Memory Subsystem Design

or

Nothing Beats Cold, Hard Cache

Memory Locality

• Memory hierarchies take advantage of memory locality.

• Memory locality is the principle that future memory accesses are near past accesses.

• Memories take advantage of two types of locality
  – Temporal locality -- near in _____ => we will often access the same data again very soon
  – Spatial locality -- near in __________ => our next access is often very close to our last access (or recent accesses).

(this sequence of addresses exhibits both temporal and spatial locality)
1,2,3,1,2,3,8,8,47,9,10,8,8...

Locality and cacheing

• Memory hierarchies exploit locality by caching (keeping close to the processor) data likely to be used again.

• This is done because we can build large, slow memories and small, fast memories, but we can’t build large, fast memories.

• If it works, we get the illusion of SRAM access time with disk capacity

SRAM access times are 2 - 25ns at cost of $100 to $250 per Mbyte.
DRAM access times are 60-120ns at cost of $5 to $10 per Mbyte.
Disk access times are 10 to 20 million ns at cost of $.10 to $.20 per Mbyte.

A typical memory hierarchy
Cache Fundamentals

- **cache hit** -- an access where the data is found in the cache.
- **cache miss** -- an access which isn’t
- **hit time** -- time to access the cache
- **miss penalty** -- time to move data from further level to closer, then to cpu
- **hit ratio** -- percentage of time the data is found in the cache
- **miss ratio** -- (1 - hit ratio)

Cache Fundamentals, cont.

- **cache block size or cache line size** -- the amount of data that gets transferred on a cache miss.
- **instruction cache** -- cache that only holds instructions.
- **data cache** -- cache that only caches data.
- **unified cache** -- cache that holds both.

Cacheing Issues

On a memory access -
- How do I know if this is a hit or miss?

On a cache miss -
- where to put the new data?
- what data to throw out?
- how to remember what data this is?

A simple cache

- A cache that can put a line of data anywhere is called fully associative
- The most popular replacement strategy is **LRU** (least recently used).
A simpler cache

- A cache that can put a line of data in exactly one place is called ____________.
- Advantages/disadvantages vs. fully-associative.

A set-associative cache

- A cache that can put a line of data in exactly \( n \) places is called \( n \)-way set-associative.
- The cache lines that share the same index are a cache ____.

Longer Cache Blocks

- Large cache blocks take advantage of spatial locality.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space.

Block Size and Miss Rate

- The graph shows the miss rate for different block sizes. Larger block sizes have lower miss rates, indicating better performance.
**Cache Parameters**

Cache size = Number of sets * block size * associativity

- 128 blocks, 32-byte block size, direct mapped, size = ?
- 128 KB cache, 64-byte blocks, 512 sets, associativity = ?

**Handling a Cache Access**

1. Use index and tag to access cache and determine hit/miss.
2. If hit, return requested data.
3. If miss, select a cache block to be replaced, and access memory or next lower cache (possibly stalling the processor).
   - Load entire missed cache line into cache
   - Return requested data to CPU (or higher cache)
4. If next lower memory is a cache, goto step 1 for that cache.

**Accessing a Sample Cache**

- 64 KB cache, direct-mapped, 32-byte cache block size

31 30 29 28 27 .......... 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

**Accessing a Sample Cache**

- 32 KB cache, 2-way set-associative, 16-byte block size

31 30 29 28 27 .......... 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Associative Caches

• Higher ___________, but...

• longer ___________ (longer to determine hit/miss, more muxing of outputs)

• more space (longer tags)
  – 16 KB, 16-byte blocks, dm, tag = ?
  – 16 KB, 16-byte blocks, 4-way, tag = ?

Dealing with Stores

• Stores must be handled differently than loads, because...
  – they don’t necessarily require the CPU to stall.
  – they change the content of cache/memory (creating memory consistency issues)
  – may require a load and a store to complete

Policy decisions for stores

• Keep memory and cache identical?
  – write-through => all writes go to both cache and main memory
  – write-back => writes go only to cache. Modified cache lines are written back to memory when the line is replaced.

