Chapter 6 Part A: Caches

Basic Caches

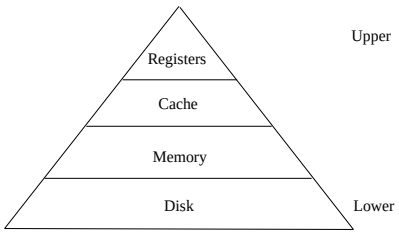
- Introduction
- Four Questions
- Cache Size, Block Size, Associativity
- Writes

Advanced Caches

Memory Hierarchy

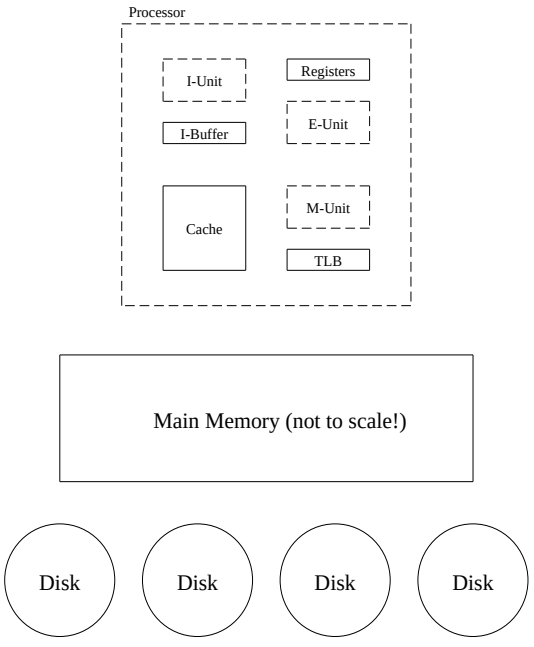
Make common case fast

- Common: temporal & spatial locality
- Fast: smaller, more expesive memory

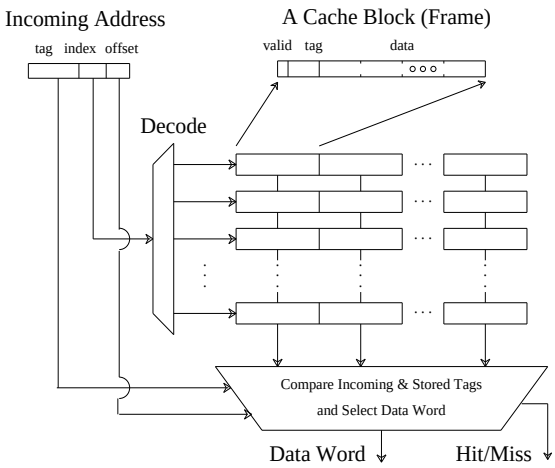


Type	Size	Speed (ns)	Bandwidth (MB/s)
Registers	<1KB	10	800
Cache	<512KB	20	200
Memory	<512MB	100	133
Disk	>1GB	20,000,000	4

Simple Memory Hierarchy



Example Cache



Memory Hierarchy Terminology

Block (line, page)

- Minimum unit that may be present
- Usually fixed length

Hit -- Block is found in upper level

Miss -- Not found in upper level

Miss ratio -- Fraction of references that miss

Hit Time -- Time to access the upper level

Miss Penalty

- Time to replace block in upper level, plus the time to deliver the block to the CPU.
- Access time -- Time to get 1st word
- Transfer time -- Time for remaining words

Memory Hierarchy Performance

Time is *always* the ultimate measure

Indirect measures can be misleading

- MIPS can be misleading
- **So can Miss ratio**

Average (effective) access time is better

$$t_{avg} = t_{hit} + \text{miss ratio} \times t_{miss}$$

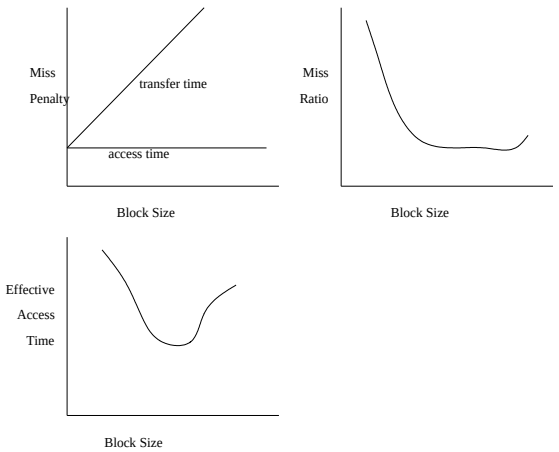
Example:

