New-School Machine Structures
(It’s a bit more complicated!)

• Parallel Requests
  Assigned to computer
e.g., Search “Katz”
• Parallel Threads
  Assigned to core
e.g., Lookup, Ads
• Parallel Instructions
  >1 instruction @ one time
e.g., 5 pipelined instructions
• Parallel Data
  >1 data item @ one time
e.g., Add of 4 pairs of words
• Hardware descriptions
  All gates @ one time

Handling Cache Misses
(Single Word Blocks)

• Read misses (I$ and D$)
  – Stall execution, fetch the block from the next level in the memory hierarchy, install it in the cache, send requested word to processor, and then let execution resume
• Write misses (D$ only)
  – Write allocate: Stall execution, fetch the block from next level in the memory hierarchy, install it in cache, write the word from processor to cache, also update memory, then let execution resume
  or
  – No-write allocate: skip the cache write and just write the word to memory (but must invalidate cache block since it will now hold stale data)

Cache-Memory Consistency? (1/2)

• Need to make sure cache and memory have same value: 2 policies

  1) Write-Through Policy: write cache and write through the cache to memory
     – Every write eventually gets to memory
     – Too slow, so include Write Buffer to allow processor to continue once data in Buffer, Buffer updates memory in parallel to processor

Review

• Principle of Locality for Libraries /Computer Memory
• Hierarchy of Memories (speed/size/cost per bit) to Exploit Locality
• Cache – copy of data lower level in memory hierarchy
• Direct Mapped to find block in cache using Tag field and Valid bit for Hit
• Larger caches reduce Miss rate via Temporal and Spatial Locality, but can increase Hit time
• Larger blocks to reduce Miss rate via Spatial Locality, but increase Miss penalty
• AMAT (Average Memory Access Time) helps balance Hit rate, Miss rate, Miss penalty

Agenda

• Cache Hits and Misses, Consistency
• Administrivia
• Cache Performance and Size
• Technology Break
• Designing Memory Systems for Caches
• (If time permits – cache blocking with video!)
Cache-Memory Consistency? (2/2)

- Need to make sure cache and memory have same value: 2 policies

2) Write-Back Policy: write only to cache and then write cache block back to memory when evict block from cache
   - Writes collected in cache, only single write to memory per block
   - Include bit to see if wrote to block or not, and then only write back if bit is set
   - Called "Dirty" bit (writing makes it "dirty")

Average Memory Access Time (AMAT)

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

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

- How reduce Miss Penalty?

Sources of Cache Misses (3 C’s)

- **Compulsory** (cold start, first reference):
  - 1st access to a block, "cold" fact of life, not a lot you can do about it.
  - 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: increase cache size (may increase hit time)
  - Solution 2: (later in semester) increase associativity (may increase hit time)

Reducing Cache Miss Rates

- Use multiple $L$ levels
- With advancing technology, have more room on die for bigger $L1$ caches and for second level cache – normally a unified $L2$ cache (i.e., it holds both instructions and data), and in some cases even a unified $L3$ cache
- E.g., CPI$_{ideal}$ of 2,
  - 100 cycle miss penalty (to main memory),
  - 25 cycle miss penalty (to $L2$),
  - 36% load/stores,
  - a 2% (4%) $L1$ IS (DS) miss rate,
  - add a 0.5% $L2$ miss rate
  - \[ \text{CPI}_{peak} = 2 + 0.02 \times 25 + 0.36 \times 0.04 \times 25 + 0.005 \times 100 + 0.36 \times 0.05 \times 100 = 3.54 \] (vs. 5.44 with no $L2$)

Local vs. Global Miss Rates

- **Local miss rate** – the fraction of references to one level of a cache that miss
- **Global miss rate** – the fraction of references that miss in all levels of a multilevel cache
  - $L2$ local miss rate >> than the global miss rate
  - Global Miss rate = $L2$ Misses / Total Accesses
  - $L2$ Misses / $L1$ Misses / Total Misses / Total Accesses
  - Local Miss rate $L2$ x Local Miss rate $L1$
  - AMAT = Time for a hit + Miss rate x Miss penalty
  - AMAT = Time for a $L1$ hit + (local) Miss rate$L1x$ (Time for a $L2$ hit + (local) Miss rate$L2x$ Miss penalty)

