Admin

- Work on Projects!
This Unit: Caches

- Memory hierarchy concepts
- Cache organization
- High-performance techniques
- Low power techniques
- Some example calculations

Review

- ABCs of caches
- 3C’s

\[
\text{Ave Mem Acc Time} = \text{Hit time} + (\text{miss rate} \times \text{miss penalty})
\]

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

- Hardware methods for reducing misses
- Software methods for reducing misses
**Write Issues**

- So far we have looked at reading from cache (loads)
- What about writing into cache (stores)?

- Several new issues
  - Tag/data access
  - Write-through vs. write-back
  - Write-allocate vs. write-not-allocate

- Buffers
  - Store buffers (queues)
  - Write buffers
  - Writeback buffers

---

**Tag/Data Access**

- Reads: read tag and data in parallel
  - Tag mis-match → data is garbage (OK)

- Writes: read tag, write data in parallel?  
  - Tag mis-match → clobbered data (oops)
  - For associative cache, which way is written?

- Writes are a pipelined 2 cycle process
  - Cycle 1: match tag
  - Cycle 2: write to matching way
Tag/Data Access

- Cycle 1: check tag
  - Hit? Advance “store pipeline”
  - Miss? Stall “store pipeline”

- Cycle 2: write data

Advanced Technique
- Decouple write pipeline
- In the same cycle
  - Check tag of store_i
  - Write data of store_{i-1}
  - Bypass data of store_{i-1} to loads
Write-Through vs. Write-Back

- When to propagate new value to (lower level) memory?
  - **Write-through**: immediately
    + Conceptually simpler
    + Uniform latency on misses
    - Requires additional bus bandwidth
  - **Write-back**: when block is replaced
    - Requires additional “dirty” bit per block
    + Lower bus bandwidth for large caches
      - Only writeback dirty blocks
    - Non-uniform miss latency
      - Clean miss: one transaction with lower level (fill)
      - Dirty miss: two transactions (writeback + fill)
        - Writeback buffer: fill, then writeback (later)

- Common design: Write through L1, write-back L2/L3

Write-allocate vs. Write-non-allocate

- What to do on a write miss?
  - **Write-allocate**: read block from lower level, write value into it
    + Decreases read misses
    - Requires additional bandwidth
    - Used mostly with write-back
  - **Write-non-allocate**: just write to next level
    - Potentially more read misses
    + Uses less bandwidth
    - Used mostly with write-through

- Write allocate is common for write-back
  - Write-non-allocate for write through
Buffering Writes 1 of 3: Store Queues

• (1) Store queues
  ➢ Part of speculative processor; transparent to architecture
  ➢ Hold speculatively executed stores
  ➢ May rollback store if earlier exception occurs
  ➢ Used to track load/store dependences

• (2) Write buffers
• (3) Writeback buffers

Buffering Writes 2 of 3: Write Buffer

• (1) Store queues
• (2) Write buffers
  ➢ Holds committed architectural state
    ▪ Transparent to single thread
    ▪ May affect memory consistency model
  ➢ Hides latency of memory access or cache miss
  ➢ May bypass values to later loads (or stall)
  ➢ Store queue & write buffer may be in same physical structure
• (3) Writeback buffers
Buffering Writes 3 of 3: Writeback Buffer

- (1) Store queues
- (2) Write buffers

- (3) Writeback buffers (Special case of Victim Buffer)
  - Transparent to architecture
  - Holds victim block(s) so miss/prefetch can start immediately
  - (Logically part of cache for multiprocessor coherence)

Increasing Cache Bandwidth

- What if we want to access the cache twice per cycle?
- Option #1: multi-ported cache
  - Same number of six-transistor cells
  - Double the decoder logic, bitlines, wordlines
    - Areas becomes "wire dominated" -> slow
  - OR, time multiplex the wires
- Option #2: banked cache
  - Split cache into two smaller "banks"
  - Can do two parallel access to different parts of the cache
  - Bank conflict occurs when two requests access the same bank
- Option #3: replication
  - Make two copies (2x area overhead)
  - Writes both replicas (does not improve write bandwidth)
  - Independent reads
  - No bank conflicts, but lots of area
  - Split instruction/data caches is a special case of this approach
Multi-Port Caches

- Superscalar processors require multiple data references per cycle

- Time-multiplex a single port (double pump)
  - Need cache access to be faster than datapath clock
  - Not scalable

- Truly multiported SRAMs are possible, but
  - More chip area
  - Slower access
  - (Very undesirable for L1-D)

<table>
<thead>
<tr>
<th>Pipe 1</th>
<th>Pipe 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr</td>
<td>$</td>
</tr>
<tr>
<td>Pipe 2</td>
<td>Pipe 2</td>
</tr>
<tr>
<td>Addr</td>
<td>Data</td>
</tr>
</tbody>
</table>

Multi-Banking (Interleaving) Caches

- Address space is statically partitioned and assigned to different caches. Which addr bit to use for partitioning?

