COEN-4730/EECE-5730 Computer Architecture

Lecture 4
Advanced Cache Optimizations
(Ch.2)

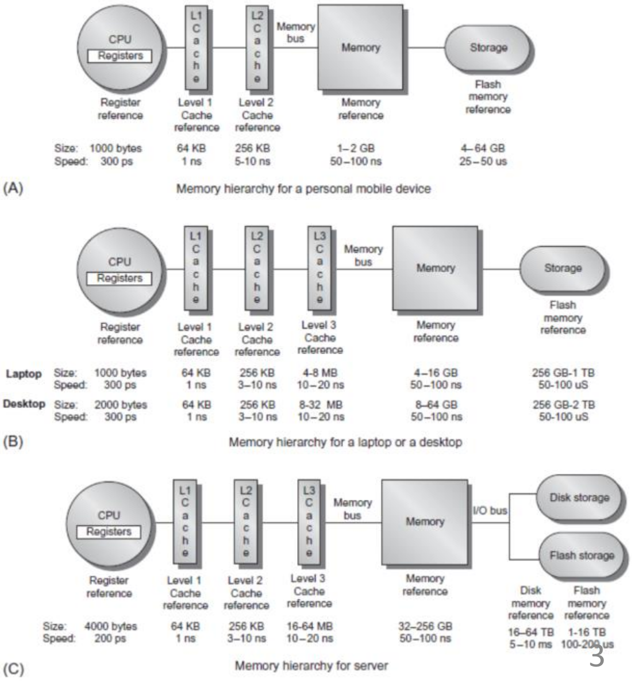
Cristinel Ababei
Dept. of Electrical and Computer Engineering
MARQUETTE UNIVERSITY
BE THE DIFFERENCE.

Credits: Slides adapted from presentations of Sudeep Pasricha and others: Kubiatowicz, Patterson, Mutlu, Elsevier

Outline

- Memory hierarchy
- 12 advanced cache optimizations
- Real world examples

Memory Hierarchy



(A) Memory hierarchy for a personal mobile device

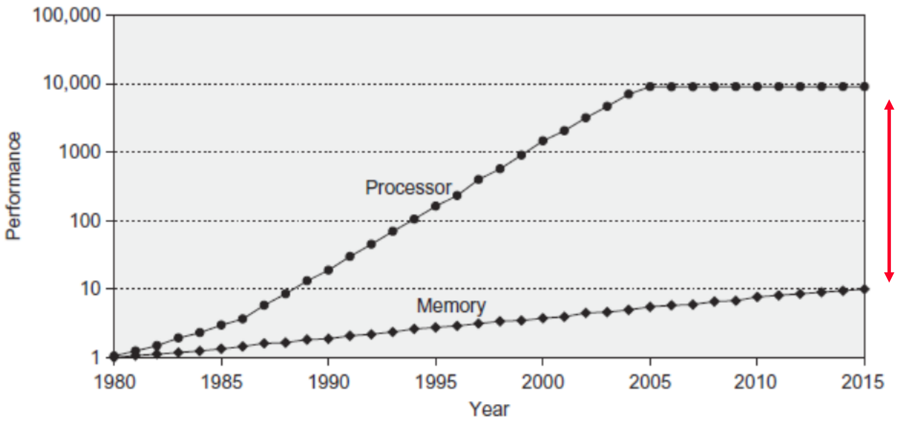
(B) Memory hierarchy for a laptop or a desktop

(C) Memory hierarchy for server

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 (34.1 GB/s)
    - Requires:
      - Multi-port, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip

Processor-DRAM Memory Gap (Latency)

