Recap: Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operate in parallel
- Example: Two-way set associative cache
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared to the input in parallel
  - Data is selected based on the tag result

Recap: Cache Performance

Execution Time = Instruction Count x Cycle Time x (ideal CPI + Memory_Stats/Inst + Other_Stats/Inst)

Memory_Stats/Inst = Instruction Miss Rate x Instruction Miss Penalty + Loads/Inst x Load Miss Rate x Load Miss Penalty + Stores/Inst x Store Miss Rate x Store Miss Penalty

Average Memory Access time (AMAT) = Hit Time_L1 + (Miss Rate_L1 x Miss Penalty_L1) = (Hit Rate_L1 x Hit Time_L1) + (Miss Rate_L1 x Miss Time_L1)

Recap: A Summary on 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
  - Note: If you are going to run “billions” of instruction, Compulsory Misses are insignificant
- Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- Capacity:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size
- Coherence (Invalidation): other process (e.g., I/O) updates memory
The Big Picture: Where are We Now?

- The Five Classic Components of a Computer
  - Processor
  - Control
  - Memory
  - Datapath
  - Input
  - Output

Today’s Topics:
- Recap last lecture
- Virtual Memory
- Protection
- TLB
- Buses

How Do you Design a Memory System?

- Set of Operations that must be supported
  - read: data <= Mem[Physical Address]
  - write: Mem[Physical Address] <= Data

  Inside it has: Tag-Data Storage, Muxes, Comparators, ...

- Determine the internal register transfers
- Design the Datapath
- Design the Cache Controller

Impact on Cycle Time

Cache Hit Time:
- directly tied to clock rate
- increases with cache size
- increases with associativity

Average Memory Access time = Hit Time + Miss Rate x Miss Penalty

Time = IC x CT x (ideal CPI + memory stalls)

Improving Cache Performance: 3 general options

Average Memory Access time =
Hit Time + (Miss Rate x Miss Penalty) =
(Hit Rate x Hit Time) + (Miss Rate x Miss Time)

Options to reduce AMAT:
1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.
Improving Cache Performance

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

2:1 Cache Rule

\[
\text{miss rate 1-way associative cache size } X = \text{miss rate 2-way associative cache size } X/2
\]

3Cs Absolute Miss Rate (SPEC92)

3Cs Relative Miss Rate
1. Reduce Misses via Larger Block Size

![Graph showing block size vs. miss rate]

- Miss Rate vs. Block Size (bytes)
- 1K, 4K, 16K, 64K, 256K

2. Reduce Misses via Higher Associativity

- 2:1 Cache Rule:
  - Miss Rate DM cache size $N$ - Miss Rate 2-way cache size $N/2$
- Beware: Execution time is only final measure!
  - Will Clock Cycle time increase?
  - Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%, internal + 2%

3. Reducing Misses via a “Victim Cache”

- Assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped

<table>
<thead>
<tr>
<th>Cache Size (KB)</th>
<th>1-way</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.33</td>
<td>2.15</td>
<td>2.07</td>
<td>2.01</td>
</tr>
<tr>
<td>2</td>
<td>1.98</td>
<td>1.86</td>
<td>1.76</td>
<td>1.68</td>
</tr>
<tr>
<td>4</td>
<td>1.72</td>
<td>1.67</td>
<td>1.61</td>
<td>1.53</td>
</tr>
<tr>
<td>8</td>
<td>1.46</td>
<td>1.48</td>
<td>1.47</td>
<td>1.43</td>
</tr>
<tr>
<td>16</td>
<td>1.29</td>
<td>1.32</td>
<td>1.32</td>
<td>1.32</td>
</tr>
<tr>
<td>32</td>
<td>1.20</td>
<td>1.24</td>
<td>1.25</td>
<td>1.27</td>
</tr>
<tr>
<td>64</td>
<td>1.14</td>
<td>1.20</td>
<td>1.21</td>
<td>1.23</td>
</tr>
<tr>
<td>128</td>
<td>1.10</td>
<td>1.17</td>
<td>1.18</td>
<td>1.20</td>
</tr>
</tbody>
</table>

(Red means A.M.A.T. not improved by more associativity)
4. Reducing Misses by Hardware Prefetching

- E.g., Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in “stream buffer”
  - On miss check stream buffer
- Works with data blocks too:
  - Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
  - Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches
- Prefetching relies on having extra memory bandwidth that can be used without penalty
  - Could reduce performance if done indiscriminantly!!

