block (to answer Q1)
Caching: A Simple First Example

Q1: Is it there?

Compare the cache tag to the high order 2 memory address bits to tell if the memory block is in the cache.

Q2: How do we find it?

Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache).

One word blocks
Two low order bits define the byte in the word (32b words)
Direct Mapped Cache

Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

<table>
<thead>
<tr>
<th></th>
<th>0 miss</th>
<th>1 miss</th>
<th>2 miss</th>
<th>3 miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Mem(0)</td>
<td>Mem(0)</td>
<td>Mem(0)</td>
<td>Mem(0)</td>
</tr>
<tr>
<td>00</td>
<td>Mem(1)</td>
<td>Mem(1)</td>
<td>Mem(1)</td>
<td>Mem(1)</td>
</tr>
<tr>
<td>00</td>
<td>Mem(2)</td>
<td>Mem(2)</td>
<td>Mem(2)</td>
<td>Mem(2)</td>
</tr>
<tr>
<td>00</td>
<td>Mem(3)</td>
<td>Mem(3)</td>
<td>Mem(3)</td>
<td>Mem(3)</td>
</tr>
</tbody>
</table>

- 4 miss
- 3 hit
- 4 hit
- 15 miss

- 8 requests, 6 misses
MIPS Direct Mapped Cache Example

- One word blocks, cache size = 1K words (or 4KB)

![Cache Diagram]

What kind of locality are we taking advantage of?
Multiword Block Direct Mapped Cache

- Four words/block, cache size = 1K words

What kind of locality are we taking advantage of?
Taking Advantage of Spatial Locality

- Let cache block hold more than one word

Start with an empty cache - all blocks initially marked as not valid

<table>
<thead>
<tr>
<th>0 miss</th>
<th>1 hit</th>
<th>2 miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 Mem(1) Mem(0)</td>
<td>00 Mem(1) Mem(0)</td>
<td>00 Mem(1) Mem(0)</td>
</tr>
<tr>
<td>00 Mem(3) Mem(2)</td>
<td>00 Mem(3) Mem(2)</td>
<td>00 Mem(3) Mem(2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 hit</th>
<th>4 miss</th>
<th>3 hit</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 Mem(1) Mem(0)</td>
<td>00 Mem(1) Mem(0)</td>
<td>00 Mem(3) Mem(2)</td>
</tr>
<tr>
<td>00 Mem(3) Mem(2)</td>
<td>00 Mem(3) Mem(2)</td>
<td>00 Mem(3) Mem(2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4 hit</th>
<th>15 miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 Mem(5) Mem(4)</td>
<td>01 Mem(5) Mem(4)</td>
</tr>
<tr>
<td>00 Mem(3) Mem(2)</td>
<td>00 Mem(3) Mem(2)</td>
</tr>
</tbody>
</table>

- 8 requests, 4 misses
Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)
Cache Field Sizes

- The number of bits in a cache includes both the storage for data and for the tags
  - 32-bit byte address
  - For a direct mapped cache with $2^n$ blocks, $n$ bits are used for the index
  - For a block size of $2^m$ words ($2^{m+2}$ bytes), $m$ bits are used to address the word within the block and 2 bits are used to address the byte within the word

- The total number of bits in a direct-mapped cache is then
  \[ 2^n \times (\text{block size} + \text{tag-field size} + \text{valid-field size}) \]

- The mapping of a 32-bit address into the cache yields
  \[ \text{tag size} + \text{index size} + \text{block-field size} + 2 = 32 \]

- How many total bits are required for a direct mapped cache with 16KB of data and 4-word blocks assuming a 32-bit address?
  - 16KB data $\Rightarrow 2^{14}$ bytes $\Rightarrow 2^{12}$ words $\Rightarrow 2^{10}$ 4-word blocks $\Rightarrow 10$-bit index
  - Tag size = 32 – (index size + block-field size + 2) = 32 – (10 + 2 + 2) = 18 bits
  - Block = 128 bits (4 words) + 18 bits (tag) + 1 bit (valid) = 147 bits
  - Total = $2^{10}$ blocks $\times$ 147 bits/block = 147 $\times$ $2^{10}$ = 150528 bits = 18816 bytes
Handling Cache Hits

- **Read hits (I$ and D$)**
  - this is what we want!

