Arch: Memory Hierarchy

Alex Chi

Update: May 14, 2020

Contents

1 Cache Organization **3**

1.1 Main Memory . . . 3

1.2 SRAM and DRAM . . . 3

1.3 Locality . . . 3

1.4 Terminology . . . 4

1.5 Management . . . 4

1.5.1 Four Main Questions . . . 4

1.5.2 Block Placement . . . 4

1.5.3 Block Identification . . . 4

1.5.4 Block Replacement . . . 5

1.5.5 Write Strategy . . . 5

1.5.6 Cache Hits . . . 5

1.5.7 Cache Miss . . . 5

1.5.8 Multiword Cache Miss . . . 5

1.5.9 Measuring Performance . . . 6

2 Virtual Memory **6**

2.1 Why? . . . 6
2.2 Address Translation . . . 6
2.2.1 Mechanisms . . . 6

3 Six Basic Cache Optimizations 7

3.1 General Technique . . . 7
3.2 Categories of Misses . . . 8
3.3 Larger Block Size . . . 8
3.4 Larger Cache . . . 8
3.5 Higher Associativity . . . 8
3.6 Multi-level Cache . . . 8
3.7 Priority to Read Miss . . . 9
3.8 Avoid Address Translation . . . 9
3.9 Summary . . . 9

4 Ten Advanced Optimizations 10

4.1 Intro . . . 10
4.2 Small and Simple First Level Cache . . . 10
4.3 Way Prediction . . . 10
4.4 Pipelining Cache . . . 11
4.5 Non-blocking Cache . . . 11
4.6 Multi-banked Cache . . . 11
4.7 Critical Word First, Early Restart . . . 11
4.8 Merging Write Buffer . . . 11
4.9 Compiler Optimizations . . . 12
4.10 Hardware Prefetching . . . 12
4.11 Compiler Prefetching . . . 12

5 Memory Technology and Optimizations **12**

5.1 Metrics 12

5.2 Optimization on DRAM Technology 12

5.3 DDR SDRAM 13

5.4 Graphics Memory 13

5.5 Memory Dependability 13

5.6 Flash Memory 13

5.7 New Trends: NVM 13

1 Cache Organization

1.1 Main Memory

- large linear array of bytes, each byte has an address
- some arch only support aligned access (32-bit boundary)
- issues with main memory: too slow
- hierarchy: Register File - ITLB/DTLB - ICache/DCache - SRAM (second level cache) - DRAM (main memory) - secondary memory (storage)
- access time: fast to slow
- size: small to big
- cost: high to low

1.2 SRAM and DRAM

- SRAM: fast, low density, static (6 transistors, data last forever)
- DRAM: slow, high density, dynamic (need to refresh regularly, 1 transistor)
  - row access strobe or column access strobe

1.3 Locality

- temporal locality / locality in time: recently referenced items are likely to be referenced
- spatial locality / locality in space: items in nearby address tend to be referenced

1.4 Terminology

- Block: minimum unit of information in cache
- Hit Rate: memory access found in cache
- Miss Rate: memory access not found in cache (1 - Hit Rate)
- Hit Time: Time to access block + Time to determine hit/miss
- Miss Penalty: Time to replace block with corresponding block from lower level, access lower level + transmit block + insert block + return to request

1.5 Management

- Place variable in register or memory? Compiler
- When to tramsfer block from main memory to cache? Cache controller
- When to put data from memory to disk? Operating System, Virtual Memory (+ TLB)

1.5.1 Four Main Questions

- Block placement
- Block identification
- Block replacement
- Write strategy

1.5.2 Block Placement

- Fully associative (anywhere in cache)
- Direct mapped (like hash table)
- Set associative (direct mapped + set)

1.5.3 Block Identification

- addr = | Block Address | Block Offset |
- Block addr = | Tag | Index |
- Address tag on each block
- Valid bit on each block

1. Direct Mapped
    - block addr mod N

    - If tag of block == tag of addr, then found
2. Set Associative Mapped
    - blocks of same index are put in a set
    - Multiple blocks in a set, identified by tag
    - N-way set-associative cache
    - Full-associative cache is set-associative of 1 set