- A compromise (e.g. Intel P6, MIPS R10K)
  - Multiple references per cyc. if no conflict
  - Only one reference goes through if conflicts are detected
  - The rest are deferred (bad news for scheduling logic)

- Most helpful is compiler knows about the interleaving rules

| Even $ | Odd $ |
Multiple Cache Copies: e.g. Alpha 21164

- Independent fast load paths
- Single shared store path

- Not a scalable solution
  - Store is a bottleneck
  - Doubles area

Evaluation Methods

- The three system evaluation methodologies
  1. Analytic modeling
  2. Software simulation
  3. Hardware prototyping and measurement
Methods: Hardware Counters

- See Clark, TOCS 1983
  - accurate
  - realistic workloads, system + user + others
  - difficult, why?
  - must first have the machine
  - hard to vary cache parameters
  - experiments not deterministic
    - use statistics!
      - take multiple measurements
      - compute mean and confidence measures
- Most modern processors have built-in hardware counters

Methods: Analytic Models

- Mathematical expressions
  - insightful: can vary parameters
  - fast
  - absolute accuracy suspect for models with few parameters
  - hard to determine parameter values
  - difficult to evaluate cache interaction with system
  - bursty behavior hard to evaluate
Methods: Trace-Driven Simulation

- experiments repeatable
- can be accurate
- much recent progress
- reasonable traces are very large (gigabytes?)
- simulation is time consuming
- hard to say if traces are representative
- don't directly capture speculative execution
- don't model interaction with system

× Widely used in industry
Methods: Execution-Driven Simulation

- Simulate the program execution
  - simulates each instruction’s execution on the computer
  - model processor, memory hierarchy, peripherals, etc.
  - reports execution time
    - accounts for all system interactions
  - no need to generate/store trace
  - much more complicated simulation model
  - time-consuming but good programming can help
  - multi-threaded programs exhibit variability

✘ Very common in academia today

✘ Watch out for repeatability in multithreaded workloads

Low-Power Caches

- Caches consume significant power
  - 15% in Pentium4
  - 45% in StrongARM

- Three techniques
  - Way prediction (already talked about)
  - Dynamic resizing
  - Drowsy caches
Low-Power Access: Dynamic Resizing

- **Dynamic cache resizing**
  - Observation I: data, tag arrays implemented as many small arrays
  - Observation II: many programs don’t fully utilize caches

  - Idea: dynamically turn off unused arrays
    - Turn off means disconnect power \( V_{DD} \) plane
    - Helps with both dynamic and static power
  - There are always tradeoffs
    - Flush dirty lines before powering down \( \rightarrow \) costs power↑
    - Cache-size↓ \( \rightarrow \%_{\text{miss}} \uparrow \) \( \rightarrow \) power↑, execution time↑

Dynamic Resizing: When to Resize

- **Use \%_{\text{miss}} feedback**
  - \%_{\text{miss}} near zero? Make cache smaller (if possible)
  - \%_{\text{miss}} above some threshold? Make cache bigger (if possible)

- **Aside: how to track miss-rate in hardware?**
  - Hard, easier to track miss-rate vs. some threshold
  - Example: is \%_{\text{miss}} higher than 5%?
    - N-bit counter (\( N = 8 \), say)
    - Hit? counter \(-= 1\)
    - Miss? counter \(+ = 19\)
    - Counter positive? More than 1 miss per 19 hits (\%_{\text{miss}} > 5%)
Dynamic Resizing: How to Resize?

- **Reduce ways**
  - ["Selective Cache Ways", Albonesi, ISCA-98]
  - Resizing doesn’t change mapping of blocks to sets → simple
    - Lose associativity

- **Reduce sets**
  - ["Resizable Cache Design", Yang+, HPCA-02]
  - Resizing changes mapping of blocks to sets → tricky
    - When cache made bigger, need to relocate some blocks
    - Actually, just flush them
  - Why would anyone choose this way?
    - More flexibility: number of ways typically small
    - Lower $\%_{\text{miss}}$: for fixed capacity, higher associativity better

Drowsy Caches

- Circuit technique to reduce leakage power
  - Lower Vdd → Much lower leakage
  - But too low Vdd → Unreliable read/destructive read

- Key: Drowsy state (low Vdd) to hold value w/ low leakage
- Key: Wake up to normal state (high Vdd) to access
  - 1-3 cycle additional latency
Memory Hierarchy Design

- Important: design hierarchy components together
- **I$$, D$$**: optimized for latency$_{hit}$ and parallel access
  - Insns/data in separate caches (for bandwidth)
  - Capacity: 8–64KB, block size: 16–64B, associativity: 1–4
  - Power: parallel tag/data access, way prediction?
  - Bandwidth: banking or multi-porting/replication
  - Other: write-through or write-back
- **L2**: optimized for %$_{miss}$ power (latency$_{hit}$: 10–20)
  - Insns and data in one cache (for higher utilization, %$_{miss}$)
  - Power: parallel or serial tag/data access, banking
  - Bandwidth: banking
  - Other: write-back
- **L3**: starting to appear (latency$_{hit}$ = 30-50)

Hierarchy: Inclusion versus Exclusion

- **Inclusion**
  - A block in the L1 is always in the L2
  - Good for write-through L1s (why?)