- **Write hits (D$ only)**
  - require the cache and memory to be **consistent**
    - always write the data into both the cache block and the next level in the memory hierarchy (**write-through**)
    - writes run at the speed of the next level in the memory hierarchy – so slow! – or can use a **write buffer** and stall only if the write buffer is full
  - allow cache and memory to be **inconsistent**
    - write the data only into the cache block (**write-back** the cache block to the next level in the memory hierarchy when that cache block is “evicted”)
    - need a **dirty** bit for each data cache block to tell if it needs to be written back to memory when it is evicted – can use a **write buffer** to help “buffer” write-backs of dirty blocks
Sources of Cache Misses

- **Compulsory** (cold start or process migration, first reference):
  - First access to a block, “cold” fact of life, not a whole lot you can do about it. If you are going to run “millions” of instruction, compulsory misses are insignificant.
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)

- **Capacity**:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size (may increase access time)

- **Conflict** (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity (stay tuned) (may increase access time)
Handling Cache Misses (Single Word Blocks)

- Read misses (I$ and D$)
  - stall the pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume (read through)

- Write misses (D$ only)
  1. General write allocate for multiple-word blocks – stall the pipeline, fetch the block from next level in the memory hierarchy, install it in the cache (which may involve having to evict a dirty block if using a write-back cache), write the word from the processor to the cache, then let the pipeline resume, or
  2. Faster write allocate for single-word blocks – just write the word into the cache updating both the tag and data, no need to check for cache hit, no need to stall, or
  3. No-write allocate – skip the cache write (but must invalidate that cache block since it will now hold stale data) and just write the word to the write buffer (and eventually to the next memory level), no need to stall if the write buffer isn’t full
Multiword Block Considerations

- Read misses (I$ and D$)
  - Processed the same as for single word blocks – a miss returns the entire block from memory
  - Miss penalty grows as block size grows
    - Early restart – processor resumes execution as soon as the requested word of the block is returned
    - Requested word first – requested word is transferred from the next level of memory to the cache (and processor) first
  - Nonblocking cache – allows the processor to continue to access the cache while the cache is handling an earlier miss

- Write misses (D$)
  - If using write allocate must first fetch the block from the next level of memory and then write the word to the block (or could end up with a “garbled” block in the cache (e.g., for 4 word blocks, a new tag, one word of data from the new block, and three words of data from the old block))
Review: (DDR) SDRAM Operation

- After a row is read into the SRAM register
  - Input CAS as the starting “burst” address along with a burst length
  - Transfers a burst of data (ideally a cache block) from a series of sequential addr’s within that row
    - The memory bus clock controls transfer of successive words in the burst
Memory Systems that Support Caches

- The off-chip interconnect and memory architecture can affect overall system performance in dramatic ways

One-word-wide organization (one-word-wide bus and one-word-wide memory)

- Assume
  1. 1 memory bus clock cycle to send the addr
  2. 15 memory bus clock cycles to get the 1\textsuperscript{st} word in the block from DRAM (row cycle time), 5 memory bus clock cycles for 2\textsuperscript{nd}, 3\textsuperscript{rd}, 4\textsuperscript{th} words (column access time)
  3. 1 memory bus clock cycle to return a word of data

- Memory-Bus to Cache bandwidth
  - number of bytes accessed from memory and transferred to cache/CPU per memory bus clock cycle
One Word Wide Bus, One Word Blocks

- If the block size is one word, then for a memory access due to a cache miss, the pipeline will have to stall for the number of cycles required to return one data word from memory:
  - 1 memory bus clock cycle to send address
  - 15 memory bus clock cycles to read DRAM
  - 1 memory bus clock cycle to return data
  - 17 total clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is:
  \[ \frac{4}{17} = 0.235 \] bytes per memory bus clock cycle
One Word Wide Bus, Four Word Blocks

- What if the block size is four words and each word is in a different DRAM row?
  - 1 cycle to send 1\textsuperscript{st} address
  - $4 \times 15 = 60$ cycles to read DRAM
  - $\frac{1}{62}$ cycles to return last data word
  - Total clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is
  - $(4 \times 4)/62 = 0.258$ bytes per clock
One Word Wide Bus, Four Word Blocks

- What if the block size is four words and all words are in the same DRAM row?
  - 1 cycle to send 1st address
  - $15 + 3 \times 5 = 30$ cycles to read DRAM
  - $\frac{1}{32}$ cycles to return last data word
  - Total clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is
  \[
  \frac{4 \times 4}{32} = 0.5 \text{ bytes per clock}
  \]
Interleaved Memory, One Word Wide Bus

