

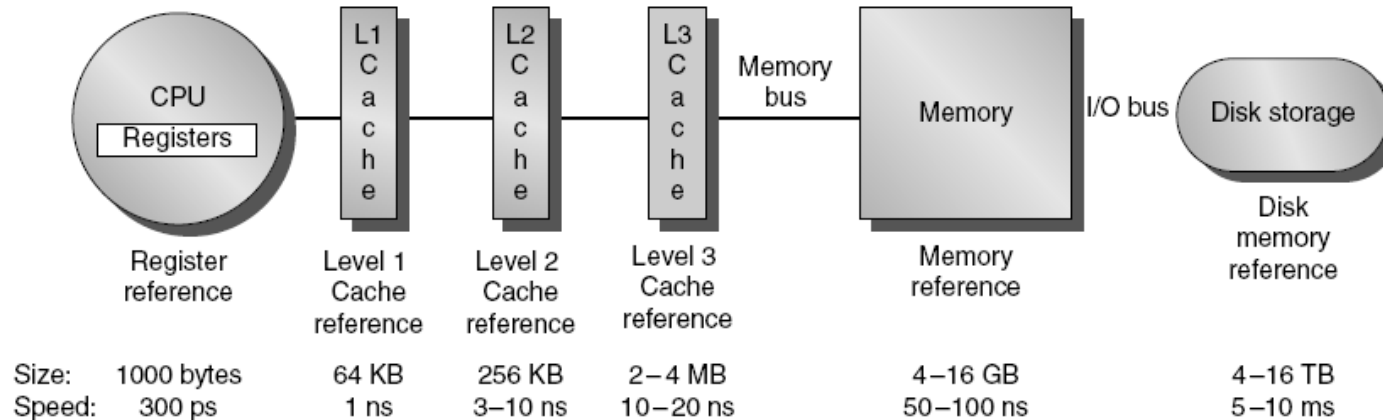
Chapter 2

Memory Hierarchy Design

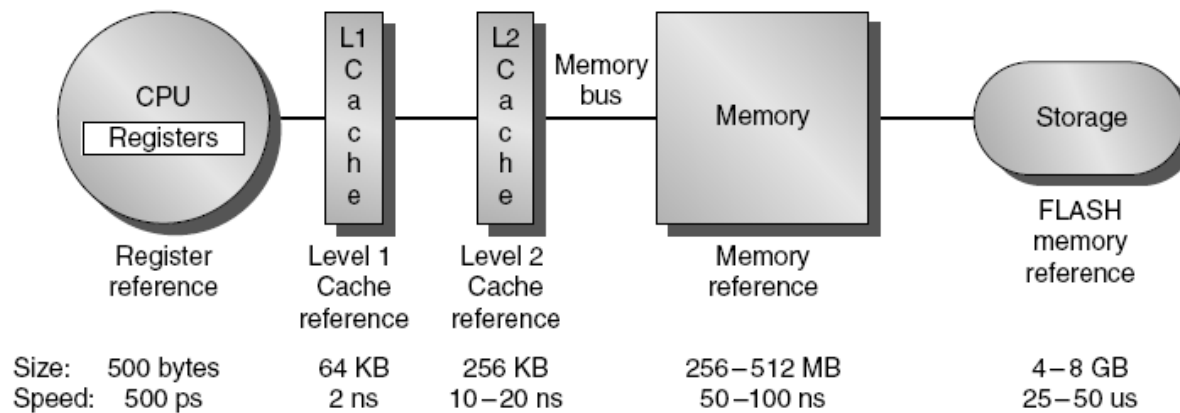
Introduction

- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution: organize memory system into a hierarchy
 - Entire addressable memory space available in largest, slowest memory
 - Incrementally smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
 - Gives the allusion of a large, fast memory being presented to the processor