Multilevel Cache Design Considerations

- Different design considerations for $L1$ and $L2$
  - $L1$ focuses on minimizing hit time for shorter clock cycle: Smaller $S$ with smaller block sizes
  - $L2$ focus on reducing miss rate to reduce penalty of long main memory access times: Larger $S$ with larger block sizes
  - Miss penalty of $L1$ is significantly reduced by presence of $L2$, so can be smaller/faster but with higher miss rate
  - For the $L2$, hit time is less important than miss rate
    - $L2$ hit time determines $L1$’s miss penalty
Agenda

• Cache Hits and Misses, Consistency
• Administrivia
• Cache Performance and Size
• Technology Break
• Memory Performance for Caches

Administrivia

• Lab #6 posted
• Project #2 Due Sunday @ 11:59:59
• No Homework this week!
• Midterm in less than 2 weeks:
  - Exam: Tu, Mar 8, 6-9 PM, 145/155 Dwinelle
  - Split: A-Lew in 145, Li-Z in 155
  - Covers everything through lecture March 3
  - Closed book, can bring one sheet notes, both sides
  - Copy of Green card will be supplied
  - No phones, calculators, ...; just bring pencils & eraser
  - TA Review: Su, Mar 6, 2-5 PM, 2050 VLSB

Getting to Know Profs:

Family

• Dad’s family Scotch-Irish, Mom’s family Swedish
• Grew up in Torrance, CA
• (Still) married to high school sweetheart
• 1st to graduate from college
• Liked it so much didn’t stop to PhD
• Spend 1 week/summer hosting Patterson Family Reunion

27 people: 2 parents, 3 siblings, 2 sons, 7 nieces and nephews, 7 spouses, 3 grandchildren, 1 grandnephew, 1 grandniece, 6 dogs …

Improving Cache Performance

(1 of 3)

1. Reduce the time to hit in the cache
   - Smaller cache
   - 1 word blocks (no multiplexor/selector to pick word)
2. Reduce the miss rate
   - Bigger cache
   - Larger blocks (16 to 64 bytes typical)
   - (Later in semester: More flexible placement by increasing associativity)

Improving Cache Performance

(2 of 3)

3. Reduce the miss penalty
   - Smaller blocks
   - Use multiple cache levels
     - L2 cache not tied to processor clock rate
   - 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 on read miss – may get lucky
   - Faster backing store/improved memory bandwidth
     - (Later in lecture)
The Cache Design Space
(3 of 3)

- Several interacting dimensions
  - Cache size
  - Block size
  - Write-through vs. write-back
  - Write allocation
  - (Later Associativity)
- Optimal choice is a compromise
  - Depends on access characteristics
    - Workload
    - Use (l-cache, d-cache)
  - Depends on technology / cost
- Simplicity often wins

Fields within an Address

<table>
<thead>
<tr>
<th>Tag&lt;32-n-(m+2)&gt;</th>
<th>Index&lt;n bits&gt;</th>
<th>Block offset&lt;m+n bits&gt;</th>
</tr>
</thead>
</table>

- 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+n bytes), m bits are used to address the word within the block and 2 bits are used to address the byte within the word: block offset
- Size of the tag field is
  - Address size – index size – block offset size
  - 32-bit byte address => 32 – n – (m+2)

Peer Instruction

- Assuming a direct-mapped, write-through cache with 16 KB of data and 4-word blocks, how divide a 32-bit byte address to access a cache?

  **A red** | Tag<14 bits> | Index<14 bits> | Block Offset<4 bits>
  **B orange** | Tag<16 bits> | Index<14 bits> | Block Offset<2 bits>
  **C green** | Tag<18 bits> | Index<10 bits> | Block Offset<4 bits>
  **D yellow** | Tag<20 bits> | Index<10 bits> | Block Offset<2 bits>
  **E pink** | Valid<1> | Tag<14 bits> | Index<14 bits> | Block Offset<4 bits>
  **F blue** | Valid<1> | Dirty<1> | Tag<14 bits> | Index<14 bits> | Block Offset<4 bits>
  **G purple** | Valid<1> | Tag<18 bits> | Index<10 bits> | Block Offset<4 bits>
  **H teal** | Valid<1> | Dirty<1> | Tag<18 bits> | Index<10 bits> | Block Offset<4 bits>