$t_{hit} = 1$
$t_{miss} = 20$
missratio = .05

$$t_{avg} = 1 + .05 \times 20 = 2$$

Effective access time is *still* an indirect measure

Performance Fallacy



Miss ratio does not take block size into account

Effective access time does

Fundamental Cache Questions

Block placement

- Where can a block be placed in the cache?

Block Identification

- How is a block found in the cache?

Block replacement

- Which block should be replaced on a miss?

Write strategy (skip for now)

- What happens on a write?

Cache Type

- What type of information is stored in the cache?

Block Placement

Fully-Associative

Block goes in any block frame

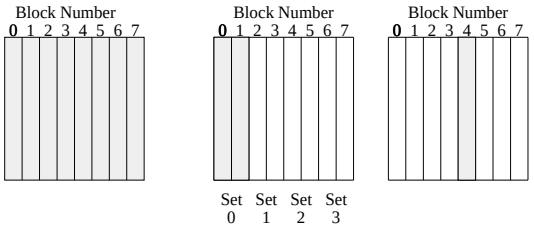
Direct-mapped

Block goes in exactly one block frame

Set-Associative

Block goes in exactly one *set*.

Example: Where does block 12 go?



1) Fully Associative 2) Set Associative 3) Direct Mapped

Block Identification

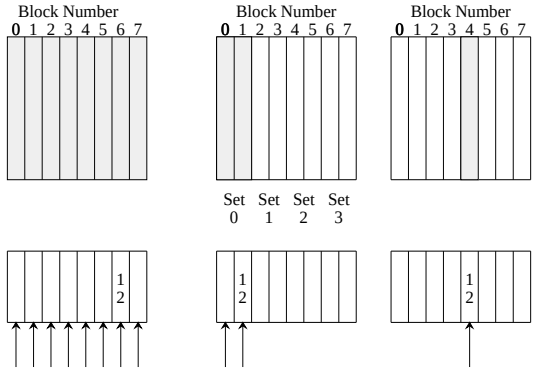
How to find the block.

Tag comparisons

Parallel search to speed lookup

Check *valid bit*

Example: Where do we search for block 12?



1) Fully Associative 2) Set Associative 3) Direct Mapped

Block Replacement

Which block to replace on a miss?

Least-recently used (LRU)

Optimize based on temporal locality

Replace block unused for longest time

State updates on non-MRU misses

Random

Select victim at random

Pseudo-random for hardware testing

Nearly as good as LRU, and easier

First-in First-out (FIFO)

Replace block loaded first

Optimal

Replace block used furthest in time

Cache Type

Unified (mixed)

Less costly

Dynamic response

Handles writes into I-stream

Separate Instruction & Data (split, Harvard)

2x bandwidth

Place closer to I and D ports

Can customize

Poor-man's associativity

No interlocks on simultaneous requests

Caches should be split if simultaneous instruction and data accesses are frequent (e.g., RISCs)

Cache Type Example

Consider building (a)16K byte I & D caches, or (b) a 32K byte unified cache.

Let t_{cache} is one cycle, t_{memory} is 10 cycles.

(a) ***Imiss*** is 5 percent, ***Dmiss*** is 6 percent, 75 percent of references are instruction fetches.

$$t_{avg} = (1 + 0.05 \times 10) \times 0.75 + (1 + 0.06 \times 10) \times 0.25 = 1.5$$

(b) ***miss*** is 4 percent

$$t_{avg} = 1 + 0.04 \times 10 = 1.4 \text{ WRONG!}$$

t_{avg} = 1.4 + cycles-lost-to-interference

Will cycles-lost-to-interference < 0.1?

Not for RISC machines!