• Make room in cache for store miss?
  – write-allocate => on a store miss, bring written line into the cache
  – write-around => on a store miss, ignore cache

Dealing with stores

• On a store hit, write the new data to cache. In a write-through cache, write the data immediately to memory. In a ___________ cache, mark the line as dirty.

• On a store miss, initiate a cache block load from memory for a write-allocate cache. Write directly to memory for a ___________ cache.

• On any kind of cache miss in a write-back cache, if the line to be replaced in the cache is dirty, write it back to memory.
Cache Performance

TCPI = BCPI + MCPI
- where TCPI = Total CPI;
- BCPI = base CPI, which means the CPI assuming perfect memory (BCPI = peak CPI + PSPI + BSPI)
  - PSPI => pipeline stalls per instruction
  - BSPI => branch hazard stalls per instruction
- MCPI = the memory CPI, the number of cycles (per instruction) the processor is stalled waiting for memory.

MCPI = accesses/instruction * miss rate * miss penalty
- this assumes we stall the pipeline on both read and write misses, that the miss penalty is the same for both, that cache hits require no stalls.
- If the miss penalty or miss rate is different for Inst cache and data cache (common case), then
MCPI = IS accesses/inst*ISMR*ISMP + DS acc/inst*DSMR*DSMP

Example -- DEC Alpha 21164 Caches

- ICache and DCache -- 8 KB, DM, 32-byte lines
- L2 cache -- 96 KB, 8-way SA, 32-byte lines
- L3 cache -- 1 MB, DM, 32-byte lines

Cache Performance

- Instruction cache miss rate of 4%, data cache miss rate of 9%, BCPI = 1.0, 20% of instructions are loads and stores, miss penalty = 12 cycles, TCPI = ?
- Unified cache, 25% of instructions are loads and stores, BCPI = 1.2, miss penalty of 10 cycles. If we improve the miss rate from 10% to 4% (e.g. with a larger cache), how much do we improve performance?
- BCPI = 1, miss rate of 8% overall, 20% loads, miss penalty 20 cycles, never stalls on stores. What is the speedup from doubling the cpu clock rate?

Example -- DEC Alpha 21164 Caches

Cache Alignment

- The data that gets moved into the cache on a miss are all data whose addresses share the same tag and index (regardless of which data gets accessed first).
- This results in
  - no overlap of cache lines
  - easy mapping of addresses to cache lines (no additions)
  - data at address X always being present in the same location in the cache block (at byte X mod blocksize) if it is there at all.
- Think of memory as organized into cache-line sized pieces (because in reality, it is!).

Example -- DEC Alpha 21164 Caches
Three types of cache misses

- Compulsory (or cold-start) misses
  - ______ access to the data.
- Capacity misses
  - we missed only because the cache isn’t _____ enough.
- Conflict misses
  - we missed because the data maps to the same line as other data that forced it out of the cache.

Three types of cache misses

- Compulsory (or cold-start) misses
  - ______ access to the data.
- Capacity misses
  - we missed only because the cache isn’t _____ enough.
- Conflict misses
  - we missed because the data maps to the same line as other data that forced it out of the cache.

Cache Miss Components

- Compulsory misses?
- Capacity misses?
- Conflict misses?
LRU replacement algorithms

- only needed for associative caches
- requires one bit for 2-way set-associative, 8 bits for 4-way, 24 bits for 8-way.
- can be emulated with $\log n$ bits (NMRU)
- can be emulated with use bits for highly associative caches (like page tables)
- However, for most caches (eg, associativity $\leq 8$), LRU is calculated exactly.

Caches in Current Processors

- Often DM at highest level (closest to CPU), associative further away (this is less true today).
- split I and D close to the processor (for throughput rather than miss rate), unified further away.
- write-through and write-back both common, but never write-through all the way to memory.
- 32-byte cache lines common (but getting larger – 64, 128)

- Non-blocking
  - processor doesn’t stall on a miss, but only on the use of a miss (if even then)
  - this means the cache must be able to handle multiple outstanding accesses.

Key Points

- Caches give illusion of a large, cheap memory with the access time of a fast, expensive memory.
- Caches take advantage of memory locality, specifically temporal locality and spatial locality.
- Cache design presents many options (block size, cache size, associativity, write policy) that an architect must combine to minimize miss rate and access time to maximize performance.