- **Exclusion**
  - Block is either in L1 or L2 (never both)
  - Good if L2 is small relative to L1
    - Example: AMD's Duron 64KB L1s, 64KB L2

- **Non-inclusion**
  - No guarantees
Memory Performance Equation

- For memory component M
  - **Access**: read or write to M
  - **Hit**: desired data found in M
  - **Miss**: desired data not found in M
    - Must get from another (slower) component
  - **Fill**: action of placing data in M

- Performance metric
  - \( t_{\text{avg}} \): average access time
    \[ t_{\text{avg}} = t_{\text{hit}} + \%_{\text{miss}} \times t_{\text{miss}} \]

Hierarchy Performance

- \( t_{\text{avg}} \): average access time
  \[ t_{\text{avg}} = t_{\text{avg-M1}} \]
- \( t_{\text{avg-M1}} = t_{\text{hit-M1}} + (\%_{\text{miss-M1}} \times t_{\text{miss-M1}}) \)
- \( t_{\text{avg-M2}} = t_{\text{hit-M2}} + (\%_{\text{miss-M1}} \times t_{\text{avg-M2}}) \)
- \( t_{\text{avg-M3}} = t_{\text{hit-M3}} + (\%_{\text{miss-M1}} \times (t_{\text{hit-M2}} + (\%_{\text{miss-M2}} \times t_{\text{avg-M2}}))) \)
- \( t_{\text{avg-M4}} = t_{\text{hit-M4}} + (\%_{\text{miss-M1}} \times (t_{\text{hit-M3}} + (\%_{\text{miss-M2}} \times t_{\text{avg-M3}}))) \)
Local vs Global Miss Rates

- Local hit/miss rate:
  - Percent of references to cache hit (e.g., 90%)
  - Local miss rate is (100% - local hit rate), (e.g., 10%)

- Global hit/miss rate:
  - Misses per instruction (1 miss per 30 instructions)
  - Instructions per miss (3% of instructions miss)
  - Above assumes loads/stores are 1 in 3 instructions

- Consider second-level cache hit rate
  - L1: 2 misses per 100 instructions
  - L2: 1 miss per 100 instructions
  - L2 "local miss rate" -> 50%

Performance Calculation I

- Parameters
  - Reference stream: all loads
  - D$: $t_{hit} = 1\text{ns}$, $\%_{miss} = 5\%$
  - L2: $t_{hit} = 10\text{ns}$, $\%_{miss} = 20\%$
  - Main memory: $t_{hit} = 50\text{ns}$
- What is $t_{avgD}$ without an L2?
  - $t_{missD} = t_{hitM}$
  - $t_{avgD} = t_{hitD} + \%_{missD} * t_{hitM} = 1\text{ns} + (0.05 * 50\text{ns}) = 3.5\text{ns}$
- What is $t_{avgD}$ with an L2?
  - $t_{missD} = t_{avgL2}$
  - $t_{avgL2} = t_{hitL2} + \%_{missL2} * t_{hitM} = 10\text{ns} + (0.2 * 50\text{ns}) = 20\text{ns}$
  - $t_{avgD} = t_{hitD} + \%_{missD} * t_{avgL2} = 1\text{ns} + (0.05 * 20\text{ns}) = 2\text{ns}$
Performance Calculation II

• In a pipelined processor, I$%/D$ t\textsubscript{hit} is “built in” (effectively 0)

• Parameters
  - Base pipeline CPI = 1
  - Instruction mix: 30% loads/stores
  - I$: %\text{miss} = 2\%, t\text{miss} = 10 cycles
  - D$: %\text{miss} = 10\%, t\text{miss} = 10 cycles

• What is new CPI?
  - CPI\textsubscript{I$} = %\text{missI$}*t\text{miss} = 0.02*10 = 0.2 cycle
  - CPI\textsubscript{D$} = %\text{memory}*%\text{missD$}*t\text{missD$} = 0.30*0.10*10 = 0.3 cycle
  - CPI\textsubscript{new} = CPI + CPI\textsubscript{I$} + CPI\textsubscript{D$} = 1 + 0.2 + 0.3 = 1.5

An Energy Calculation

• Parameters
  - 2-way SA D$
  - 10\%$ miss rate
  - 5\mu W/access tag way, 10\mu W/access data way

• What is power/access of parallel tag/data design?
  - Parallel: each access reads both tag ways, both data ways
    - Misses write additional tag way, data way (for fill)
    - [2 * 5\mu W + 2 * 10\mu W] + [0.1 * (5\mu W + 10\mu W)] = 31.5 \mu W/access

• What is power/access of serial tag/data design?
  - Serial: each access reads both tag ways, one data way
    - Misses write additional tag way (actually...)
    - [2 * 5\mu W + 10\mu W] + [0.1 * 5\mu W] = 20.5 \mu W/access
Let’s talk about NUCA paper...