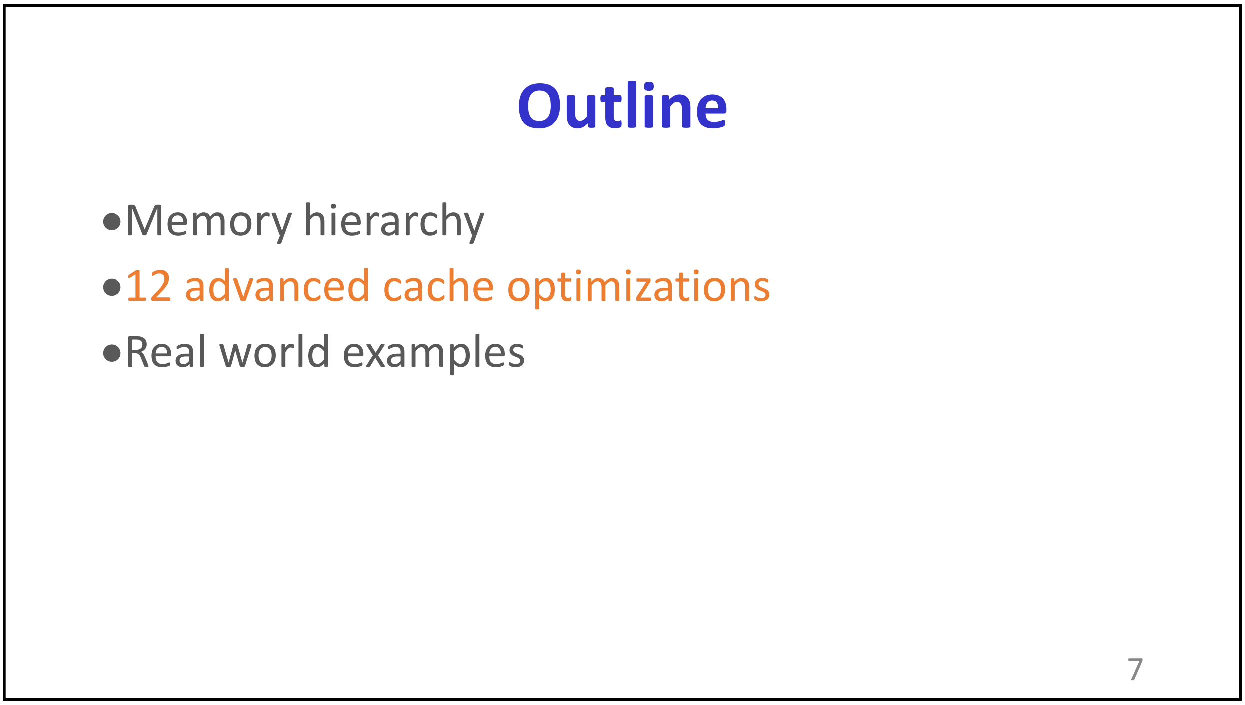


Processor-Memory Performance Gap: (grows ~50% / year)

