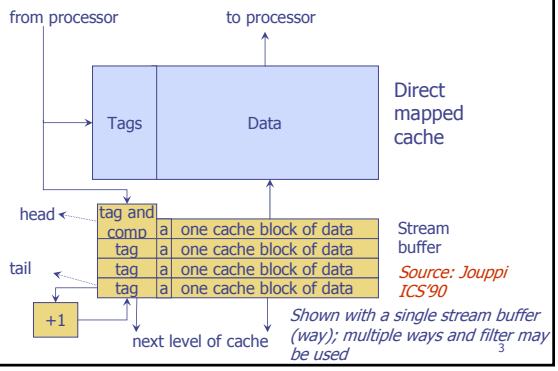
Lecture 17: Reducing Cache Miss Penalty and Reducing Cache Hit Time

Hardware prefetching and stream buffer, software prefetching, virtually indexed cache,

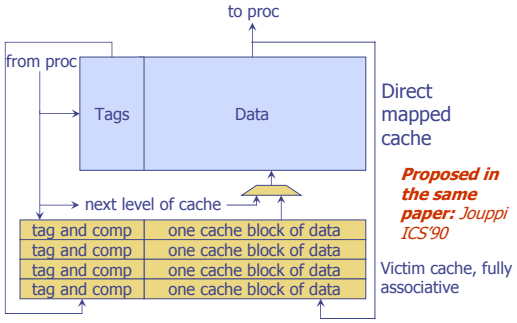
Reducing Misses by Hardware Prefetching of Instructions & Data

- 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

Stream Buffer Diagram



Victim Buffer Diagram



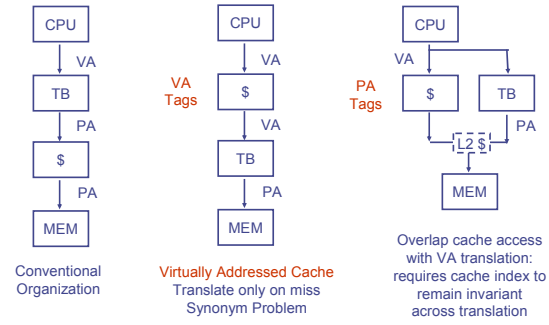
Reducing Misses by Software Prefetching Data

- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
- Prefetching comes in two flavors:
  - Binding prefetch: Requests load directly into register.
    - Must be correct address and register!
  - Non-Binding prefetch: Load into cache.
    - Can be incorrect. Frees HW/SW to guess!
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

Improving Cache Performance

1. Reducing miss rates
   - Larger block size
   - larger cache size
   - higher associativity
   - victim caches
   - way prediction and Pseudoassociativity
   - compiler optimization
2. Reducing miss penalty
   - Multilevel caches
   - critical word first
   - read miss first
   - merging write buffers
3. Reducing miss penalty or miss rates via parallelism
   - Non-blocking caches
   - Hardware prefetching
   - Compiler prefetching
4. Reducing cache hit time
   - **Small and simple caches**
   - **Avoiding address translation**
   - **Pipelined cache access**
   - **Trace caches**

Fast hits by Avoiding Address Translation



Conventional Organization

Virtually Addressed Cache
Translate only on miss
Synonym Problem

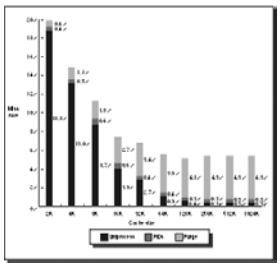
Overlap cache access with VA translation: requires cache index to remain invariant across translation