Cache Sizes

- Number of bits in a direct-mapped cache includes both the storage for data and for the tags + valid bit + dirty bit (if needed)
- Total number of bits in a cache is then
  - 2^n x (block size + tag field size + valid field size + dirty field size if needed)
- Why don’t need to store Block Offset in Cache? Why not Index in Cache?
Peer Instruction

- How many total bits are required for that cache? (Round to nearest Kbits)
  - Direct-mapped, write-through, 16 KBytes of data, 4-word (16 Byte) blocks, 32-bit address
  - Tag <18 bits> | Index <10 bits> | Block Offset <4 bits>

A red) 16 Kbits
B orange) 18 Kbits
C green) 128 Kbits
D yellow) 138 Kbits

Memory Systems that Support Caches

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

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

Assume

- 1 memory bus clock cycle to send address
- 15 memory bus clock cycles to get the 1st word in the block from DRAM (row cycle time), 5 memory bus clock cycles for 2nd, 3rd, 4th words (subsequent column access time)—note effect of latency!
- 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

Reading Miss Penalty

In the above figures, we have assumed that

- Each cache block is a single word
- Each memory block is a single word

However, in some cases, this assumption does not hold.

One Word Wide Bus, One Word Blocks

- If 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

- Total clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is $4/17 \times 0.235$ bytes per memory bus clock cycle

One Word Wide Bus, Four Word Blocks

- What if block size is four words and each word is in a different DRAM row?
  - 1 cycle to send 1st address cycle
  - 60 cycles to read DRAM
  - 1 cycle to return last data word

- Total clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is $(4 \times 62 \div 0.235)$ 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: 1 cycle
  2. Cycles to read DRAM banks: $15 + 3 \times 5 = 30$ cycles
  3. Cycles to return last data word: 4 cycles
  4. Total clock cycles miss penalty: $31$ cycles

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is $4 \times 4 / 32 = 0.5$ bytes per clock

---

**Interleaved Memory, One Word Wide Bus**

- For a block size of four words
  1. Cycle to send 1st address: 1 cycle
  2. Cycles to read DRAM banks: $15$
  3. Cycles to return last data word: $4 \times 4 = 4$ cycles
  4. Total clock cycles miss penalty: $20$ cycles

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is $4 \times 4 / 20 = 0.8$ bytes per clock

---

**DRAM Memory System Observations**

- It’s important to match the cache characteristics
  - Caches access one block at a time (usually more than one word)
  1. With the DRAM characteristics
     - Use DRAMs that support fast multiple word accesses, preferably ones that match the block size of the cache
  2. 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

---

**Performance Programming: Adjust software accesses to improve miss rate**

- Now that you understand how caches work, can revise program to improve cache utilization
  - Cache size
  - Block size
  - Multiple levels

---

**Performance of Loops and Arrays**

- Array performance often limited by memory speed
- OK if access memory different order as long as get correct result
- **Goal:** Increase performance by minimizing traffic from cache to memory
  - That is, reduce miss rate by getting better reuse of data already in cache
- One approach called **Cache Blocking:**
  - “shrink” problem by performing multiple iterations within smaller cache blocks
- Use Matrix Multiply as example: Next Lab and Project 3

---

**Matrix Multiplication**

$$c = a \times b$$
### Matrix Multiplication

Matrix \[ c = a \times b \]

\[ c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} \]

\[ \text{Simple Matrix Multiply - www.youtube.com/watch?v=yl0LTcDIhxc} \]

100 x 100 Matrix, Cache 1000 blocks, 1 word/block

### The simplest algorithm

Assumption: the matrices are stored as 2-D N x N arrays

- for \( i = 0 \) to \( N \) do
  - for \( j = 0 \) to \( N \) do
    - for \( k = 0 \) to \( N \) do
      - \[ c[i][j] += a[i][k] \times b[k][j]; \]

Advantage: code simplicity
Disadvantage: Marches through memory and caches

---

### Note on Matrix in Memory

- A matrix is a 2-D array of elements, but memory addresses are “1-D”
- Conventions for matrix layout:
  - by column, or “column major” (Fortran default): \( A[i][j] \) at \( A+i+jn \)
  - by row, or “row major” (C default): \( A[i][j] \) at \( A+i*n+j \)

### Improving reuse via Blocking: 1st “Naïve” Matrix Multiply

\[ \text{implements } C = C + A*B \]