Memory Hierarchy

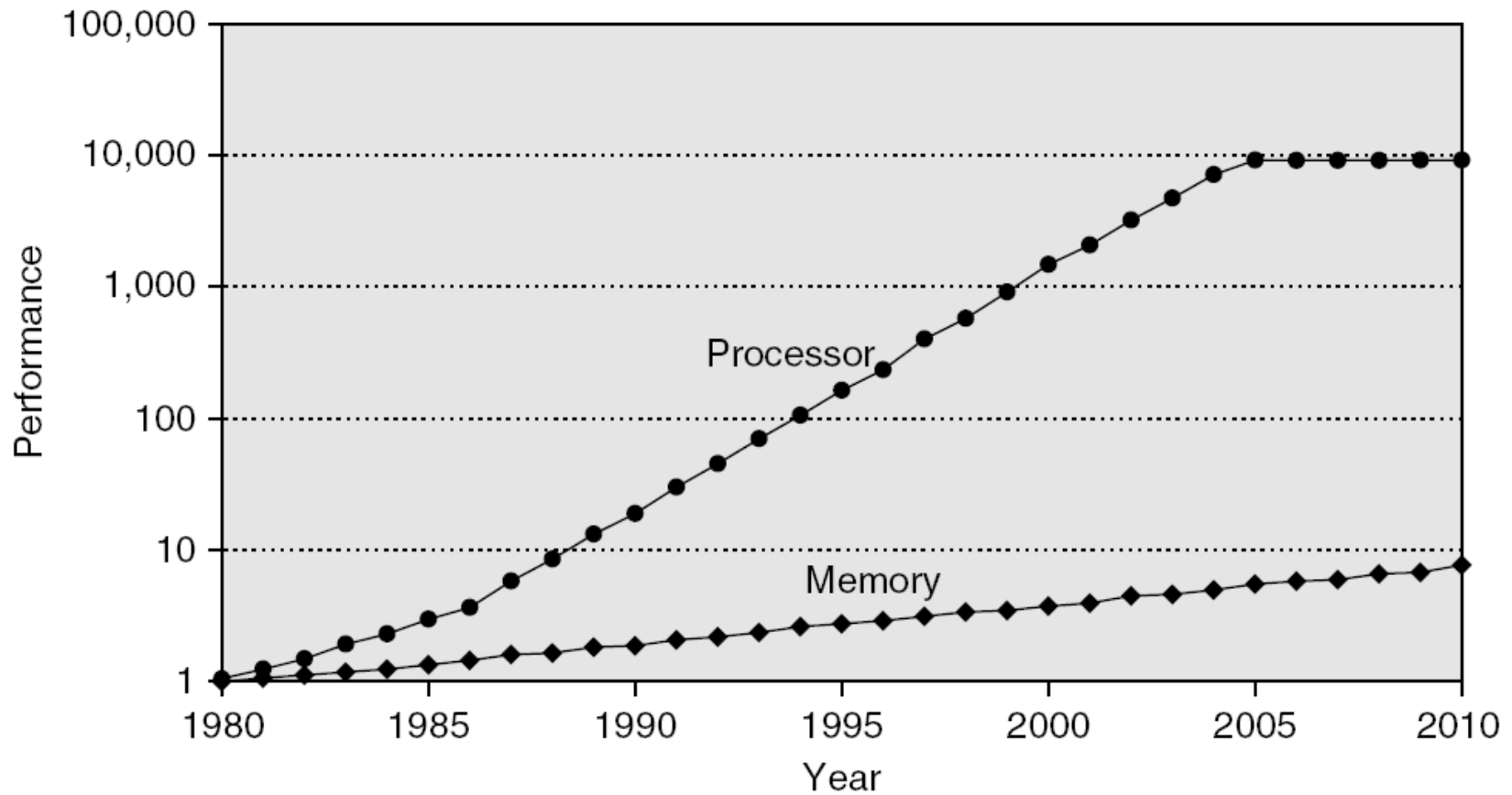


(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Memory Performance Gap



Memory Hierarchy Design

- Memory hierarchy design becomes more crucial with recent multi-core processors:
 - Aggregate peak bandwidth grows with # cores:
 - Intel Core i7 can generate two references per core per clock
 - Four cores and 3.2 GHz clock
 - 25.6 billion 64-bit data references/second +
 - 12.8 billion 128-bit instruction references
 - = 409.6 GB/s!
 - DRAM bandwidth is only 6% of this (25 GB/s)
 - Requires:
 - Multi-port, pipelined caches
 - Two levels of cache per core
 - Shared third-level cache on chip

Performance and Power

- High-end microprocessors have >10 MB on-chip cache
 - Consumes large amount of area and power budget

Memory Hierarchy Basics

- When a word is not found in the cache, a *miss* occurs:
 - Fetch word from lower level in hierarchy, requiring a higher latency reference
 - Lower level may be another cache or the main memory
 - Also fetch the other words contained within the *block*
 - Takes advantage of spatial locality
 - Place block into cache in any location within its *set*, determined by address
 - block address MOD number of sets

Memory Hierarchy Basics

- n sets \Rightarrow n -way set associative
 - *Direct-mapped cache* \Rightarrow one block per set
 - *Fully associative* \Rightarrow one set
- Writing to cache: two strategies
 - *Write-through*
 - Immediately update lower levels of hierarchy
 - *Write-back*
 - Only update lower levels of hierarchy when an updated block is replaced
 - Both strategies use *write buffer* to make writes asynchronous

Memory Hierarchy Basics

- Miss rate
 - Fraction of cache access that result in a miss
- Causes of misses
 - Compulsory
 - First reference to a block
 - Capacity
 - Blocks discarded and later retrieved
 - Conflict
 - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache

Memory Hierarchy Basics

$$\frac{\text{Misses}}{\text{Instruction}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

Average memory access time = Hit time + Miss rate \times Miss penalty

- Note that speculative and multithreaded processors may execute other instructions during a miss
 - Reduces performance impact of misses

Memory Hierarchy Basics

- Six basic cache optimizations:
 - Larger block size
 - Reduces compulsory misses
 - Increases capacity and conflict misses, increases miss penalty
 - Larger total cache capacity to reduce miss rate
 - Increases hit time, increases power consumption
 - Higher associativity
 - Reduces conflict misses
 - Increases hit time, increases power consumption
 - Higher number of cache levels
 - Reduces overall memory access time
 - Giving priority to read misses over writes
 - Reduces miss penalty
 - Avoiding address translation in cache indexing
 - Reduces hit time

These are covered in Appendix B!