Fast hits by Avoiding Address Translation

- Send virtual address to cache? Called *Virtually Addressed Cache* or just *Virtual Cache* vs. *Physical Cache*
  - Every time process is switched logically must flush the cache; otherwise get false hits
    - Cost is time to flush + "compulsory" misses from empty cache
  - Dealing with *aliases* (sometimes called *synonyms*); Two different virtual addresses map to same physical address
  - I/O use physical addresses and must interact with cache, so need virtual address
- Antialiasing solutions
  - HW guarantees covers index field & direct mapped, they must be unique; called *page coloring*
- Solution to cache flush
  - Add *process identifier tag* that identifies process as well as address within process: can't get a hit if wrong process

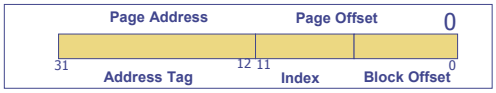
Fast Cache Hits by Avoiding Translation: Process ID impact

- Black is uniprocess
- Light Gray is multiprocess when flush cache
- Dark Gray is multiprocess when use Process ID tag
- Y axis: Miss Rates up to 20%
- X axis: Cache size from 2 KB to 1024 KB



Fast Cache Hits by Avoiding Translation: Index with Physical Portion of Address

- If a direct mapped cache is no larger than a page, then the index is physical part of address
- can start tag access in parallel with translation so that can compare to physical tag



- Limits cache to page size: what if want bigger caches and uses same trick?
  - Higher associativity moves barrier to right
  - Page coloring
- Compared with virtual cache used with page coloring?

Pipelined Cache Access

For multi-issue, cache bandwidth affects *effective* cache hit time

- Queueing delay adds up if cache does not have enough read/write ports

Pipelined cache accesses: reduce cache cycle time and improve bandwidth

Cache organization for high bandwidth

- Duplicate cache
- Banked cache
- Double clocked cache

Pipelined Cache Access

Alpha 21264 Data cache design

- The cache is 64KB, 2-way associative; cannot be accessed within one-cycle
- One-cycle used for address transfer and data transfer, pipelined with data array access
- Cache clock frequency doubles processor frequency; wave pipelined to achieve the speed

Trace Cache

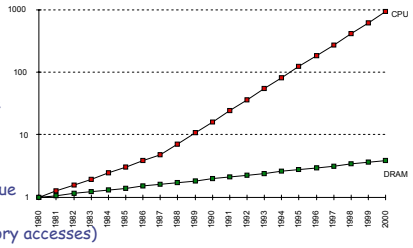
- Trace: a dynamic sequence of instructions including taken branches
- Traces are dynamically constructed by processor hardware and frequently used traces are stored into trace cache
- Example: Intel P4 processor, storing about 12K mops

Summary of Reducing Cache Hit Time

- Small and simple caches: used for L1 inst/data cache
  - Most L1 caches today are small but set-associative and pipelined (emphasizing throughput?)
  - Used with large L2 cache or L2/L3 caches
- Avoiding address translation during indexing cache
  - Avoid additional delay for TLB access

What is the Impact of What We've Learned About Caches?

- 1960-1985: Speed = *f*(no. operations)
- 1990
  - Pipelined Execution & Fast Clock Rate
  - Out-of-Order execution
  - Superscalar Instruction Issue
- 1998: Speed = *f*(non-cached memory accesses)
- What does this mean for
  - Compilers? Operating Systems? Algorithms? Data Structures?



Cache Optimization Summary

	Technique	MP	MR	HT	Complexity
miss penalty	Multilevel cache	+			2
	Critical work first	+			2
	Read first	+			1
	Merging write buffer	+			1
miss rate	Victim caches	+	+		2
	Larger block	-	+		0
	Larger cache		+	-	1
	Higher associativity		+	-	1
	Way prediction		+		2
	Pseudoassociative		+		2
	Compiler techniques		+		0

Cache Optimization Summary

	Technique	MP	MR	HT	Complexity
miss penalty	Nonblocking caches	+			3
	Hardware prefetching	+			2/3
	Software prefetching	+	+		3
hit time	Small and simple cache		-	+	0
	Avoiding address translation			+	2
	Pipeline cache access			+	1
	Trace cache			+	3