CS152
Computer Architecture and Engineering
Lecture 21

Memory Systems (recap)
Caches

November 11th, 2001
John Kubiatowicz (http.cs.berkeley.edu/~kubitron)

lecture slides: http://www-inst.eecs.berkeley.edu/~cs152/

Recall: The Big Picture: Where are We Now?
° The Five Classic Components of a Computer

Recall: Who Cares About the Memory Hierarchy?
Processor-DRAM Memory Gap (latency)

Recap: Memory Hierarchy of a Modern Computer System
° By taking advantage of the principle of locality:
  • Present the user with as much memory as is available in the cheapest technology.
  • Provide access at the speed offered by the fastest technology.

Recall: Who Cares About the Memory Hierarchy?
Processor-DRAM Memory Gap (latency)
Recap: Memory Hierarchy: Why Does it Work? Locality!

- **Temporal Locality (Locality in Time):**
  - Keep most recently accessed data items closer to the processor

- **Spatial Locality (Locality in Space):**
  - Move blocks consists of contiguous words to the upper levels

Recap: Cache Performance

\[
\text{Execution Time} = \text{Instruction Count} \times \text{Cycle Time} \times (\text{ideal CPI} + \frac{\text{Memory Stalls/Inst}}{\text{Inst}} + \frac{\text{Other Stalls/Inst}}{\text{Inst}})
\]

\[
\text{Memory Stalls/Inst} = \text{Instruction Miss Rate} \times \text{Instruction Miss Penalty} + \text{Loads/Inst} \times \text{Load Miss Rate} \times \text{Load Miss Penalty} + \text{Stores/Inst} \times \text{Store Miss Rate} \times \text{Store Miss Penalty}
\]

Average Memory Access time (AMAT) =
\[
\text{Hit Time}_{L1} + (\text{Miss Rate}_{L1} \times \text{Miss Time}_{L1}) = (\text{Hit Rate}_{L1} \times \text{Hit Time}_{L1}) + (\text{Miss Rate}_{L1} \times \text{Miss Time}_{L1})
\]

Recall: Static RAM Cell

6-Transistor SRAM Cell

- **Write:**
  1. Drive bit lines (bit=1, bit=0)
  2.. Select row

- **Read:**
  1. Precharge bit and bit to Vdd or Vdd/2 => make sure equal!
  2.. Select row
  3. Cell pulls one line low
  4. Sense amp on column detects difference between bit and bit

Recall: 1-Transistor Memory Cell (DRAM)

- **Write:**
  1. Drive bit line
  2.. Select row

- **Read:**
  1. Precharge bit line to Vdd/2
  2.. Select row
  3. Cell and bit line share charges
     - Very small voltage changes on the bit line
  4. Sense (fancy sense amp)
     - Can detect changes of ~1 million electrons
  5. Write: restore the value

- **Refresh**
  1. Just do a dummy read to every cell.
Recall: Classical DRAM Organization (square)

- Row and Column Address Select 1 bit at a time
- Act of reading refreshes one complete row
  - Sense amps detect slight variations from VDD/2 and amplify them

Recall: Classical DRAM Organization

Recall: DRAM Read Timing

- Every DRAM access begins at:
  - The assertion of the RAS_L
  - 2 ways to read:
    - early or late v. CAS

Recall: Fast Page Mode Operation

- Regular DRAM Organization:
  - N rows x N column x M-bit
  - Read & Write M-bit at a time
  - Each M-bit access requires a RAS / CAS cycle
- Fast Page Mode DRAM
  - N x M “SRAM” to save a row
- After a row is read into the register
  - Only CAS is needed to access other M-bit blocks on that row
  - RAS_L remains asserted while CAS_L is toggled

SDRAM: Synchronous DRAM?

- More complicated, on-chip controller
  - Operations synchronized to clock
    - So, give row address one cycle
    - Column address some number of cycles later (say 3)
    - Date comes out later (say 2 cycles later)
  - Burst modes
    - Typical might be 1,2,4,8, or 256 length burst
    - Thus, only give RAS and CAS once for all of these accesses
  - Multi-bank operation (on-chip interleaving)
    - Lets you overlap startup latency (5 cycles above) of two banks
- Careful of timing specs!
  - 10ns SDRAM may still require 50ns to get first data!
  - 50ns DRAM means first data out in 50ns
Memroy Systems: Delay more than RAW DRAM

\[
T_c = T_{cycle} + T_{controller} + T_{driver}
\]

Main Memory Performance

- **Simple**: CPU, Cache, Bus, Memory same width (32 bits)
- **Interleaved**: CPU, Cache, Bus 1 word; Memory N Modules (4 Modules); example is word interleaved
- **Wide**: CPU/Mux 1 word; Mux/Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits)

