CS152 – Computer Architecture and Engineering
Lecture 13 – *Fastest Cache Ever!*

14 October 2003

Kurt Meinz
(www.eecs.berkeley.edu/~kurtm)

www-inst.eecs.berkeley.edu/~cs152/

Review

- SDRAM/SRAM
  - Clocks are good; handshaking is bad!
    - (From a latency perspective.)
- 4 Types of cache misses:
  - Compulsory
  - Capacity
  - Conflict
  - (Coherence)
- 4 Questions of cache design:
  - Placement
  - Re-placement
  - Identification (Sorta determined by placement…)
  - Write Strategy

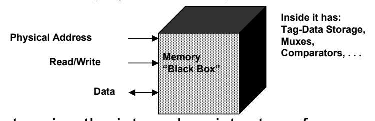
Recap: Measuring Cache Performance

- **CPU time** = Clock cycle time x
  (CPU execution clock cycles + Memory stall clock cycles)
  - Memory stall clock cycles =
    (Reads x Read miss rate x Read miss penalty +
    Writes x Write miss rate x Write miss penalty)
  - Memory stall clock cycles =
    Memory accesses x Miss rate x Miss penalty
- **AMAT** =
  Hit Time + (Miss Rate x Miss Penalty)

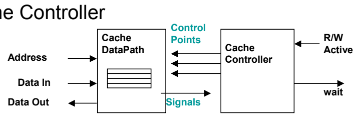
Note: *memory hit time is included in execution cycles.*

How Do you Design a Memory System?

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



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



Improving Cache Performance: 3 general options

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

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

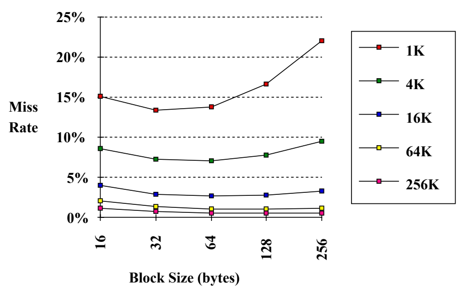
(Hit Rate x Hit Time) + (Miss Rate x Miss Time)

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.

1. Reduce Misses via Larger Block Size (61c)



2. Reduce Misses via Higher Associativity (61c)

- 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%
  - Example …

Example: Avg. Memory Access Time vs. Miss Rate

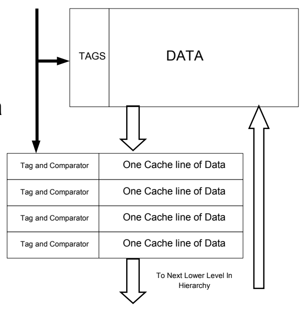
- Assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped

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

(Red means A.M.A.T. not improved by more associativity)

3. Reducing Misses via a "Victim Cache" (New!)

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines



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!!!

Improving Cache Performance (Continued)

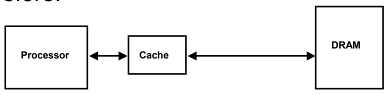
1. Reduce the miss rate,
2. *Reduce the miss penalty,* or
3. Reduce the time to hit in the cache.

0. Reducing Penalty: Faster DRAM / Interface

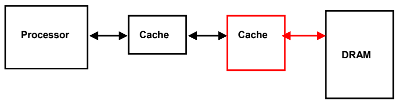
- New DRAM Technologies
  - Synchronous DRAM
  - Double Data Rate SDRAM
  - RAMBUS
    - same initial latency, but much higher bandwidth
- Better BUS interfaces
- CRAY Technique: only use SRAM!

1. Add a (lower) level in the Hierarchy

- Before:



- After:



2. 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*
  - *DRAM FOR LAB 5 can do this in burst mode! (Check out sequential timing)*
- 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

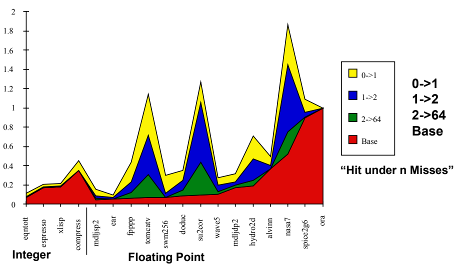
What happens on a Cache miss?

- 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 stalling 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



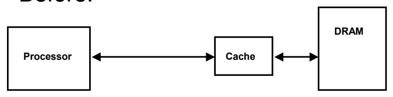
- FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26
- Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

Improving Cache Performance (Continued)

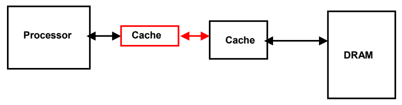
1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. *Reduce the time to hit in the cache*.

1. Add a (higher) level in the Hierarchy (61c)

- Before:



- After:



2: Pipelining the Cache! (new!)

- Cache accesses now take multiple clocks:
  - 1 to start the access,
  - X (> 0) to finish
  - PIII uses 2 stages; PIV takes 4
  - Increases hit bandwidth, not latency!

3: Way Prediction (new!)

- Remember: Associativity negatively impacts hit time.
- We can recover some of that time by pre-selecting one of the sets.
  - Every block in the cache has a field that says which index in the set to try on the ***next*** access. Pre-select mux to that field.
  - Guess right: Avoid mux propagate time
  - Guess wrong: Recover and choose other index
    - Costs you a cycle or two.

3: Way Prediction (new!)

- Does it work?
  - You can guess and be right 50%
  - Intelligent algorithms can be right ~85%
  - Must be able to recover quickly!
  - On Alpha 21264:
    - Guess right: ICache latency 1 cycle
    - Guess wrong: ICache latency 3 cycles
    - (Presumably, without way-predict would require push clock period or #cycles/hit.)

PRS: Load Prediction (new!)

- Load-Value Prediction:
  - Small table of recent load instruction addresses, resulting data values, and confidence indicators.
  - On a load, look in the table. If a value exists and the confidence is high enough, use that value. Meanwhile, do the cache access …
    - If the guess was **correct**: increase confidence bit and keep going
    - If the guess was **incorrect**: quash the pipe and restart with correct value.

PRS: Load Prediction

- So, will it work?
  - If so, what factor will it improve
  - If not, why not?

1. **No way! – There is no such thing as data locality!**
2. **No way! – Load-value mispredictions are too expensive!**
3. **Oh yeah! – Load prediction will decrease hit time**
4. **Oh yeah! – Load prediction will decrease the miss penalty**
5. **Oh yeah! – Load prediction will decrease miss rates**

6) 1 and 2 7) 3 and 4 8) 4 and 5 9) 3 and 5 10) None!

Memory Summary (1/3)

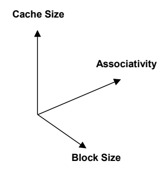
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.
- SRAM is fast but expensive and not very dense:
  - 6-Transistor cell (no static current) or 4-Transistor cell (static current)
  - Does not need to be refreshed
  - Good choice for providing the user FAST access time.
  - Typically used for CACHE
- DRAM is slow but cheap and dense:
  - 1-Transistor cell (+ trench capacitor)
  - Must be refreshed
  - Good choice for presenting the user with a BIG memory system
  - Both asynchronous and synchronous versions
  - Limited signal requires “sense-amplifiers” to recover

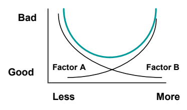
Memory Summary 2/ 3:

- 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 3 / 3: 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


Cache Size
Associativity
Block Size


Bad
Good
Factor A
Factor B
Less
More