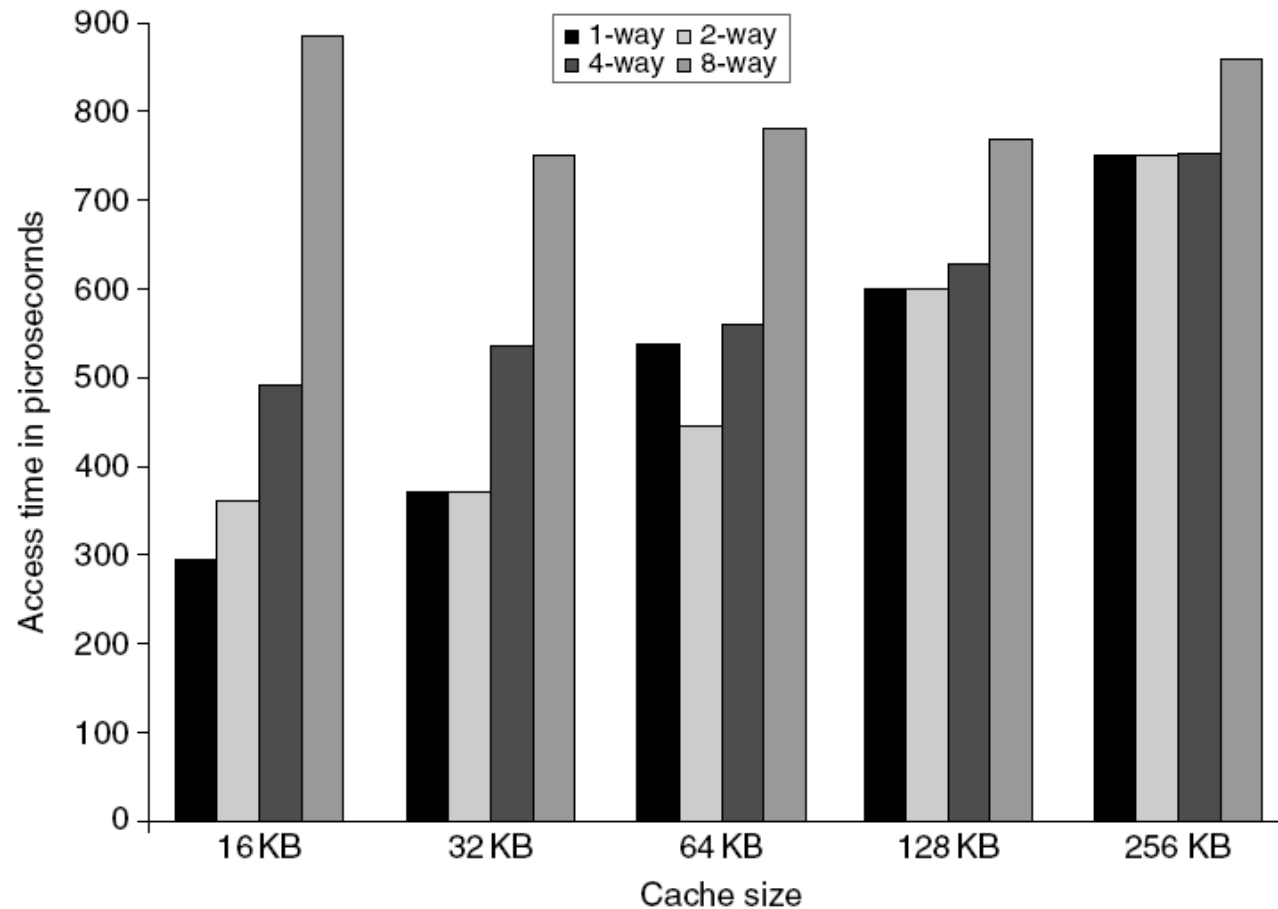
Ten Advanced Optimizations

- Reducing the Hit Time (and less power consumption)
 - 1: Small and simple L1
 - 2: Way prediction
- Increasing cache bandwidth
 - 3: Pipelined caches
 - 4: Multibanked caches
 - 5: Nonblocking caches
- Reducing the Miss Penalty
 - 6: Critical word fist
 - 7: Merging write buffers
- Reducing the Miss Rate
 - 8: Compiler optimization
- Reducing the Miss Rate/Penalty via parallelism
 - 9: Hardware prefetching
 - 10: Compiler prefetching