A Miss Classification (3Cs or 4Cs)

Cache misses can be classified as:

Compulsory

The first access to a block

Capacity

Misses that occur when a replaced block is rereferenced.

Conflict

Misses that occur because blocks are discarded because of the set-mapping strategy.

Coherence (shared-memory multiprocessors)

Misses that occur because blocks are invalidated due to references by other processors.

Fundamental Cache Parameters

Cache Size

How large should the cache be?

Block Size

What is the smallest unit represented in the cache?

Associativity

How many entries must be searched for a given address?

Cache Size

Cache size is the total capacity of the cache

Bigger caches exploit temporal locality better than smaller caches

But are *not always better*

Too large a cache size

Smaller means faster ==> bigger means slower

Access time may degrade critical path

Too small a cache size

Don't exploit temporal locality well

Useful data is prematurely replaced

Block Size

Block (line) size is the data size that is both

(a) associated with an address tag, and

(b) transferred to/from memory

Advanced caches allow different (a) & (b)

Too small blocks

Don't exploit spatial locality well

Have inordinate address tag overhead

Too large blocks cause

Unused data to be transferred

Useful data to be prematurely replaced

Associativity

Partition cache blocks frames into

s equivalence classes (number of sets)

of **n** block frames each (associativity)

Typical values for associativity

1 -- direct-mapped

n = 2, 4, 8, 16 -- n-way set-associative

All blocks -- fully-associative

Larger associativities

Lower miss ratios & less variation

Intuitively satisfying

Smaller associativities

Lower cost

Faster access (hit) time (perhaps)

An Implementation Effect Case Study

Associativity that minimizes t_{avg} is often smaller than associativity that minimizes miss ratio!

Consider DM & SA caches w/ same t_{memory}.

$\Delta t_{cache} = t_{cache}(SA) - t_{cache}(DM) > 0$

$\Delta miss = miss(SA) - miss(DM) < 0$

$t_{avg}(SA) < t_{avg}(DM)$ only if

$t_{cache}(SA) + miss(SA) \times t_{memory} <$

$t_{cache}(DM) + miss(DM) \times t_{memory}$

$\Delta t_{cache} + \Delta miss \times t_{memory} < 0$

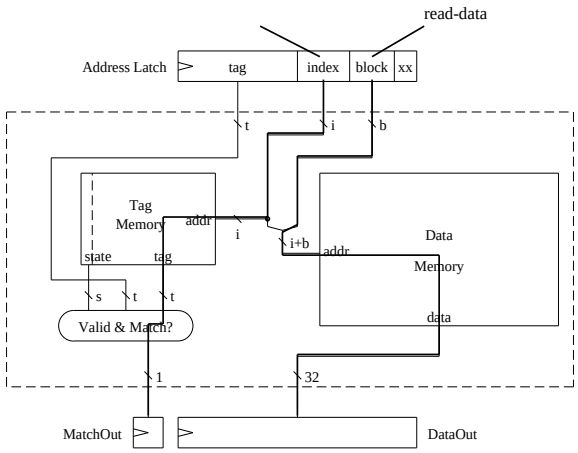
E.g.s,

(a) Assuming $\Delta t_{cache} = 0$ ==> SA better

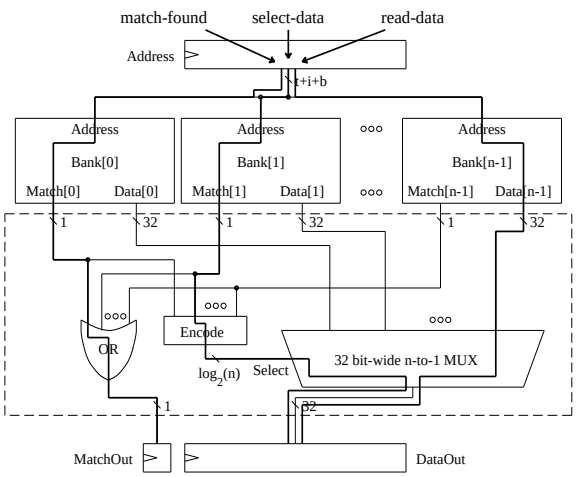
(b) $\Delta miss$ = -1/2%, t_{memory} = 20 cycles

==> Δt_{cache} < 0.1 cycle

A Direct-Mapped Cache - Critical Paths



A Set-Associative Cache - Critical Paths



Write Policies

Writes are harder

Reads done in parallel with tag compare; writes are not

Thus, writes are often slower
(but processor need not wait)

On hits, update memory?