5. Reducing Misses by Software Prefetching Data

- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache
    (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

6. Reducing Misses by Compiler Optimizations

- McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts(using tools they developed)
- Data
  - Merging Arrays: improve spatial locality by single array of compound elements vs. 2 arrays
  - Loop Interchange: change nesting of loops to access data in order stored in memory
  - Loop Fusion: Combine 2 independent loops that have same looping and some variables overlap
  - Blocking: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows

Administrivia

- Second midterm coming up (Monday, May 5th)
  Will be 5:30 - 8:30 in 306 Soda Hall. LaVal’s afterwards!
- Pipelining
  - Hazards, branches, forwarding, CPI calculations
  - (may include something on dynamic scheduling)
- Memory Hierarchy (including Caches, TLBs, DRAM)
- Simple Power issues
- Possibly I/O
- Tomorrow Sections ⇒ Lab
  - Give a design review to your TA.
  - Bring block diagrams, design notes, etc.
- Lab 6 debugging?
  - Dual Port memories: Not a good idea unless absolutely necessary
    - Often built as multiple banks of single-port RAMs
  - Fifo queuing to handle clock domains: a good idea
    - Can actually improve DRAM performance!
    - Multiple clock domains often an issue right around DRAM
Improving Cache Performance (Continued)

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

---

0. Reducing Penalty: Faster DRAM / Interface

° New DRAM Technologies
  - RAMBUS - same initial latency, but much higher bandwidth
  - Synchronous DRAM
  - TMJ-RAM (Tunneling magnetic-junction RAM) from IBM??
  - Merged DRAM/Logic - IRAM project here at Berkeley

° Better BUS interfaces
  - Simple Lab6 example: add write fifo to DRAM controller
    - Cache can dump to DRAM immediately without waiting
    for RAS/CAS....

° CRAY Technique: only use SRAM

---

1. Reducing Penalty: Read Priority over Write on Miss

A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory

° Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
  - Must handle burst behavior as well!

---

RAW Hazards from Write Buffer!

° Write-Buffer Issues: Could introduce RAW Hazard with memory!
  - Write buffer may contain only copy of valid data ⇒
    Reads to memory may get wrong result if we ignore write buffer

° Solutions:
  - Simply wait for write buffer to empty before servicing reads:
    - Might increase read miss penalty (old MIPS 1000 by 50%)
  - Check write buffer contents before read (“fully associative”):
    - If no conflicts, let the memory access continue
    - Else grab data from buffer

° Can Write Buffer help with Write Back?
  - Read miss replacing dirty block
    - Copy dirty block to write buffer while starting read to memory
2. Reduce Penalty: Early Restart and Critical Word First

- Don’t wait for full block to be loaded before restarting CPU
  - *Early restart*—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called *wrapped fetch* and *requested word first*

  - Generally useful only in large blocks,
  - Spatial locality a problem; tend to want next sequential word, so not clear if benefit by early restart

3. Reduce Penalty: Non-blocking Caches

- *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss
  - Requires F/E bits on registers or out-of-order execution
  - Requires multi-bank memories
  - “hit under miss” reduces the effective miss penalty by working during miss vs. ignoring CPU requests
  - “hit under multiple miss” or “miss under miss” may further lower the effective miss penalty by overlapping multiple misses

  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires multiple memory banks (otherwise cannot support)
  - Pentium Pro allows 4 outstanding memory misses

What happens on a Cache miss?

- For in-order pipeline, 2 options:
  - Freeze pipeline in Mem stage (popular early on: Sparc, R4000)
    - IF ID EX Mem stall stall stall ... stall Mem Wr
    - IF ID EX stall stall stall ... stall stall Ex Wr
  - Use Full/Empty bits in registers + MSHR queue
    - MSHR = "Miss Status/Handler Registers" (Kroft)
      - Each entry in this queue keeps track of status of outstanding memory requests to one complete memory line.
        - Per cache-line: keep info about memory address.
        - For each word: register (if any) that is waiting for result.
        - Used to “merge” multiple requests to one memory line
      - New load creates MSHR entry and sets destination register to "Empty". Load is "released" from pipeline.
      - Attempt to use register before result returns causes instruction to block in decode stage.
      - Limited “out-of-order” execution with respect to loads.
        - Popular with in-order superscalar architectures.
  - Out-of-order pipelines already have this functionality built in... (load queues, etc).