6 Basic Cache Optimizations

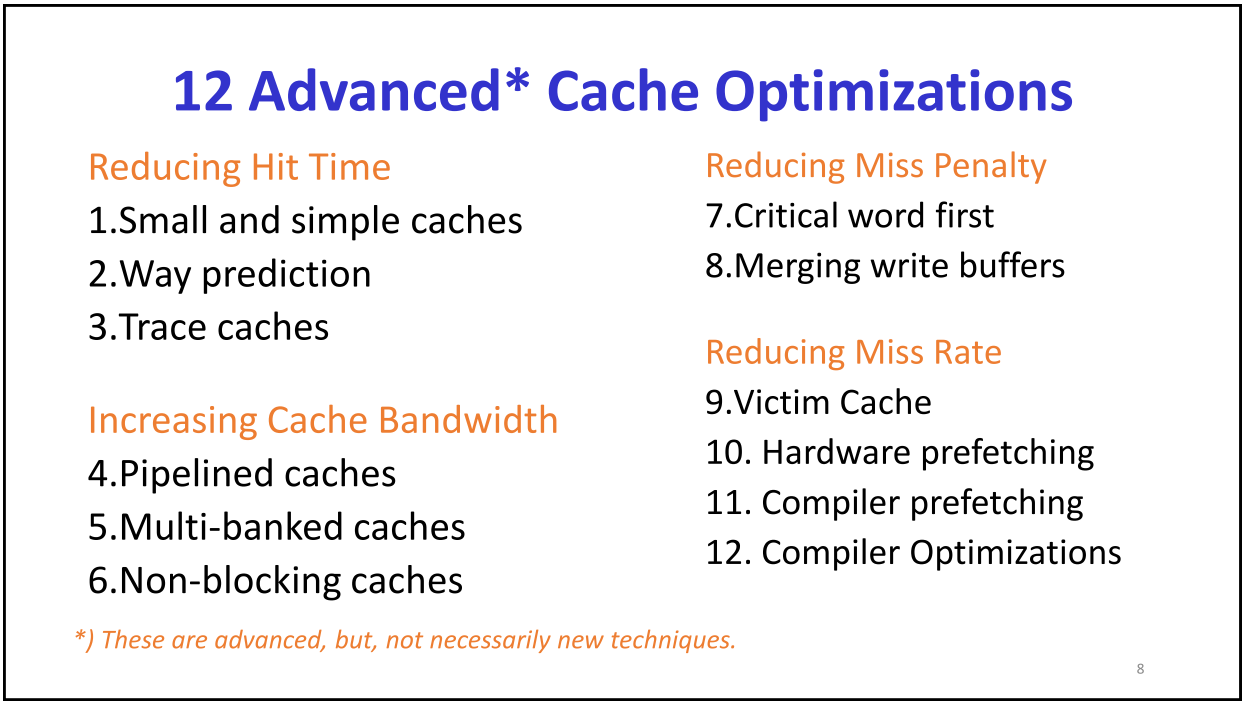
- Reducing Miss Rate
  1. Larger Block size (compulsory misses)
  2. Larger Cache size (capacity misses)
  3. Higher Associativity (conflict misses)
- Reducing Miss Penalty
  4. Multilevel Caches
  5. Giving Read Misses priority over Writes
- Reducing Hit Time
  6. Avoid address translation during indexing of Cache

avg. memory access time = AMAT = Hit time + Miss rate x Miss penalty



Outline

- Memory hierarchy
- 12 advanced cache optimizations
- Real world examples



12 Advanced* Cache Optimizations

Reducing Hit Time

1.Small and simple caches

2.Way prediction

3.Trace caches

Increasing Cache Bandwidth

4.Pipelined caches

5.Multi-banked caches

6.Non-blocking caches

Reducing Miss Penalty

7.Critical word first

8.Merging write buffers

Reducing Miss Rate

9.Victim Cache

10. Hardware prefetching

11. Compiler prefetching

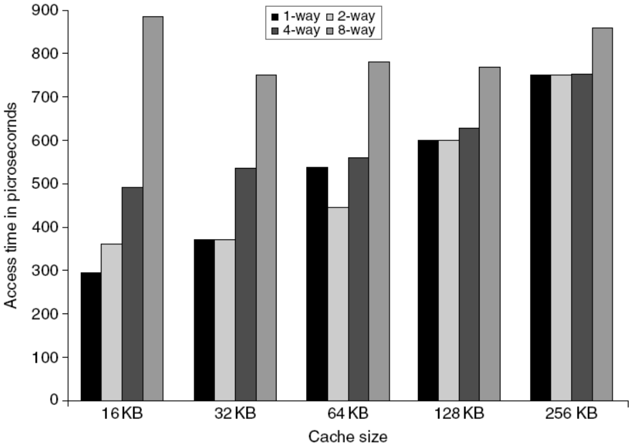
12. Compiler Optimizations

\) These are advanced, but, not necessarily new techniques.