Yes - write-through (store-through)

No - write-back (store-in, copy-back)

On misses, allocate cache block?

Yes - write-allocate
(usually used w/ write-back)

No - no-write-allocate
(usually used w/ write-through)

Write Policies, cont.

Write-Back

Update memory only on block replacement

Dirty bits used so clean blocks can be replaced without updating memory

Traffic/Reference = $fract_dirty \times miss \times B$

Traffic/Reference = $1/2 \times 0.05 \times 4 = 0.10$

Less traffic for larger caches

Write-Through

Update memory on each write

Keeps memory up-to-date (almost)

Traffic/Reference = $fraction_writes = 0.20$

Traffic independent of cache parameters

Write Buffers

Buffer CPU writes

Allow reads to proceed

Stall only when full

Data dependencies

Detect

Stall or bypass

write policy	*write-alloc?*	*hit or miss?*	*write buffer writes to*
back	yes	both	cache
back	no	hit	cache
back	no	miss	memory
thru	yes	both	both
thru	no	hit	both
thru	no	miss	memory

Write Buffers, cont.

More on write buffers

- Design for bursts
- Coalesce adjacent writes?
- Four is typical number

Can also "pipeline" writes

- Reads: read tag & data
- Writes: read tag, save current data, write previous data

Writeback buffers

- Between write-back cache and memory
- Buffers replaced, dirty blocks
- Allow read misses (fills) to proceed
- Usually only need 1 or 2 writeback buffers

Advanced Caches

Evaluation Methods

Two Levels of Cache

Reduce Miss Cost

Lockup-Free Caches

Beyond Simple Blocks

Software Restructuring

Prefetching

Software Prefetching

Victim Caches & Stream Buffers

Other

- Instructions
- Register Windows

Methods: Hardware Counters

Count hits & misses in hardware

See [Clark, TOCS 1983]

Advantages

- Accurate
- Realistic workload

Disadvantages

- Hard to do
- Requires machine to exist
- Hard to vary cache parameters
- Experiments not deterministic

Methods: Analytic Models

Mathematical expressions

Advantages

- Insight -- can vary parameters
- Fast
- Much recent progress

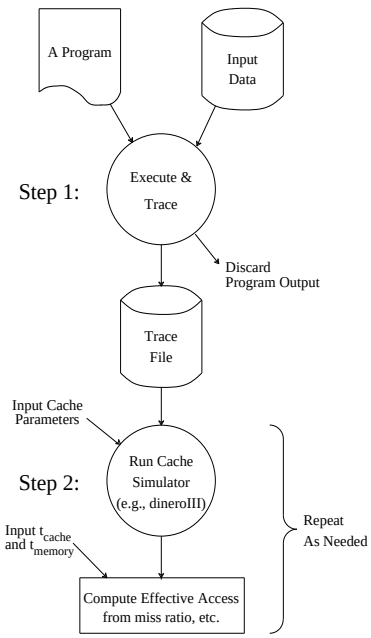
Disadvantages

- (Absolute) accuracy suspect for models w/ few parameters
- Hard/time-consuming to determine many parameter values

Questions

- Cache as black box?
- Simple or accurate?
- Comprehensive or single-aspect?

Methods: Trace-Driven Simulation



Trace-Driven Simulation, cont.

Advantages

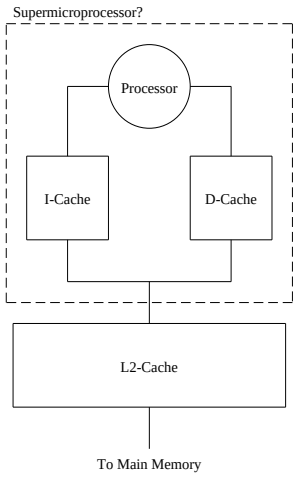
- Experiments repeatable
- Can be accurate
- Much recent progress

Disadvantages

- Reasonable traces are very large -- megabytes to gigabytes
- Simulation can be time-consuming
- Hard to say whether traces are representative

TDS is currently cache evaluation method of choice

Level Two Caches



Why Level Two Caches?