Small and simple L1 caches

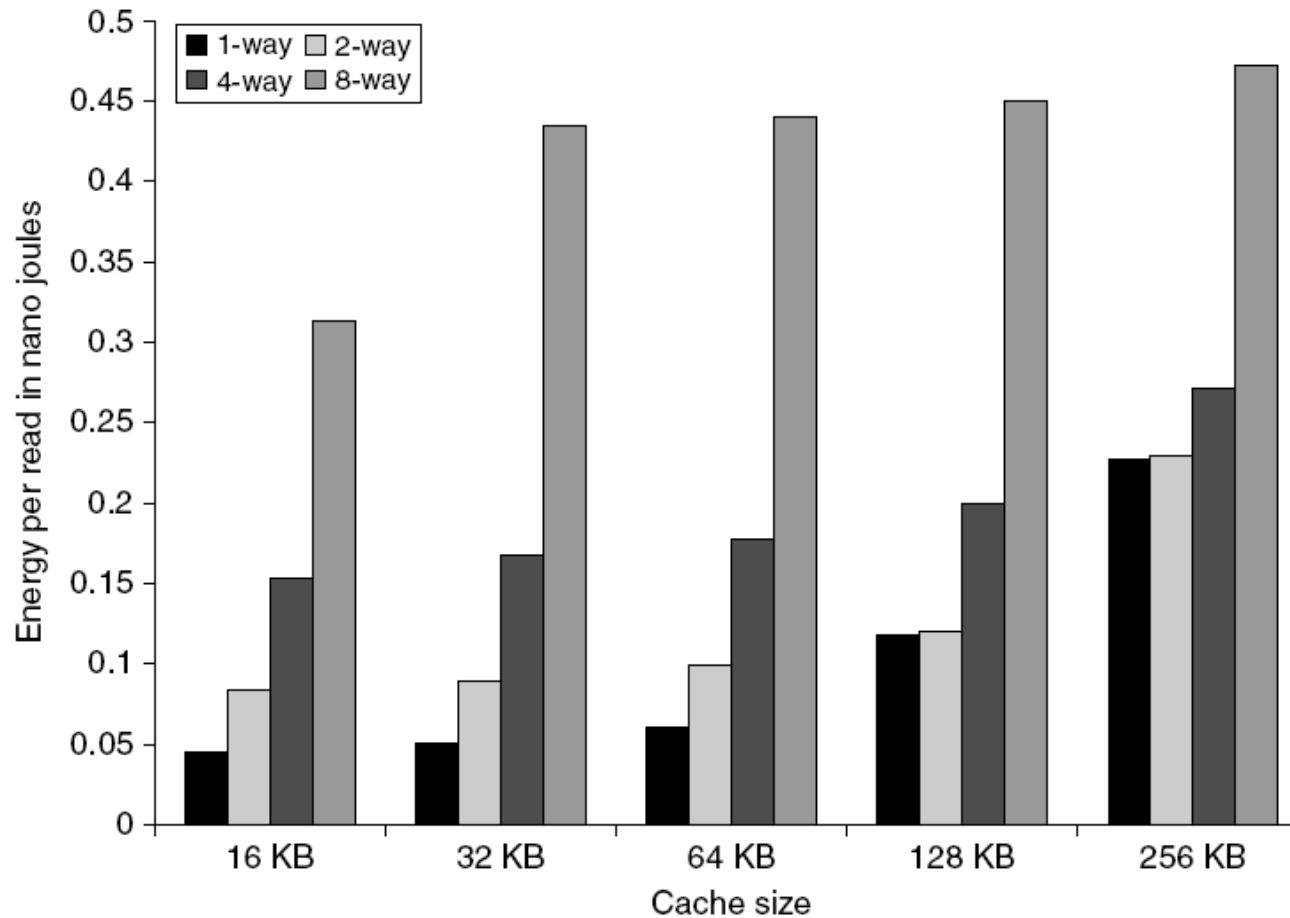
- Critical timing path:
 - addressing tag memory, then
 - comparing tags, then
 - selecting correct set (if associative)
- Direct-mapped caches can overlap tag compare and transmission of data
- Lower associativity reduces power because fewer cache lines are accessed