1.5.4 Block Replacement

- FIFO / LRU / Random

1.5.5 Write Strategy

- Write-through: information written to cache and main memory
- Write-back: write information to block, delayed write to main memory when being replaced (crux: data inconsistence)

1.5.6 Cache Hits

- Read hits (I/D), no further operation
- Write hits (D), two ways
    - cache and memory are consistent: write-through or write-through + write buffer
    - cache and memory are inconsistent
        - write-back, need dirty bit for cache
        - Use write buffer to accelerate

1.5.7 Cache Miss

- Read miss (I/D), stall pipeline, fetch from memory
- Write miss
    - write allocate: stall pipeline, fetch from next level
    - No write allocate: invalidate cache, put to write buffer

1.5.8 Multiword Cache Miss

- Read miss (I/D)
    - early restart: wait whole block

    - Requested word first: ???
- Write miss (D)
    - write allocate: fetch block from memory, then write to block

1.5.9 Measuring Performance

- Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty
- CPU Time = (CPU Ex Clock Cycles + Mem Stall Cycles) * cycle time
- CPU Time55212 = IC * (CPI + mem/all * Miss Rate * Miss Penalty) * Cycle Time

2 Virtual Memory

- Requires integration between hardware and software.
- TLB + page table translates virtual address to physical address.
- OS places page from disk to memory

2.1 Why?

- easily run programs larger than physical memory
- simplify code loading (all executable place code segment at same place)
- allow efficient and safe sharing of memory (copy-on-write, etc.)
- locality makes it work: a program access small portion of address space (in a page)

2.2 Address Translation

- `VA = | VPN | Offset |` $\rightarrow$ `PA = | PPN | Offset |`
- Memory request requires translation from VA to PA
- If page is not in physical memory, it's a page fault

2.2.1 Mechanisms

- page table: a table reside in physical memory
- TLB: a cache for translating VA to PA
- handle a memory request
    - VA in TLB? If yes, jump to 3, otherwise this is a TLB miss

    - fetch page table from main memory, load to TLB (software-managed / hardware-managed)
    - data in cache? get data if yes, otherwise jump to 4
    - fetch data from memory
- TLB miss: 10 cycles to load from memory, 100000 cycles to load from disk
- TLB misses are much more frequent than true page fault

Table 1: TLB event combination

TLB	Page Table	Cache	Circumstance
H	H	H	Y, for most case
H	H	M	Y, know PA but data in PA not in cache
M	H	H	Y, TLB miss, but we can recover quickly
M	H	M	Y, TLB miss + PA not in cache
M	M	M	Y, true page fault
H	M	M/H	N, if in TLB, must in page table
M	M	H	N, if not in page table, then not in memory

- Note: we only consider consistent state. For example, it's possible that we switch from kernel space to user space without invalidating the cache. In this way, it's possible that TLB hit (but contain wrong data), and page table + cache miss.

3 Six Basic Cache Optimizations

3.1 General Technique

- Average Mem Access Time = Hit + Miss * Miss Penalty
    - reduce miss rate
        - larger block size
        - larger cache size
        - higher associativity
    - reduce miss penalty
        - multi-level cache
        - read priority
    - reduce time to hit
        - avoid address translation when indexing cache

3.2 Categories of Misses

- compulsory: very first access cannot be served
- capacity: cache is too small
- conflict: data mapped to same set
- distribution of miss rate: refer to CA:AQA

3.3 Larger Block Size

- ↓ compulsory miss (spatial locality)
- ↑ miss penalty
- ↑ conflict miss
- all: ↓ miss rage

3.4 Larger Cache

- ↓ capacity miss
- ↑ hit time (latency), cost, power
- popular in off-chip caches
- all: ↓ miss rate

3.5 Higher Associativity

- rules of thumb
  - 8-way set associative, as effective as fully-associative
  - 2:1, direct-mapped cache of size N has same miss rate as 2-way N/2
- ↓ miss rate

3.6 Multi-level Cache

- ↓ penalty
- L1, L2, L3 cache
- Average Memory Access Time = $H_1 + M_1 * P_1$, $P_1 = H_2 + M_2 * P_2$
- Local Miss Rate: M_1, M_2
- Global Miss Rate: M_1, $M_1 * M_2$