Processors getting faster w.r.t. main memory

- Want larger caches to reduce frequency of more costly misses
- But larger caches are too slow for processor
- Solution: reduce the cost of misses with a second level of cache instead

Exploits today's technological boundary

- Can't put large cache on microprocessor
- Board designer can vary cost/performance

Can handle synonyms for virtual L1 caches

- Make L2 use physical address
- Add pointers to L1 cache to implement Goodman's scheme
- Requires multi-level inclusion

Multi-Level Inclusion

Multi-level inclusion holds if L2 cache always contains superset of data in L1 cache(s)

- Handle synonyms
- Filter coherence traffic
- Makes L1 writes simpler
  (for both write-through and write-back)

Example: Local LRU not sufficient

- Assume that L1 and L2 hold two and three blocks and both use local LRU
- Processor references: 1, 2, 1, 3, 1, 4
- Final contents of L1: 1, 4
- L1 misses: 1, 2, 3, 4
- Final contents of L2: 2, 3, 4, but not 1

Multi-Level Inclusion, cont.

Multi-level inclusion takes effort to maintain

- Make L2 cache have bits or pointers giving L1 contents
- Invalidate from L1 before replacing block from L2
- Number of pointers per L2 block
  is (L2 blocksize / L1 blocksize)

See [Wang, Baer, Levy, ISCA 1989]

Level Two Cache Design

L1 cache design similar to single-level cache design when main memories were ‘‘faster’’

Apply previous experience to L2 cache design?

L2 ‘‘global’’ miss ratios not significantly altered by presence of L1 cache if

- L2 cache size $\geq 8\times$ L1 cache size
  [Przybylski, et al., ISCA 1989]

BUT: L2 caches bigger than previous experience

- (256K - 16M bytes)

What is ‘‘miss ratio’’?

- Global -- L2 misses after L1 / references
- Local -- L2 misses after L1 / L1 misses
- Solo -- misses as only cache / references

Level Two Cache Example

Recall adding associativity to a single-level cache helped performance if

$\Delta t_{cache} + \Delta miss \times t_{memory} < 0$

$\Delta miss$ = -1/2%, t_{memory} = 20 cycles

==> $\Delta t_{cache} \ll 0.1$ cycle

Consider doing the same in an L2 cache, where

$$t_{avg} = t_{cache\,1} + miss\,1 \times t_{cache\,2} + miss\,2 \times t_{memory}$$

Improvement only if

$$miss\,1 \times \Delta t_{cache\,2} + \Delta miss\,2 \times t_{memory} < 0$$

$$\Delta t_{cache\,2} < \left(\frac{-\Delta miss\,2}{miss\,1} \right) \times t_{memory}$$

$$\Delta t_{cache\,2} < \left(\frac{0.0005}{0.05} \right) \times 100 = 1 \text{ cycle}$$

Reducing Miss Cost

If the main memory takes 8 cycles before delivering two words per cycle, we assumed

$$t_{memory} = t_{access} + B \times t_{transfer} = 8 + B \times 1/2$$

where B is block size in words

This implies the whole block is loaded before data returned

If main memory returned the reference first (requested-word-first) and the cache returned it to the processor before loading it into the cache data array (fetch-bypass),

$$t_{memory} = t_{access} + MB \times t_{transfer} = 8 + 2 \times 1/2$$

where MB is memory bus width in words

BUT ...

Reducing Miss Cost, cont.

What if processor references unloaded word in block being loaded?

- Must add (equivalent to) ‘‘word’’ valid bits
- Performance penalty significant?

Why not generalize?

- Handle other references that hit before any part of block is back?
- Handle other references to other blocks that miss?