L1 Size and Associativity



Access time vs. size and associativity

L1 Size and Associativity



Energy per read vs. size and associativity

Way Prediction

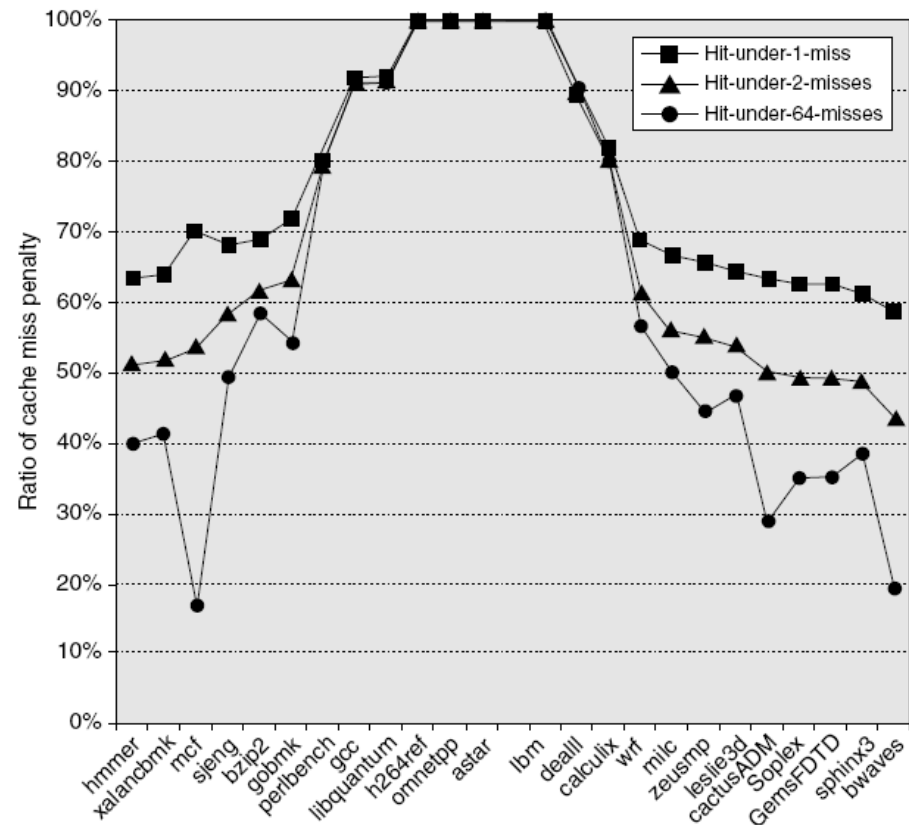
- To improve hit time, predict the way to pre-set mux
 - Mis-prediction gives longer hit time
 - Prediction accuracy
 - > 90% for two-way
 - > 80% for four-way
 - I-cache has better accuracy than D-cache
 - First used on MIPS R10000 in mid-90s
 - Used on ARM Cortex-A8
- Extend to predict block as well
 - “Way selection”
 - Increases mis-prediction penalty

Pipelining Cache

- Pipeline cache access to improve bandwidth
 - Examples:
 - Pentium: 1 cycle
 - Pentium Pro – Pentium III: 2 cycles
 - Pentium 4 – Core i7: 4 cycles
- Increases branch mis-prediction penalty
- Makes it easier to increase associativity

Nonblocking Caches

- Allow hits before previous misses complete
 - “Hit under miss”
 - “Hit under multiple miss”
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty



Nonblocking caches - Latency

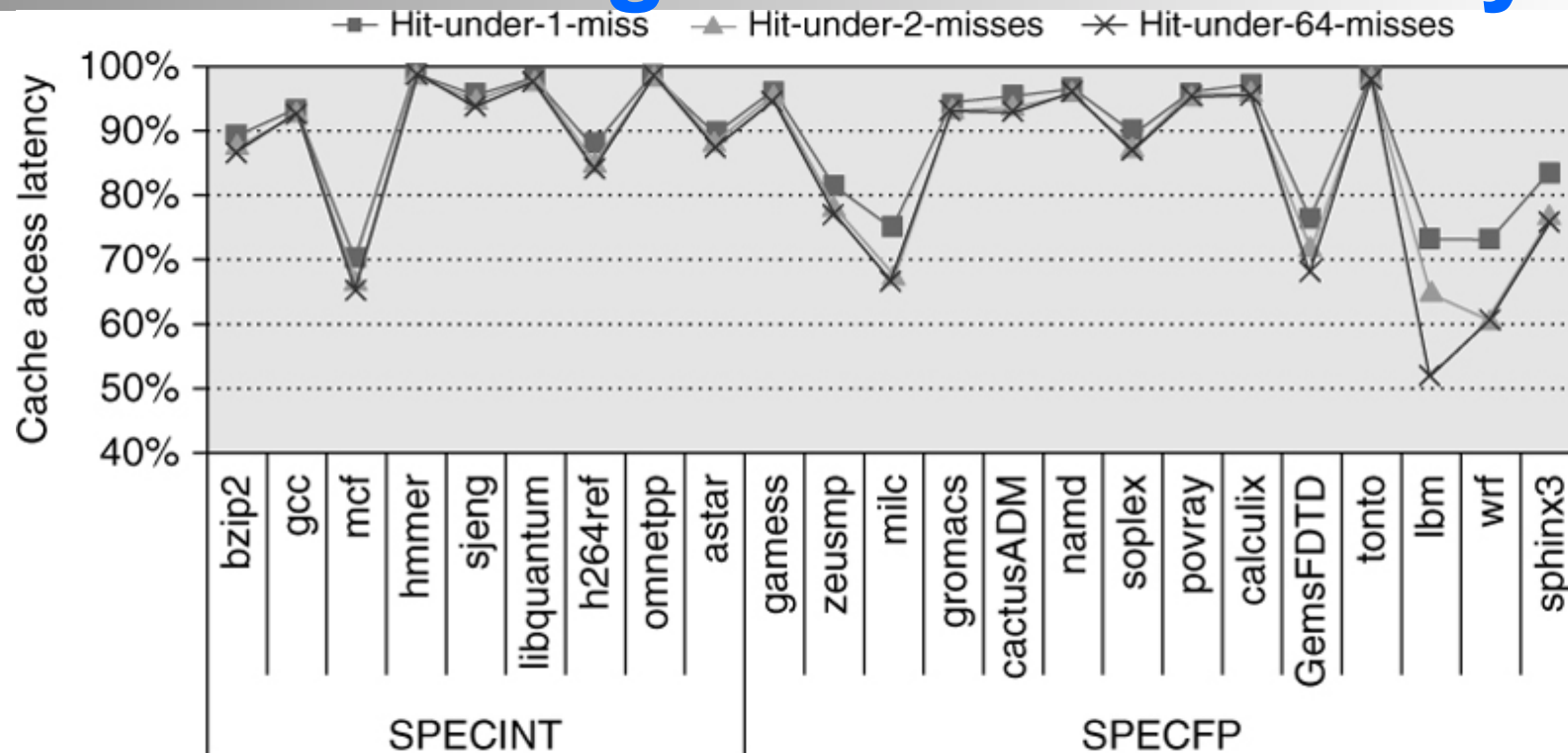


Figure 2.5 The effectiveness of a nonblocking cache is evaluated by allowing 1, 2, or 64 hits under a cache miss with 9 SPECINT (on the left) and 9 SPECFP (on the right) benchmarks. The data memory system modeled after the Intel i7 consists of a 32KB L1 cache with a four cycle access latency. The L2 cache (shared with instructions) is 256 KB with a 10 clock cycle access latency. The L3 is 2 MB and a 36-cycle access latency. All the caches are eight-way set associative and have a 64-byte block size. Allowing one hit under miss reduces the miss penalty by 9% for the integer benchmarks and 12.5% for the floating point. Allowing a second hit improves these results to 10% and 16%, and allowing 64 results in little additional improvement.

Exmpl p83: non blocking caches

Example Which is more important for floating-point programs: two-way set associativity or hit under one miss for the primary data caches? What about integer programs? Assume the following average miss rates for 32 KB data caches: 5.2% for floating-point programs with a direct-mapped cache, 4.9% for these programs with a two-way set associative cache, 3.5% for integer programs with a direct-mapped cache, and 3.2% for integer programs with a two-way set associative cache. Assume the miss penalty to L2 is 10 cycles, and the L2 misses and penalties are the same.

Answer For floating-point programs, the average memory stall times are

$$\text{Miss rate}_{\text{DM}} \times \text{Miss penalty} = 5.2\% \times 10 = 0.52$$

$$\text{Miss rate}_{\text{2-way}} \times \text{Miss penalty} = 4.9\% \times 10 = 0.49$$

Exmpl p83: non blocking caches (cont)

The cache access latency (including stalls) for two-way associativity is 0.49/0.52 or 94% of direct-mapped cache. The caption of Figure 2.5 says hit under one miss reduces the average data cache access latency for floating point programs to 87.5% of a blocking cache. Hence, for floating-point programs, the direct mapped data cache supporting one hit under one miss gives better performance than a two-way set-associative cache that blocks on a miss.

For integer programs, the calculation is

$$\text{Miss rate}_{\text{DM}} \times \text{Miss penalty} = 3.5\% \times 10 = 0.35$$

$$\text{Miss rate}_{\text{2-way}} \times \text{Miss penalty} = 3.2\% \times 10 = 0.32$$

The data cache access latency of a two-way set associative cache is thus 0.32/0.35 or 91% of direct-mapped cache, while the reduction in access latency when allowing a hit under one miss is 9%, making the two choices about equal.