- For a block size of four words
  - 1 cycle to send 1st address
  - 15 cycles to read DRAM banks
  - \(4 \times 1 = 4\) cycles to return last data word
  - 20 total clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is
  \[
  \frac{(4 \times 4)}{20} = 0.8 \text{ bytes per clock}
  \]
DRAM Memory System Summary

- It is important to match the cache characteristics
  - caches access one block at a time (usually more than one word)

- with the DRAM characteristics
  - use DRAMs that support fast multiple word accesses, preferably ones that match the block size of the cache

- with the memory-bus characteristics
  - make sure the memory-bus can support the DRAM access rates and patterns
  - with the goal of increasing the Memory-Bus to Cache bandwidth
Measuring Cache Performance

- Assuming cache hit costs are included as part of the normal CPU execution cycle, then

  \[ \text{CPU time} = IC \times CPI \times CC \]
  \[ = IC \times (\text{CPI}_{\text{ideal}} + \text{Memory-stall cycles}) \times CC \]

- Memory-stall cycles come from cache misses (a sum of read-stalls and write-stalls)

  \[ \text{Read-stall cycles} = \text{reads/program} \times \text{read miss rate} \times \text{read miss penalty} \]
  \[ \text{Write-stall cycles} = (\text{writes/program} \times \text{write miss rate} \times \text{write miss penalty}) + \text{write buffer stalls} \]

- For write-through caches, we can simplify this to

  \[ \text{Memory-stall cycles} = \text{accesses/program} \times \text{miss rate} \times \text{miss penalty} \]
Impacts of Cache Performance

- Relative cache penalty increases as processor performance improves (faster clock rate and/or lower CPI)
  - The memory speed is unlikely to improve as fast as processor cycle time. When calculating $CPI_{stalls}$, the cache miss penalty is measured in *processor* clock cycles needed to handle a miss.
  - The lower the $CPI_{ideal}$, the more pronounced the impact of stalls.

- A processor with a $CPI_{ideal}$ of 2, a 100 cycle miss penalty, 36% load/store instr’s, and 2% I$ and 4% D$ miss rates
  
  Memory-stall cycles = $2\% \times 100 + 36\% \times 4\% \times 100 = 3.44$
  
  So $CPI_{stalls} = 2 + 3.44 = 5.44$
  
  more than twice the $CPI_{ideal}$!

- What if the $CPI_{ideal}$ is reduced to 1? 0.5? 0.25?
- What if the D$ miss rate went up 1%? 2%?
- What if the processor clock rate is doubled (doubling the miss penalty)?
Average Memory Access Time (AMAT)

- A larger cache will have a longer access time. An increase in hit time will likely add another stage to the pipeline. At some point the increase in hit time for a larger cache will overcome the improvement in hit rate leading to a decrease in performance.

- Average Memory Access Time (AMAT) is the average time to access memory considering both hits and misses:

  \[
  \text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}
  \]

- What is the AMAT for a processor with a 20 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache access time of 1 clock cycle?
  
  \[
  \text{AMAT} = 20 \text{ ps.} + 0.02 \times (50 \text{ cycles} \times 20 \text{ ps./cycle}) = 40 \text{ ps.}
  \]
Reducing Cache Miss Rates #1

1. Allow more flexible block placement

- In a direct mapped cache a memory block maps to exactly one cache block
- At the other extreme, could allow a memory block to be mapped to any cache block – fully associative cache

- A compromise is to divide the cache into sets each of which consists of n “ways” (n-way set associative). A memory block maps to a unique set (specified by the index field) and can be placed in any way of that set (so there are n choices)

  \[ \text{(block address) modulo (# sets in the cache)} \]
Another Reference String Mapping

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

```
0 4 0 4 0 4 0 4
```

- Miss

- \[ \text{Start with an empty cache - all blocks initially marked as not valid} \]

- Ping pong effect due to conflict misses - two memory locations that map into the same cache block

- 8 requests, 8 misses

- Ping pong effect due to conflict misses - two memory locations that map into the same cache block
Set Associative Cache Example

Q1: Is it there?

Compare *all* the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache.

Q2: How do we find it?

Use the *next 1 low order memory address bit* to determine which cache set (i.e., modulo the number of sets in the cache).