3.7 Priority to Read Miss

- serve reads before write completes
- with write buffer (write-through cache)
- ↓ penalty
- example

```
SW R3, 512(R0)  # flush cache
LW R1, 1024(R0) # read miss
LW R2, 512(R0)  # read miss (RAW, if not check write buffer, will get wrong data!)
```

- solution for read miss + write buffer
  - wait until write buffer is empty (wait until cache is consistent)
  - check contents in write buffer (priority to read miss)

3.8 Avoid Address Translation

- ↓ hit time
- why not use virtual address in cache?
  - "virtual caches, with physical cache"
  - protection: even if data is in cache, process may have no permission. solution: copy protection bits from TLB
  - process switching: flush cache every time we switch process
  - aliases: multiple virtual page mapped to same PA, cause coherence issue
- overlapping address translation and cache reading
  - "virtually indexed, physically tagged"
  - `| VPN | Offset |`
  - `| Tag | Index | Block Offset |`
  - use page offset to index the cache
  - resolve PA with TLB
  - simultaneously fetch data from cache with index
  - compare cache tag and PA tag
  - hence, cache size limited to page size * associativity

3.9 Summary

Table 2: Impact of Cache Performance and Complexity

Technique	Hit time	Miss penalty	Miss rate	Hardware complexity
Larger block size		-	+	0
Larger cache size	-		+	1
Higher associativity	-		+	1
Multi-level caches		+		2
Read priority over writes		+		1
Avoiding address translation during cache indexing	+			1

4 Ten Advanced Optimizations

4.1 Intro

- we have already considered
  - miss rate
  - miss penalty
  - hit time
- and we will consider more
  - cache bandwidth
  - power consumption

4.2 Small and Simple First Level Cache

- reduce hit time
- higher associativity in L1 cache
  - processors nowadays take at least 2 clock cycle to access cache
  - keep TLB out of critical path, L1 virtually indexed + physically tagged
  - higher associativity reduces conflict misses
- cache size limited in v-indexed + p-tagged
  - page size $\times$ associativity

4.3 Way Prediction

- improve hit time
- reduce conflict miss

- accuracy: 90% for 2-way, 80% for 40way, I-cache better
- MIPS R10000, mid-90s + ARM Cortex-A8

4.4 Pipelining Cache

- improve cache bandwidth
- increase branch-mispredict penalty
- easy to increase associativity

4.5 Non-blocking Cache

- allow hit before previous miss complete
- miss penalty reduced
- out-of-order execution allows data miss not to block

4.6 Multi-banked Cache

- organize cache as independent banks, support simultaneous access
- work best when access naturally spread across banks

4.7 Critical Word First, Early Restart

- background: cache line normally contains more than a word
- critical word first
  - request missed work from memory first
  - send to processor as soon as arrival
- early restart
  - request work in normal order
  - send missed word to processor as soon as arrival
- reduce miss penalty
- effect depends on block size + access to same block

4.8 Merging Write Buffer

- write to same cache line previously requires separate write buffer
- now we can write to pending write buffer

4.9 Compiler Optimizations

- column-access first
- reduce miss rate

4.10 Hardware Prefetching

- processor typically prefetch two blocks
  - reduce penalty, miss rate
  - utilize memory bandwidth

4.11 Compiler Prefetching

- use instruction to hint processor, that to prefetch some data
- insert prefetch instructions
- register prefetch
- cache prefetch

5 Memory Technology and Optimizations

5.1 Metrics

- latency
  - access time
  - cycle time
- bandwidth

5.2 Optimization on DRAM Technology

- multiple access to same row
- synchronous DRAM
- wider interface
- double data rate (DDR)
- multiple banks on each DRAM device

5.3 DDR SDRAM

- DDR2, DDR3, DDR4

5.4 Graphics Memory

- 2-5x bandwidth per DRAM vs DDR3
- GDDR5 is based on DDR3

5.5 Memory Dependability

- susceptible to cosmic rays
- soft error: fixed by ECC
- hard error: use spare rows

5.6 Flash Memory

- type of EEPROM
- can hold data without power
- must be erase before overwritten
- write may affect adjacent blocks
- limited number of write cycles
- cost: disk < flash < SDRAM
- speed: SDRAM > flash > disk

5.7 New Trends: NVM

- FeRAM, PCM, MRAM
- FaRM: a distributed key-value store based on RDMA + NVRAM