Multibanked Caches

- Organize cache as independent banks to support simultaneous access
 - ARM Cortex-A8 supports 1-4 banks for L2
 - Intel i7 supports 4 banks for L1 and 8 banks for L2

- Interleave banks according to block address

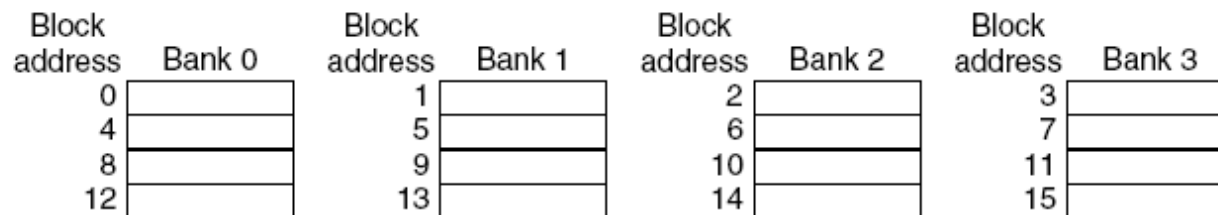


Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

Critical Word First, Early Restart

- Critical word first
 - Request missed word from memory first
 - Send it to the processor as soon as it arrives
- Early restart
 - Request words in normal order
 - Send missed word to the processor as soon as it arrives
- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched

Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer
- Do not apply to I/O addresses

Write address	V		V		V		V
100	1	Mem[100]	0		0		0
108	1	Mem[108]	0		0		0
116	1	Mem[116]	0		0		0
124	1	Mem[124]	0		0		0

No write buffering

Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

Write buffering

Compiler Optimizations

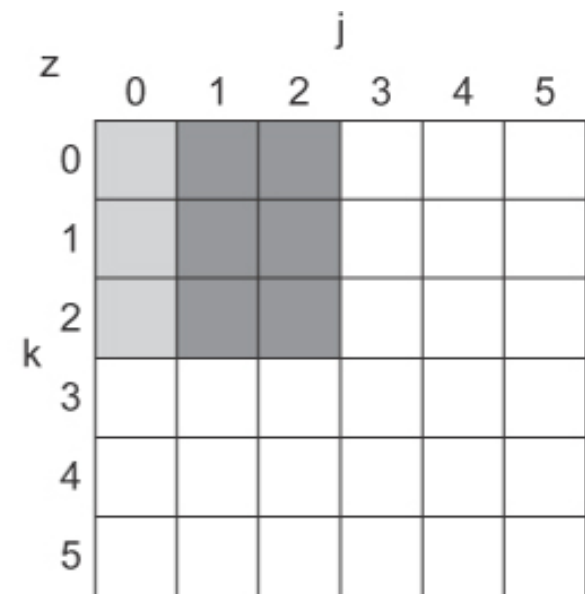
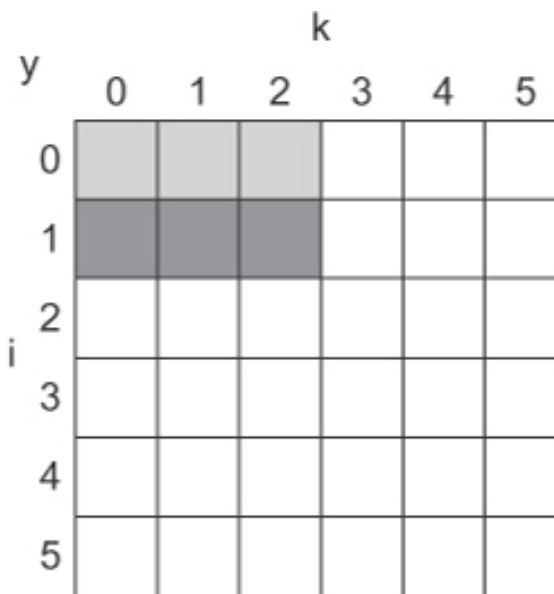
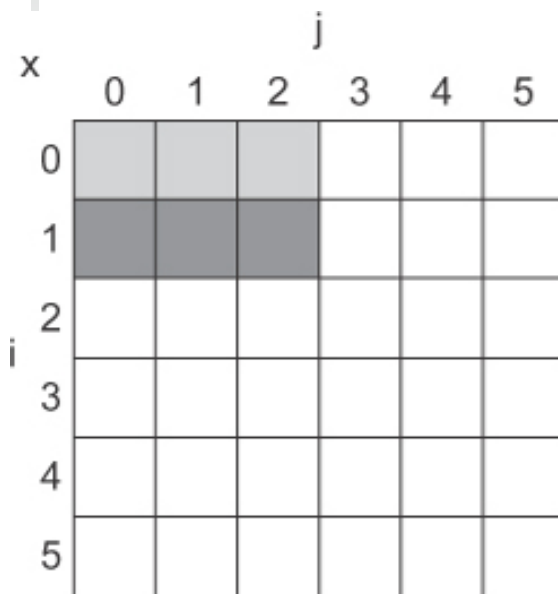
- Loop Interchange
 - Swap nested loops to access memory in sequential order (spacial locality)

```
/* Before */  
for (j = 0; j < 100; j = j+1)  
    for (i = 0; i < 5000; i = i+1)  
        x[i][j] = 2 * x[i][j];
```

```
/* After */  
for (i = 0; i < 5000; i = i+1)  
    for (j = 0; j < 100; j = j+1)  
        x[i][j] = 2 * x[i][j];
```

