CMSC 411
Computer Systems Architecture
Lecture 10
Memory Hierarchy (cont.)

Alan Sussman
als@cs.umd.edu

Administrivia

- Homework #2 due date moved to Thursday
- First exam next Thursday, March 13
- Project #1 due date moved to after spring break, on Wed., March 26
- Keep reading Ch. 2, through 2.3

Multi-level cache

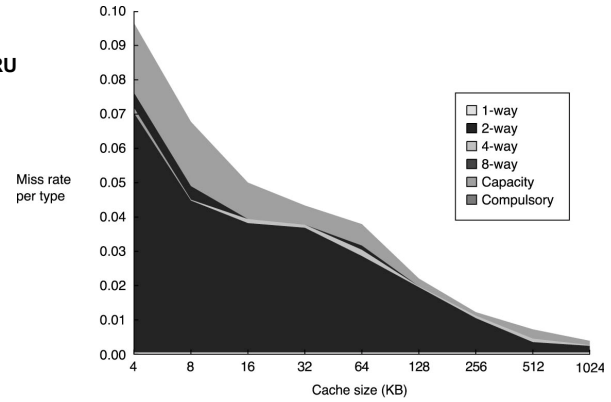
- For example, if cache takes 1 clock cycle, and memory takes 50, might be a good idea to add a larger (but necessarily slower) secondary cache in between, perhaps capable of 10 clock cycle access
- Complicates performance analysis (see H&P), but 2nd level cache captures many of 1st level cache misses, lowering effective miss penalty
  - and 3rd level cache has same benefits for 2nd level cache
- Most modern machines have separate 1st level instruction and data caches, shared 2nd level cache
  - and off processor chip shared 3rd level cache

Cache miss terminology

- Sometimes cache misses are inevitable:
  - The first time a block is used, need to bring it into cache (a *compulsory miss*)
  - If need to use more blocks at once than can fit into cache, some will bounce in and out (*capacity miss*)
  - In direct mapped or set associative caches, there are certain combinations of addresses that cannot be in cache at the same time (*conflict miss*)

Miss rate

Fig. B.9
SPEC2000, LRU replacement



How to reduce the miss rate?

- Use larger blocks
- Use more associativity, to reduce conflict misses
- Victim cache
- Pseudo-associative caches (won't talk about this)
- Prefetch (hardware controlled)
- Prefetch (compiler controlled)
- Compiler optimizations

Increasing block size

- **Want the block size** large **so don't have to stop so often to load blocks**
- **Want the block size** small **so that blocks load quickly**

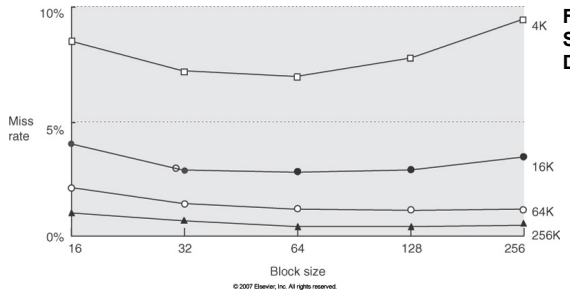


Fig. B.10
SPEC92 on DECstation 5000

Increasing block size (cont.)

- **So large block size may reduce miss rates, but …**
- **Example:**
  - **Suppose that loading a block takes 80 cycles (overhead) plus 2 clock cycles for each 16 bytes**
  - **A block of size 64 bytes can be loaded in 80 + 2*64/16 cycles = 88 cycles (miss penalty)**
  - **If the miss rate is 7%, then the average memory access time is**

    1 + .07 * 88 = 7.16 cycles

Memory Access Times

Fig. B.12 - SPEC92 on DECstation 5000

		Cache size			
Block size	Miss penalty	4K	16K	64K	256K
16	82	8.027	4.231	2.673	1.894
32	84	7.082	3.411	2.134	1.588
64	88	7.160	3.323	1.933	1.449
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

Higher associativity

- A direct-mapped cache of size *N* has about the same miss rate as a 2-way set-associative cache of size *N/2*
  - 2:1 cache rule of thumb (seems to work up to 128KB caches)
- But associative cache is slower than direct-mapped, so the clock may need to run slower
- Example:
  - Suppose that the clock for 2-way cache needs to run at a factor of 1.1 times the clock for 1-way cache
    - » the hit time increases with higher associativity
  - Then the average memory access time for 2-way is 1.10 + miss rate × 50 (assuming that the miss penalty is 50)

Memory access time

Fig. B.13 - SPEC92 on DECstation 5000

	Associativity			
Cache size (KB)	One-way	Two-way	Four-way	Eight-way
4	3.44	3.25	3.22	3.28
8	2.69	2.58	2.55	2.62
16	2.23	2.40	2.46	2.53
32	2.06	2.30	2.37	2.45
64	1.92	2.14	2.18	2.25
128	1.52	1.84	1.92	2.00
256	1.32	1.66	1.74	1.82
512	1.20	1.55	1.59	1.66

- If cache big enough, slowdown in access time hurts mem performance

Victim caches

- To remember a cache block that has recently been replaced (*evicted*)
  - use a small, *fully associative* cache between a cache and where it gets data from
  - check the victim cache on a cache miss, before going to next lower-level memory
    - » if found, swap victim block and cache block
  - reduces *conflict* misses

Victim caches (cont.)