1. Fast Hit Time: Small and Simple L1 Caches

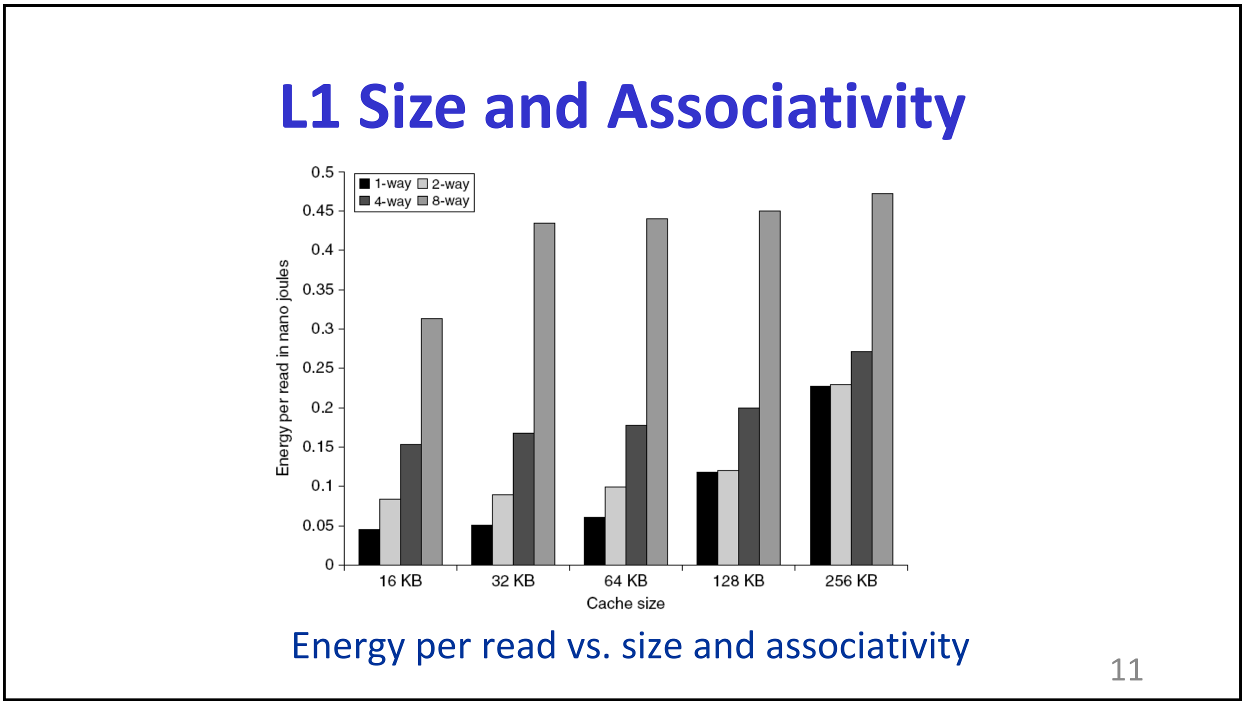
- Index into memory-cache and then compare tag(s) takes time
  - Critical timing path:
    - addressing tag memory, then
    - comparing tags, then
    - selecting correct set
- Small cache can help hit time since smaller memory takes less time to index to find right set of block(s) in cache
  - E.g., fast L1 caches were same small size for 3 generations of AMD microprocessors: K6, Athlon, and Opteron (smaller => shorter access lines)
  - Also, having a L2 cache small enough to fit on-chip with the processor avoids time penalty of going off chip (off-chip ~10X longer data latency, from capacitance)
- Simple cache ⇒ direct mapping
  - Overlap tag check with data transmission since no choice (kill data out if tag bad)