Value of Hit Under Miss for SPEC

- FP programs on average: Penalty = 0.68 -> 0.52 -> 0.34 -> 0.26
- Int programs on average: Penalty = 0.24 -> 0.20 -> 0.19 -> 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss
4. Reduce Penalty: Second-Level Cache

- **L2 Equations**
  
  $\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$

  $\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$

  $\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$

- **Definitions:**
  
  - *Local miss rate*—misses in this cache divided by the total number of memory accesses to this cache ($\text{Miss rate}_{L2}$)
  
  - *Global miss rate*—misses in this cache divided by the total number of memory accesses generated by the CPU ($\text{Miss Rate}_{L1} \times \text{Miss Rate}_{L2}$)

  - Global Miss Rate is what matters

Reducing Misses: which apply to L2 Cache?

- **Reducing Miss Rate**
  
  1. Reduce Misses via Larger Block Size
  2. Reduce Conflict Misses via Higher Associativity
  3. Reducing Conflict Misses via Victim Cache
  4. Reducing Misses by HW Prefetching Instr, Data
  5. Reducing Misses by SW Prefetching Data
  6. Reducing Capacity/Conf. Misses by Compiler Optimizations

Improving Cache Performance (Continued)

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache:

   - Lower Associativity (+victim caching or 2nd-level cache)?
   - Multiple cycle Cache access (e.g. R4000)
   - Harvard Architecture
   - Careful Virtual Memory Design (rest of lecture!)
° Sample Statistics:
  - 16KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
  - 32KB unified: Aggregate miss rate=1.99%
° Which is better (ignore L2 cache)?
  - Assume 33% loads/store, hit time=1, miss time=50
  - Note: data hit has 1 stall for unified cache (only one port)

\[
\text{AMAT}_{\text{Harvard}} = \frac{1}{1 + 0.64\% \times 50} + \frac{0.33}{1.33} \times (1 + 6.47\% \times 50) = 2.05
\]
\[
\text{AMAT}_{\text{Unified}} = \frac{1}{1.33} \times (1 + 1.99\% \times 50) + \frac{0.33}{1.33} \times (1 + 1 + 1.99\% \times 50) = 2.24
\]

° Summary: Cache techniques
  - Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions:
    1) Where can block be placed?
    2) How is block found?
    3) What block is replaced on miss?
    4) How are writes handled?
  - More cynical version of this: Everything in computer architecture is a cache!
  - Techniques people use to improve the miss rate of caches:

<table>
<thead>
<tr>
<th>Technique</th>
<th>MR</th>
<th>MP</th>
<th>HT</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger Block Size</td>
<td>+</td>
<td>–</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>Higher Associativity</td>
<td>+</td>
<td>–</td>
<td>–</td>
<td>1</td>
</tr>
<tr>
<td>Victim Caches</td>
<td>+</td>
<td>–</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Pseudo-Associative Caches</td>
<td>+</td>
<td>–</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>HW Prefetching of Instr/Data</td>
<td>+</td>
<td>–</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Compiler Controlled Prefetching</td>
<td>+</td>
<td>–</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Compiler Reduce Misses</td>
<td>+</td>
<td>–</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

 Example: Harvard Architecture

<table>
<thead>
<tr>
<th>Proc</th>
<th>I-Cache-1</th>
<th>Proc</th>
<th>D-Cache-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unified Cache-1</td>
<td>Unified Cache-2</td>
<td>Unified Cache-2</td>
<td></td>
</tr>
</tbody>
</table>

Harvard Architecture

4/23/03 ©UCB Spring 2003 CS152 / Kubiatowicz Lec22.33 4/23/03 ©UCB Spring 2003 CS152 / Kubiatowicz Lec22.34