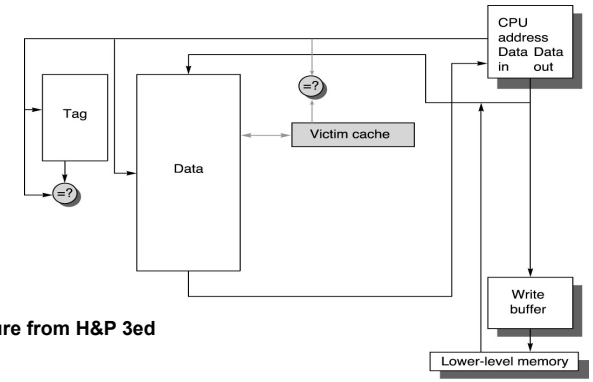


Figure from H&P 3ed

How to reduce the cache miss rate?

- Use larger blocks
- Use more associativity, to reduce conflict misses
- Victim cache
- **Prefetch (hardware controlled)**
- **Prefetch (compiler controlled)**
- **Compiler optimizations**

Hardware prefetch

- **Idea: If read page *k* of a book, the next page read is most likely page *k+1***
- **So, when a block is read from memory, read the next block too**
  - **maybe into a separate buffer that is accessed on a cache miss before going to memory**
- **Advantage:**
  - **if access blocks sequentially, will need to fetch only half as often from memory**
- **Disadvantages:**
  - **more data to move**
  - **may fill the cache with useless blocks**
  - **may compete with demand misses for memory bandwidth**

Compiler-controlled prefetch

- **Idea: The compiler has a better idea than the hardware does of when blocks are being use sequentially**
- **Want the prefetch to be nonblocking:**
  - **don't slow the pipeline waiting for it**
- **Usually want the prefetch to fail quietly:**
  - **if ask for an illegal block (one that generates a page fault or protection exception), don't generate an exception; just continue as if the fetch wasn't requested**
  - **called a *non-binding* cache prefetch**

Reducing the time for cache hits

- K.I.S.S.
- Use *virtual addresses* rather than *physical addresses* in the cache.
- Pipeline cache accesses
- Trace caches

K.I.S.S.

- Cache should be small enough to fit on the processor chip
- Direct mapped is faster than associative, especially on *read*
  - overlap tag check with transmitting data
- For current processors, relatively small L1 caches to keep fast clock cycle time, hide L1 misses with dynamic scheduling, and use L2 and L3 caches to avoid main memory accesses

Use virtual addresses

- Each process has its own *address space*, and no addresses outside that space can be accessed
- To keep address length small, each user addresses by offsets relative to some physical address in memory (pages)
- For example:

Physical address	Virtual address
5400	00
5412	12
5500	100

Virtual addresses (cont.)

- Since instructions use virtual addresses, use them for index and tag in cache, to save the time of translating to physical address space (the subject of a later part of this unit)
- Note that it is important to flush the cache and set all blocks invalid when switch to a new user in the OS (a *context switch*), since the same virtual address then may refer to a different physical address
  - or use the process/user ID as part of the tag in cache
- Aliases are another problem
  - when two different virtual addresses map to the same physical address – can get 2 copies in cache
    - » what happens when one copy is modified?

Pipelined cache access

- Latency to first level cache is more than one cycle
  - we've already seen this in Unit 3
- Benefit is fast cycle time
- Penalty is slower hits
  - also more clock cycles between a load and the use of the data (maybe more pipeline stalls)

Four compiler techniques

- 3 techniques to improve cache locality:
  - merging arrays
  - loop interchange
  - loop fusion

Compiler optimizations to reduce cache miss rate

Technique 1: merging arrays

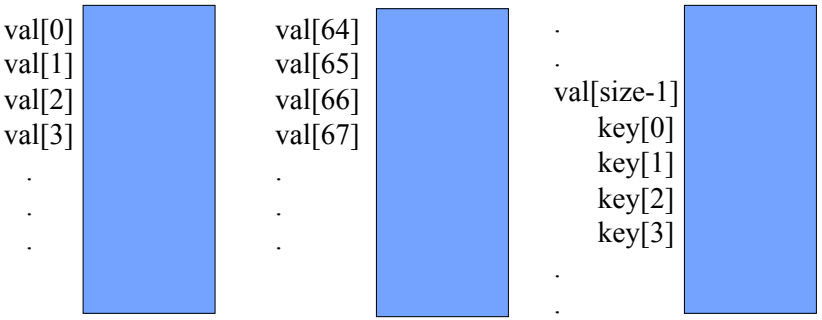
- Suppose have two arrays:

```
int val[size];
int key[size];
```

- and that usually use both of them together

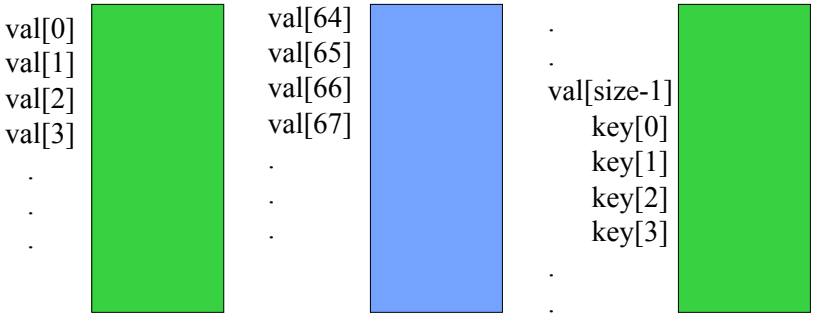
Merging arrays (cont.)

This is how they would be stored if cache blocksize is 64 words:



Merging arrays (cont.)

Means that at least 2 blocks must be in cache to begin using the arrays.



Merging arrays (cont.)

More efficient, especially if more than two arrays are coupled this way, to store them together.