L1 Size and Associativity



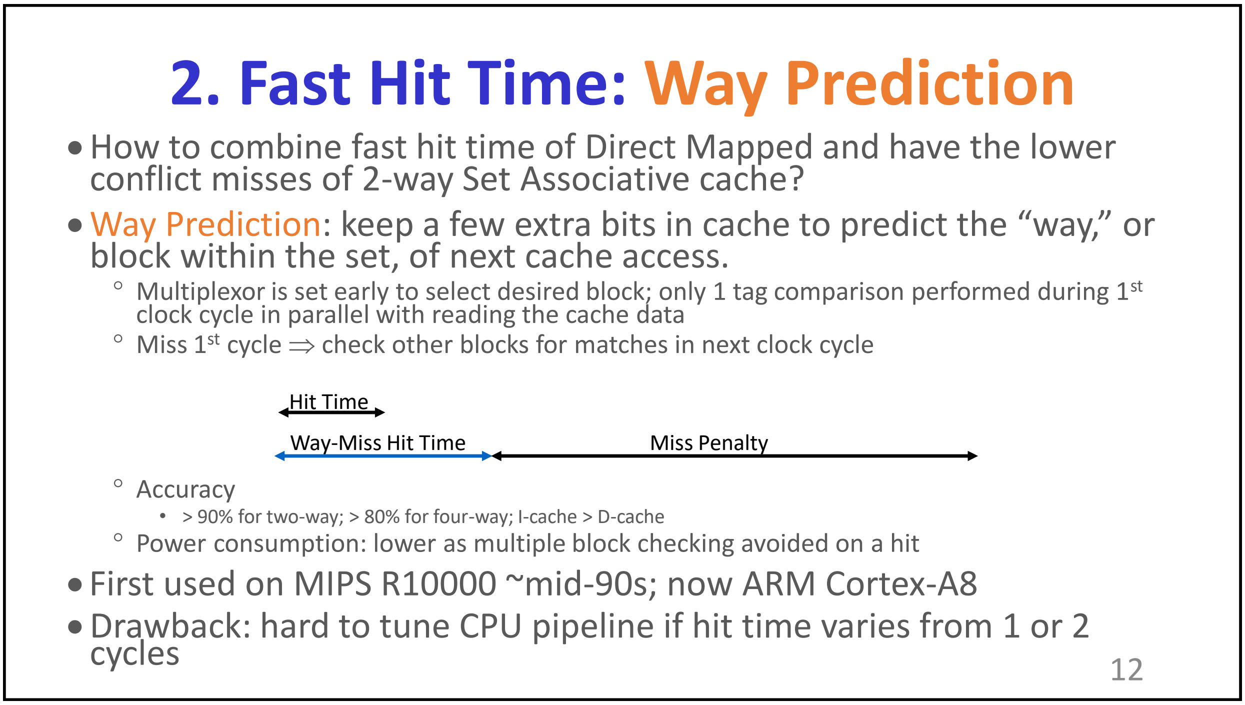
Using **CACTI 6.5**
40 nm

Access time vs. size and associativity



L1 Size and Associativity

Energy per read vs. size and associativity



2. Fast Hit Time: Way Prediction

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way Set Associative cache?
- Way Prediction: keep a few extra bits in cache to predict the "way," or block within the set, of next cache access.
  - Multiplexor is set early to select desired block; only 1 tag comparison performed during 1st clock cycle in parallel with reading the cache data
  - Miss 1st cycle ⇒ check other blocks for matches in next clock cycle

Hit Time

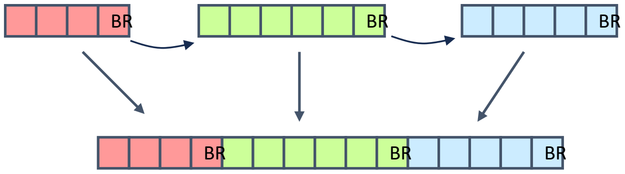
Way-Miss Hit Time

Miss Penalty

  - Accuracy
    - > 90% for two-way; > 80% for four-way; I-cache > D-cache
  - Power consumption: lower as multiple block checking avoided on a hit
- First used on MIPS R10000 ~mid-90s; now ARM Cortex-A8
- Drawback: hard to tune CPU pipeline if hit time varies from 1 or 2 cycles

3. Fast (Instruction Cache) Hit Times via: Trace Cache

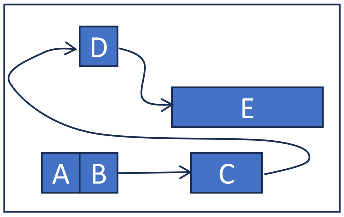
- Key Idea: Pack multiple non-contiguous basic blocks into one contiguous trace cache line



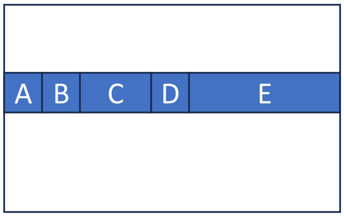
- Single fetch brings in multiple basic blocks
- Trace cache indexed by start address **AND** next *n* branch predictions

Trace Cache

- A trace is a sequence of instructions starting at any point in a dynamic instruction stream.
- It is specified by a start address and the branch outcomes of control instructions.