- for \( i = 1 \) to \( n \) do
  - (read row of \( A \) into cache)
  - (read \( C[i][j] \) into cache)
  - (read column of \( B \) into cache)
  - for \( k = 1 \) to \( n \) do
    - \( C[i][j] += A[i][k] \times B[k][j] \)
  - (write \( C[i][j] \) back to main memory)

### Linear Algebra to the Rescue!

- Instead of multiplying two, say, 6 x 6 matrices

\[
\begin{align*}
A &= \begin{bmatrix}
  A_{00} & A_{01} & A_{02} \\
  A_{10} & A_{11} & A_{12} \\
  A_{20} & A_{21} & A_{22}
\end{bmatrix}, \\
B &= \begin{bmatrix}
  B_{00} & B_{01} & B_{02} \\
  B_{10} & B_{11} & B_{12} \\
  B_{20} & B_{21} & B_{22}
\end{bmatrix}
\end{align*}
\]

where \( A_{ij} = a_{i0}b_{0j} + a_{i1}b_{1j} + a_{i2}b_{2j} \) from this “blocked” representation, we can now calculate the matrix product as such:

\[
AB = \begin{bmatrix}
  (A_{00}B_{00} + A_{01}B_{10} + A_{02}B_{20}) & (A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{21}) & (A_{00}B_{02} + A_{01}B_{12} + A_{02}B_{22}) \\
  (A_{10}B_{00} + A_{11}B_{10} + A_{12}B_{20}) & (A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21}) & (A_{10}B_{02} + A_{11}B_{12} + A_{12}B_{22}) \\
  (A_{20}B_{00} + A_{21}B_{10} + A_{22}B_{20}) & (A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21}) & (A_{20}B_{02} + A_{21}B_{12} + A_{22}B_{22})
\end{bmatrix}
\]

- Thus, can get same result as multiplication of a set of submatrices

### Blocked Matrix Multiply

Consider \( A, B, C \) to be \( N \times N \) matrices of \( b \times b \) subblocks where \( b = n/N \) is called the block size

- for \( i = 1 \) to \( N \) do
  - (read block \( A[i][j] \) into cache)
  - (read block \( B[i][j] \) into cache)
  - for \( k = 1 \) to \( N \) do
    - (read block \( A[i][k] \) into cache)
    - (read block \( B[k][j] \) into cache)
    - \( C[i][j] = C[i][j] + A[i][k] \times B[k][j] \) (do a matrix multiply on blocks)
    - (write block \( C[i][j] \) back to main memory)

### Blocked Matrix Multiply

\[ \text{www.youtube.com/watch?v=IFWgwGMMrh0} \]

100 x 100 Matrix, 1000 cache blocks, 1 word/block, block 30x30
Another View of “Blocked” Matrix Multiplication

\[
C_{22} = A_{21}B_{12} + A_{22}B_{22} + A_{23}B_{32} + A_{24}B_{42} = \sum_k A_{2k}B_{k2}
\]

- **Main Point:** each multiplication operates on small “block” matrices, whose size may be chosen so that they fit in the cache.

Maximum Block Size

- The blocking optimization works only if the blocks fit in cache.
- That is, 3 blocks of size \( r \times r \) must fit in memory (for A, B, and C).
- \( M = \) size of cache (in elements/words)
- We must have: \( 3r^2 = M \) or \( r = \sqrt{M/3} \)
- Ratio of cache misses blocked vs. unblocked up to \( \approx \sqrt{M} \)

Simple Matrix Multiply Whole Thing: www.youtube.com/watch?v=53-stKx7wv

1x1 blocks: 1,020,000 misses: read A once, read B 100 times, read C once

Blocked Matrix Multiply Whole Thing. www.youtube.com/watch?v=tgmnX3xOrk

30x30 blocks: 90,000 misses = read A and B four times, read C once

“Only” 11X vs 30X Matrix small enough that row of A in simple version fits completely in cache; other things

Review

- To access cache, Memory Address divided into 3 fields: Tag, Index, Block Offset
- Cache size is Data + Management (tags, valid, dirty bits)
- Write misses trickier to implement than reads
  - Write back vs. Write through
  - Write allocate vs. No write allocate
- Cache Performance Equations:
  - CPU time = IC \times CPI \times CC
  = IC \times (CPI_{ideal} + Memory-stall cycles) \times CC
- AMAT = Time for a hit + Miss rate \times Miss penalty
- If you understand caches, can adapt software to improve cache performance and thus program performance