° **DRAM (Read/Write) Cycle Time** >> **DRAM (Read/Write) Access Time**
° DRAM (Read/Write) Cycle Time:
  - How frequent can you initiate an access? \(1/T_{cycle}\)
  - Analogy: A little kid can only ask his father for money on Saturday
° DRAM (Read/Write) Access Time:
  - How quickly will you get what you want once you initiate an access?
  - Analogy: As soon as he asks, his father will give him the money

Access Pattern without Interleaving:

Access Pattern with 4-way Interleaving:
Timing model
- 1 to send address,
- 4 for access time, 10 cycle time, 1 to return data
- Cache Block is 4 words

Simple M.P. = 4 x (1+10+1) = 48
Wide M.P. = 1 + 10 + 1 = 12
Interleaved M.P. = 1+10+1+3 =15

How many banks?
- number banks ≥ number clocks to access word in bank
  - For sequential accesses, otherwise will return to original bank before it has next word ready
  - For strided accesses (access every N words), want prime number of banks!!! (Address computation easy if prime of form 2^k-1)

Increasing DRAM => fewer chips => harder to have banks
- Growth bits/chip DRAM : 50%-60%/yr
- Nathan Myrvold M/S: mature software growth (33%/yr for NT) - growth MB/§ of DRAM (25%-30%/yr)

Independent Memory Banks

Fewer DRAMs/System over Time
(from Pete MacWilliams, Intel)

<table>
<thead>
<tr>
<th>DRAM Generation</th>
<th>1Mb</th>
<th>4Mb</th>
<th>16Mb</th>
<th>64Mb</th>
<th>256Mb</th>
<th>1Gb</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘86</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘89</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘92</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘96</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘99</td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘02</td>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory per DRAM growth @ 60% / year

Minimum PC Memory Size

<table>
<thead>
<tr>
<th>Memory per System growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 MB</td>
</tr>
<tr>
<td>8 MB</td>
</tr>
<tr>
<td>16 MB</td>
</tr>
<tr>
<td>32 MB</td>
</tr>
<tr>
<td>64 MB @ 25%-30% / year</td>
</tr>
<tr>
<td>128 MB</td>
</tr>
<tr>
<td>256 MB</td>
</tr>
</tbody>
</table>

Administrative Issues
- Should be reading Chapter 7 of your book
- Second midterm: Wednesday November 28th
  - Pipelining
    - Hazards, branches, forwarding, CPI calculations
    - (may include something on dynamic scheduling)
  - Memory Hierarchy
  - Possibly something on I/O (see where we get in lectures)
  - Possibly something on power (Broderson Lecture)
- Lab 6: Hopefully all is well!
  - Sorry about fact that burst writes don’t work: will fix this!
  - Cache/DRAM Bus options:
    - #0 (no extra credit): 32-bit bus, 1 DRAM, 4 RAS/CAS per cache line
    - #1 (extra cred 2a): 64-bit bus, 2 DRAMs, 2 RAS/CAS per line
    - #2 (extra cred 2b): 32-bit bus, 1 DRAM, 1 RAS/CAS, 3 CAS per line
  - Be careful for option #1: single word write must only mod 1 DRAM!
    - This would happen with write-through policy!
**The Art of Memory System Design**

Processor

Memory

Optimize the memory system organization to minimize the average memory access time for typical workloads.

---

**Example: 1 KB Direct Mapped Cache with 32 B Blocks**

- For a $2^N$ byte cache:
  - The uppermost $(32 - N)$ bits are always the Cache Tag
  - The lowest $N$ bits are the Byte Select (Block Size = $2^N$)
  - One cache miss, pull in complete “Cache Block” (or “Cache Line”)

---

**Set Associative Cache**

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates 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

---

**Disadvantage of Set Associative Cache**

- N-way Set Associative Cache versus Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss decision and set selection

- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.
Example: Fully Associative

- **Fully Associative Cache**
  - Forget about the Cache Index
  - Compare the Cache Tags of all cache entries in parallel
  - Example: Block Size = 32 B blocks, we need N 27-bit comparators

- **By definition: Conflict Miss = 0 for a fully associative cache**

- Example: Block Size = 32 B blocks, we need N 27-bit comparators
  - By definition: Conflict Miss = 0 for a fully associative cache

- Cache Data
  - Byte 0
  - Cache Tag (27 bits long)
  - Valid Bit
  - Byte 32
  - Byte 33