Main Memory

One word blocks
Two low order bits define the byte in the word (32b words)
Another Reference String Mapping

Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 4 0 4 0 4 0 4


- 0 miss

000 Mem(0)

- 4 miss

000 Mem(0)

010 Mem(4)

- 0 hit

000 Mem(0)

010 Mem(4)

- 4 hit

000 Mem(0)

010 Mem(4)

- 8 requests, 2 misses

- Solves the ping pong effect in a direct mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist!
Four-Way Set Associative Cache

- $2^8 = 256$ sets each with four ways (each with one block)

Byte offset:

```
31 30 ... 13 12 11 ... 2 1 0
```

Diagram:

- Tag
- Index
- Way 0
- Way 1
- Way 2
- Way 3
- Hit
- Data
**Range of Set Associative Caches**

- For a fixed size cache, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or ways) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit.

<table>
<thead>
<tr>
<th>Used for tag compare</th>
<th>Selects the set</th>
<th>Selects the word in the block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Index</td>
<td>Block offset Byte offset</td>
</tr>
</tbody>
</table>

- **Decreasing associativity**
  - Direct mapped (only one way)
  - Smaller tags, only a single comparator
  - Increasing associativity

- **Increasing associativity**
  - Fully associative (only one set)
  - Tag is all the bits except block and byte offset
Costs of Set Associative Caches

- When a miss occurs, which way’s block do we pick for replacement?
  - Least Recently Used (LRU): the block replaced is the one that has been unused for the longest time
    - Must have hardware to keep track of when each way’s block was used relative to the other blocks in the set
    - For 2-way set associative, takes one bit per set → set the bit when a block is referenced (and reset the other way’s bit)

- N-way set associative cache costs
  - N comparators (delay and area)
  - MUX delay (set selection) before data is available
  - Data available after set selection (and Hit/Miss decision). In a direct mapped cache, the cache block is available before the Hit/Miss decision
    - So it is not possible to just assume a hit and continue and recover later if it was a miss
Benefits of Set Associative Caches

- The choice of direct mapped or set associative depends on the cost of a miss versus the cost of implementation.

- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate).

Data from Hennessy & Patterson, *Computer Architecture*, 2003
Reducing Cache Miss Rates #2

1. Use multiple levels of caches

- With advancing technology have more than enough room on the die for bigger L1 caches or for a second level of caches – normally a unified L2 cache (i.e., it holds both instructions and data) and in some cases even a unified L3 cache

- For our example, CPI_{ideal} of 2, 100 cycle miss penalty (to main memory) and a 25 cycle miss penalty (to UL2$), 36% load/stores, a 2% (4%) L1 I$ (D$) miss rate, add a 0.5% UL2$ miss rate

\[
\text{CPI}_{\text{stalls}} = 2 + 0.02 \times 25 + 0.36 \times 0.04 \times 25 + 0.005 \times 100 + 0.36 \times 0.005 \times 100 = 3.54
\]

(as compared to 5.44 with no L2$)
Multilevel Cache Design Considerations

- Design considerations for L1 and L2 caches are very different
  - Primary cache should focus on **minimizing hit time** in support of a shorter clock cycle
    - Smaller with smaller block sizes
  - Secondary cache(s) should focus on **reducing miss rate** to reduce the penalty of long main memory access times
    - Larger with larger block sizes
    - Higher levels of associativity

- The miss penalty of the L1 cache is significantly reduced by the presence of an L2 cache – so it can be smaller (i.e., faster) but have a higher miss rate

- For the L2 cache, hit time is less important than miss rate
  - The L2$ hit time determines L1$’s miss penalty
  - L2$ local miss rate >> than the global miss rate
## Two Machines’ Cache Parameters