Called ‘‘lockup-free’’

Lockup-Free Caches

Normal cache stalls while a miss is pending

Lockup-Free Caches [Kroft 1981, Sohi 1991]

(a) Handle hits while first miss is pending
(b) Handle hits & misses until K misses are pending

Potential benefit

(a) Overlap misses with useful work & hits
(b) Also overlap misses with each other

Only makes sense if processor

- Handles pending references *correctly*
- Often can do useful work with references pending (delayed loads, scoreboarding, etc.)
- Has misses that can be overlapped (for (b))

Lockup-Free Caches, cont.

Key implementation problems

(1) Handling reads to pending miss
(2) Handling writes to pending miss
(3) Keep multiple requests straight

MSHRs -- miss status holding registers

- Valid bit and tag -- associatively compared on every reference
- Status and pointer to block frame
- For every word
  - Input ID (destination register?)
  - Send to CPU
  - In input buffer
  - In block frame
  - Already-overwritten

Beyond Simple Blocks

Break block size into

- Address block associated with tag
- Transfer block transferred to/from memory

Larger address blocks

- Decrease address tag overhead
- But allow fewer blocks to resident

Larger transfer blocks

- Exploit spatial locality
- Amortize memory latency
- But take longer to load
- But replace more data already cached
- But cause unnecessary traffic

Beyond Simple Blocks, cont.

Address block size > transfer block size

- Usually implies valid (& dirty) bit per transfer block

Used in 360/85 to reduce tag comparison logic

- 1K byte sectors with 64 byte subblocks

On-Chip caches

- Pins limit data bandwidth, making t_{memory} (almost) proportional to block size ==> small transfer sizes
- Tag overhead too great on one/two-word blocks ==> larger address blocks

Transfer block size > address block size

- ‘‘Prefetch on miss’’
- E.g., early MIPS R2000 board

Software Restructing

If column-major

- `x[i+1,j]` follows `x[i,j]`
- `x[i,j+1]` long after `x[i,j]`

Poor code

```
do i = 1 to rows
  do j = 1 to cols
sum = sum + x[i,j]
```

Better code

```
do j = 1 to cols
  do i = 1 to rows
    sum = sum + x[i,j]
```

Optimizations

- Loop interchange (used above)
- Merging Arrays
- Loop Fusion
- Blocking

Prefetching

Even ‘‘demand fetching’’ prefetches other words in the referenced block

Prefetching is useless unless a prefetch ‘‘costs’’ less than demand miss

Prefetches should

- (a) Always get data back before it is referenced
- (b) Never get data not used
- (c) Never prematurely replace other data (less important in large caches)
- (d) Never interfere with other cache activity

Item (a) conflicts with (b), (c) and (d)

Hardware Prefetching

What to prefetch

- One block (spatially) ahead

When to prefetch

- On every reference
  - Hard determine whether block to be prefetched is already in cache
- On every miss
  - Better than doubling block size
- Tagged
  - Prefetch when prefetched item referenced
- On every miss, prefetch a sequence (for instruction caches when item small)

K.I.S.S.

Software Prefetching

Use compiler to try to

- Prefetch early
- Prefetch accurately

Simplied example

```
do j = 1 to cols
  do ii = 1 to rows by BLOCK
    prefetch(&(x[i,j])+BLOCK)
    do i = ii to ii+BLOCK-1
       sum = sum + x[i,j]
```

Prefetch into

- register (binding)
- cache (non-binding) -- preferred

Victim Caches & Stream Buffers

See [Jouppi 1990]

For Instructions

Exploit sequentiality by prefetching

Instruction Prefetch Buffer

- Word
- FIFO
- Cache block

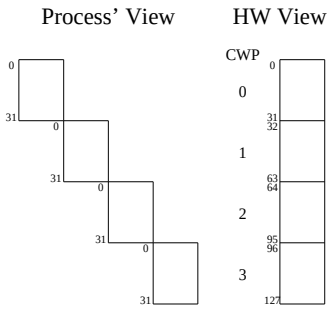
Could also prefetch at branch target(s)

- Use data fetch
- Multiple prefetch buffers
- Branch target cache

Pipelining

- Increases instruction bandwidth required
- Makes providing the bandwidth harder

For Registers



Register Windows

- Improve call/return performance
- Reduce register save/restore traffic
- Allow larger register file

But

- Traps on register window over/under-flow
- Slower on decode and context switches