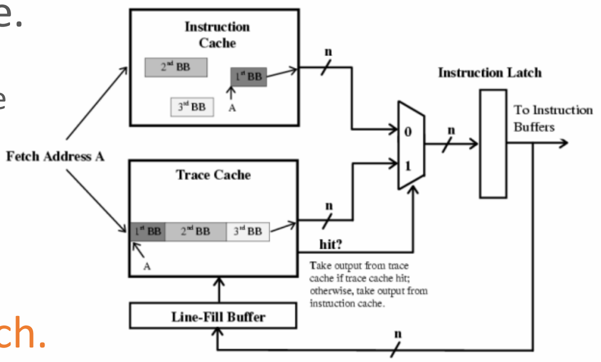
Instruction Cache



Trace Cache

Fetch Mechanism

- Trace cache is accessed in parallel with instruction cache.
  - Hit → Trace read into issue buffer
  - Miss → Fetch from instruction cache
- Trace cache hit if:
  - Fetch address match
  - Branch predictions match
- Trace cache is NOT on the critical path of instruction fetch.



Take output from trace cache if trace cache hit; otherwise, take output from instruction cache.

4. Increase Cache Bandwidth: Pipelining Cache

- Pipeline cache access to improve bandwidth
  - Examples: (I-cache access)
    - 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

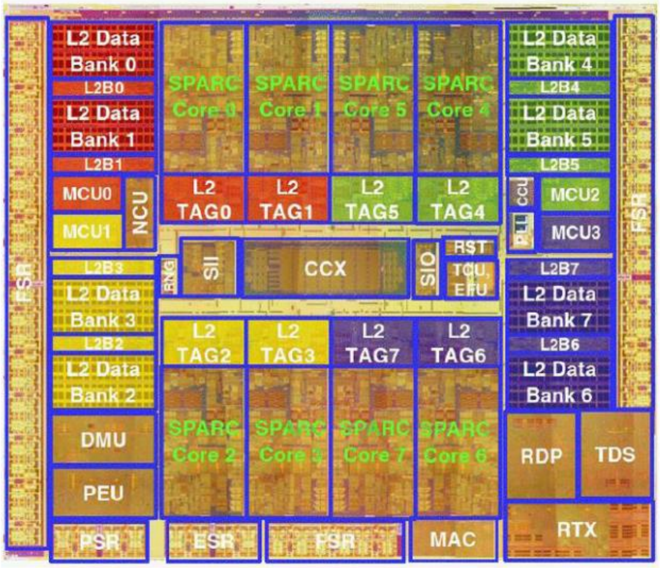
5. Increase Cache Bandwidth: Multibanked Caches

Block address	Bank 0	Block address	Bank 1	Block address	Bank 2	Block address	Bank 3
0		1		2		3	
4		5		6		7	
8		9		10		11	
12		13		14		15	

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.

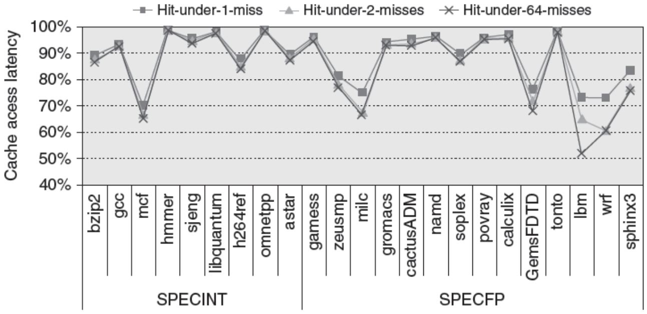
- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2
- Banking works best when accesses naturally spread themselves across banks ⇒ mapping of addresses to banks affects behavior of memory system
  - Sequential interleaving (see figure above)
- Also reduces power consumption