- Cache Tag
  - Byte Select
  - Ex: 0x01

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

- **Capacity**:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size

- **Conflict** (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity

- **Coherence** (Invalidation): other process (e.g., I/O) updates memory

Design options at constant cost

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>Direct Mapped</th>
<th>N-way Set Associative</th>
<th>Fully Associative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory Miss</td>
<td>Same</td>
<td>Same</td>
<td>Same</td>
</tr>
<tr>
<td>Conflict Miss</td>
<td>High</td>
<td>Medium</td>
<td>Zero</td>
</tr>
<tr>
<td>Capacity Miss</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
</tbody>
</table>

Note: If you are going to run “billions” of instruction, Compulsory Misses are insignificant (except for streaming media types of programs).

Recap: Four Questions for Caches and Memory Hierarchy

- **Q1**: Where can a block be placed in the upper level? *(Block placement)*
- **Q2**: How is a block found if it is in the upper level? *(Block identification)*
- **Q3**: Which block should be replaced on a miss? *(Block replacement)*
- **Q4**: What happens on a write? *(Write strategy)*
Q1: Where can a block be placed in the upper level?
- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets

- Fully associative:
  - block 12 can go anywhere

- Direct mapped:
  - block 12 can go only into block 4 (12 mod 8)

- Set associative:
  - block 12 can go anywhere in set 0 (12 mod 4)

Q2: How is a block found if it is in the upper level?
- Direct indexing (using index and block offset), tag compares, or combination
- Increasing associativity shrinks index, expands tag

Q3: Which block should be replaced on a miss?
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Associativity: 2-way 4-way 8-way
Size LRU Random LRU Random LRU Random
16 KB 5.2% 5.7% 4.7% 5.3% 4.4% 5.0%
64 KB 1.9% 2.0% 1.5% 1.7% 1.4% 1.5%
256 KB 1.15% 1.17% 1.13% 1.13% 1.12% 1.12%

Q4: What happens on a write?
- **Write through**—The information is written to both the block in the cache and to the block in the lower-level memory.
- **Write back**—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - is block clean or dirty?
  - Pros and Cons of each?
    - WT: read misses cannot result in writes
    - WB: no writes of repeated writes
  - WT always combined with write buffers so that don’t wait for lower level memory
Write Buffer for Write Through

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

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

Write Buffer Saturation

- Store frequency (w.r.t. time) > 1 / DRAM write cycle
  - If this condition exist for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
    - Store buffer will overflow no matter how big you make it
    - The CPU Cycle Time <= DRAM Write Cycle Time

Solution for write buffer saturation:
- Use a write back cache
- Install a second level (L2) cache: (does this always work?)

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
  - CPU stall less since restarts as soon as do read

Write-miss Policy: Write Allocate versus Not Allocate

- Assume: a 16-bit write to memory location 0x0 and causes a miss
  - Do we allocate space in cache and possibly read in the block?
    - Yes: Write Allocate
    - No: Not Write Allocate

31 9 4 0
Cache Tag | Example: 0x00 | Cache Index | Byte Select
--- | --- | --- | ---
Ex: 0x00 | Ex: 0x00

Valid Bit
Cache Tag
0x50
0x63

Cache Data
Byte 31  ** Byte 1 Byte 0
1
2
3
Byte 1023  ** Byte 992 31
Impact of Memory Hierarchy on Algorithms

- Today CPU time is a function of (ops, cache misses)
- What does this mean to Compilers, Data structures, Algorithms?
  - Quicksort: fastest comparison based sorting algorithm when keys fit in memory
  - Radix sort: also called “linear time” sort For keys of fixed length and fixed radix a constant number of passes over the data is sufficient independent of the number of keys
  - For Alphastation 250, 32 byte blocks, direct mapped L2 2MB cache, 8 byte keys, from 4000 to 4000000

Quicksort vs. Radix as vary number keys: Instructions

- Radix sort
- Quicksort (Instructions/key)

Quicksort vs. Radix as vary number keys: Instrs & Time

- Radix sort
- Quicksort (Instructions/key)

Quicksort vs. Radix as vary number keys: Cache misses

- Radix sort
- Quicksort (Cache misses/key)

What is proper approach to fast algorithms?
Summary #1 / 2:

° The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space

° Three (+1) Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Conflict Misses: increase cache size and/or associativity.
    Nightmare Scenario: ping pong effect!
  - Capacity Misses: increase cache size
  - Coherence Misses: Caused by external processors or I/O devices

° Cache Design Space
  - total size, block size, associativity
  - replacement policy
  - write-hit policy (write-through, write-back)
  - write-miss policy

Summary #2 / 2: 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