<table>
<thead>
<tr>
<th></th>
<th>Intel Nehalem</th>
<th>AMD Barcelona</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L1 cache</strong></td>
<td>Split I$ and D$; 32KB for each per core; 64B blocks</td>
<td>Split I$ and D$; 64KB for each per core; 64B blocks</td>
</tr>
<tr>
<td><strong>organization &amp; size</strong></td>
<td>4-way (I), 8-way (D) set assoc.; ~LRU replacement</td>
<td>2-way set assoc.; LRU replacement</td>
</tr>
<tr>
<td><strong>associativity</strong></td>
<td>write-back, write-allocate</td>
<td>write-back, write-allocate</td>
</tr>
<tr>
<td><strong>L2 cache</strong></td>
<td>Unified; 256KB (0.25MB) per core; 64B blocks</td>
<td>Unified; 512KB (0.5MB) per core; 64B blocks</td>
</tr>
<tr>
<td><strong>organization &amp; size</strong></td>
<td>8-way set assoc.; ~LRU</td>
<td>16-way set assoc.; ~LRU</td>
</tr>
<tr>
<td><strong>associativity</strong></td>
<td>write-back</td>
<td>write-back</td>
</tr>
<tr>
<td><strong>write policy</strong></td>
<td>write-back, write-allocate</td>
<td>write-back, write-allocate</td>
</tr>
<tr>
<td><strong>L3 cache</strong></td>
<td>Unified; 8192KB (8MB) shared by cores; 64B blocks</td>
<td>Unified; 2048KB (2MB) shared by cores; 64B blocks</td>
</tr>
<tr>
<td><strong>organization &amp; size</strong></td>
<td>16-way set assoc.</td>
<td>32-way set assoc.; evict block shared by fewest cores</td>
</tr>
<tr>
<td><strong>associativity</strong></td>
<td>write-back, write-allocate</td>
<td>write-back; write-allocate</td>
</tr>
<tr>
<td><strong>write policy</strong></td>
<td>write-back, write-allocate</td>
<td>write-back, write-allocate</td>
</tr>
</tbody>
</table>
## Two Older Machines’ Cache Parameters

<table>
<thead>
<tr>
<th></th>
<th>Intel P4</th>
<th>AMD Opteron</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 organization</td>
<td>Split I$ and D$</td>
<td>Split I$ and D$</td>
</tr>
<tr>
<td>L1 cache size</td>
<td>8KB for D$, 96KB for trace cache (~I$)</td>
<td>64KB for each of I$ and D$</td>
</tr>
<tr>
<td>L1 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L1 associativity</td>
<td>4-way set assoc.</td>
<td>2-way set assoc.</td>
</tr>
<tr>
<td>L1 replacement</td>
<td>~ LRU</td>
<td>LRU</td>
</tr>
<tr>
<td>L1 write policy</td>
<td>write-through</td>
<td>write-back</td>
</tr>
<tr>
<td>L2 organization</td>
<td>Unified</td>
<td>Unified</td>
</tr>
<tr>
<td>L2 cache size</td>
<td>512KB</td>
<td>1024KB (1MB)</td>
</tr>
<tr>
<td>L2 block size</td>
<td>128 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L2 associativity</td>
<td>8-way set assoc.</td>
<td>16-way set assoc.</td>
</tr>
<tr>
<td>L2 replacement</td>
<td>~LRU</td>
<td>~LRU</td>
</tr>
<tr>
<td>L2 write policy</td>
<td>write-back</td>
<td>write-back</td>
</tr>
</tbody>
</table>
FSM Cache Controller

- Key characteristics for a simple L1 cache
  - Direct mapped
  - Write-back using write-allocate
  - Block size of 4 32-bit words (so 16B); Cache size of 16KB (so 1024 blocks)
  - 18-bit tags, 10-bit index, 2-bit block offset, 2-bit byte offset, dirty bit, valid bit, LRU bits (if set associative)
Four State Cache Controller

Idle

- Cache Hit
- Mark Cache Ready
- Valid CPU request

Allocate
- Read new block from memory
- Memory Not Ready

Compare Tag
- If Valid && Hit
- Set Valid, Set Tag,
- If Write set Dirty

Write Back
- Write old block to memory
- Memory Not Ready

Cache Miss
- Old block is clean
- Cache Miss
- Old block is Dirty

Memory Ready

Memory
- Not Ready
Summary: Improving Cache Performance

0. Reduce the time to hit in the cache
   - smaller cache
   - direct mapped cache
   - smaller blocks
   - for writes
     - no write allocate – no “hit” on cache, just write to write buffer
     - write allocate – to avoid two cycles (first check for hit, then write)
       pipeline writes via a delayed write buffer to cache

1. Reduce the miss rate
   - bigger cache
   - more flexible placement (increase associativity)
   - larger blocks (16 to 64 bytes typical)
   - victim cache – small buffer holding most recently discarded blocks
Summary: Improving Cache Performance

2. Reduce the miss penalty
   - smaller blocks
   - use a write buffer to hold dirty blocks being replaced so don’t have to wait for the write to complete before reading
   - check write buffer (and/or victim cache) on read miss – may get lucky
   - for large blocks fetch critical word first
   - use multiple cache levels – L2 cache not tied to CPU clock rate
   - faster backing store/improved memory bandwidth
     - wider buses
     - memory interleaving, DDR SDRAMs
Summary: The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation

- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost

- Simplicity often wins