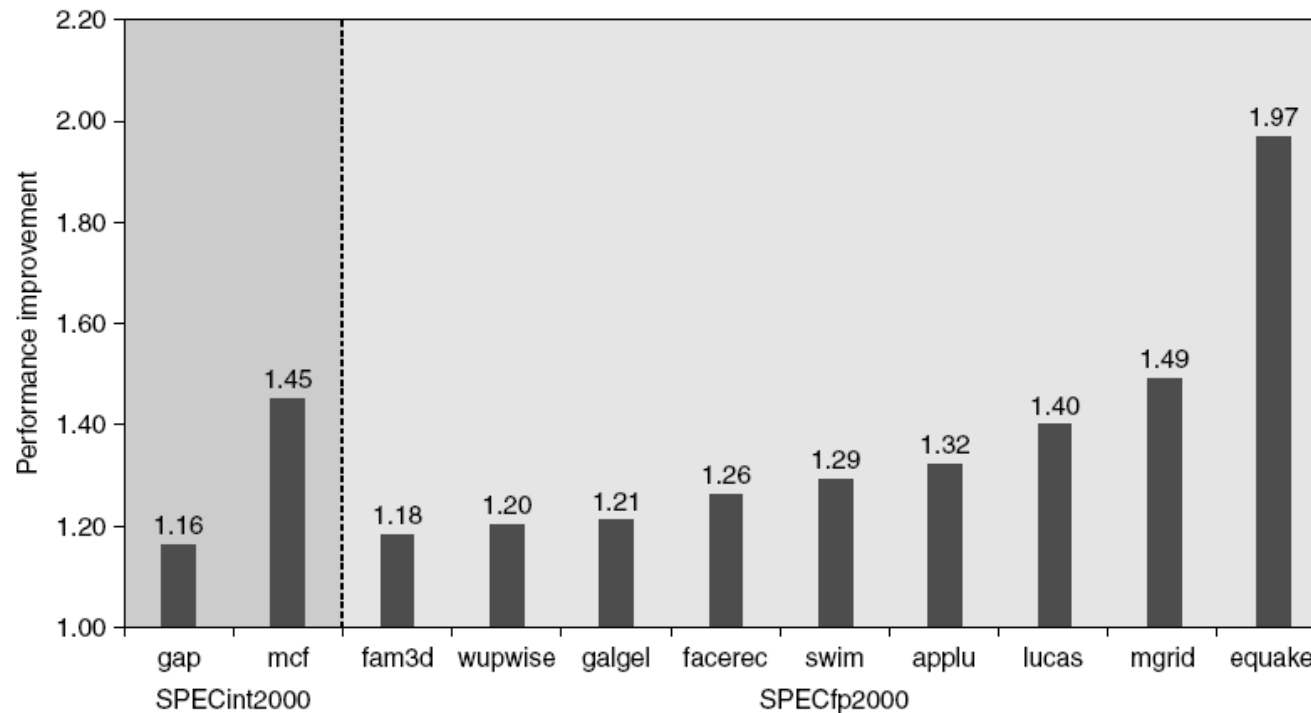
Compiler Optimizations

- Blocking
 - Instead of accessing entire rows or columns, subdivide matrices into blocks
 - Requires more memory accesses but improves temporal locality of accesses



Hardware Prefetching

- Fetch two blocks on miss (include next sequential block)



Pentium 4 Pre-fetching

Compiler Prefetching

- Insert prefetch instructions before data is needed
- Non-faulting: prefetch doesn't cause exceptions
- Register prefetch
 - Loads data into register
- Cache prefetch
 - Loads data into cache
- Combine with loop unrolling and software pipelining

Summary

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/complexity	Comment
Small and simple caches	+			-	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	-	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both i7 and Cortex-A8
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	-	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Figure 2.11 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, - means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.

Virtual Memory

- Protection via virtual memory
 - Keeps processes in their own memory space
- Role of architecture:
 - Provide user mode and supervisor mode
 - Protect certain aspects of CPU state
 - Provide mechanisms for switching between user mode and supervisor mode
 - Provide mechanisms to limit memory accesses
 - Provide TLB to translate addresses

Virtual Machines

- Supports isolation and security
- Sharing a computer among many unrelated users
- Enabled by raw speed of processors, making the overhead more acceptable

- Allows different ISAs and operating systems to be presented to user programs
 - “System Virtual Machines”
 - SVM software is called “virtual machine monitor” or “hypervisor”
 - Individual virtual machines run under the monitor are called “guest VMs”

Impact of VMs on Virtual Memory

- Each guest OS maintains its own set of page tables
 - VMM adds a level of memory between physical and virtual memory called “real memory”
 - VMM maintains shadow page table that maps guest virtual addresses to physical addresses
 - Requires VMM to detect guest’s changes to its own page table
 - Occurs naturally if accessing the page table pointer is a privileged operation