Sun UltraSPARCT2 8-bank L2 cache



6. Increase Cache Bandwidth: Nonblocking Caches

- Allow hits before previous misses complete
  - "Hit under miss"
  - "Hit under multiple miss"
- Intel i7 supports both; ARM A8 has limited support in L2
- In general, processors can hide L1 miss penalty but not L2 miss penalty



Nonblocking cache in Intel i7

32KB L1 cache, 4-cycle latency

256 KB L2 cache,10-cycle latency

2 MB L3 cache, 36-cycle latency

All caches 8-way set associative, 64-byte block size

9-12.5% saving for hit-under-1-miss
10-16% for hit-under-2-misses
Little improvement for 64 case

- **Deciding how many outstanding misses to support is complex**
- **Larger caches and L3 caches have reduced benefits**

7. Reduce Miss Penalty: Early Restart (minor help) & Critical Word First (major)

- Do not wait for full block before restarting CPU during Load
- Early restart - As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Spatial locality ⇒ tend to want next sequential word, so first access to a block is normally to 1st word, but next is to 2nd word, which may stall again and so on, so benefit from early restart alone is not clear
- Critical Word First - Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block
  - Large blocks more popular today ⇒ Critical Word 1st Widely Used

Block

8. Reduce Miss Penalty: 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
  - Multiword writes faster than writes performed one word at a time

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 merging

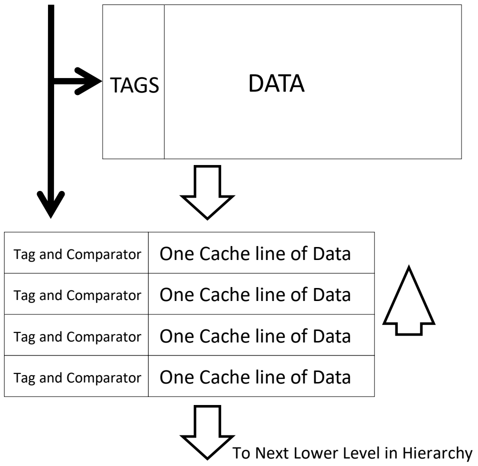
Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

Write merging

Sun T1 (Niagara) and Intel Core i7 processors, among many others, use write merging for fast write-through L1 (to L2) caches.

9. Reducing Misses: Victim Cache

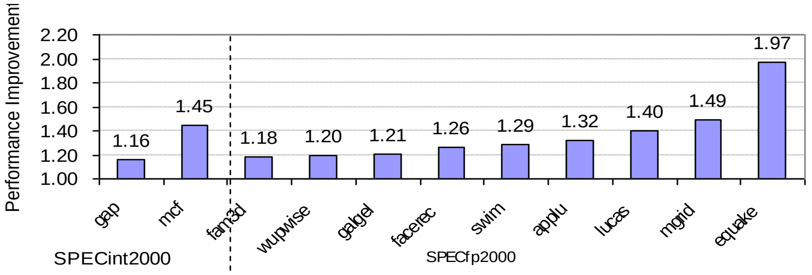
- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines



Tag and Comparator	One Cache line of Data
Tag and Comparator	One Cache line of Data
Tag and Comparator	One Cache line of Data
Tag and Comparator	One Cache line of Data

10. Reducing Misses by Hardware Prefetching of Instructions & Data

- Prefetching relies on having extra memory bandwidth that can be used without penalty
- Instruction Prefetching
  - Typically, CPU fetches 2 blocks on a miss: the requested block and the next consecutive block.
  - Requested block is placed in instruction cache when it returns, and prefetched block is placed into instruction stream buffer
- Data Prefetching
  - Pentium 4 can prefetch data into L2 cache from up to 8 streams from 8 different 4 KB pages
  - Prefetching invoked if 2 successive L2 cache misses to a page, if distance between those cache blocks is < 256 bytes



Pentium 4

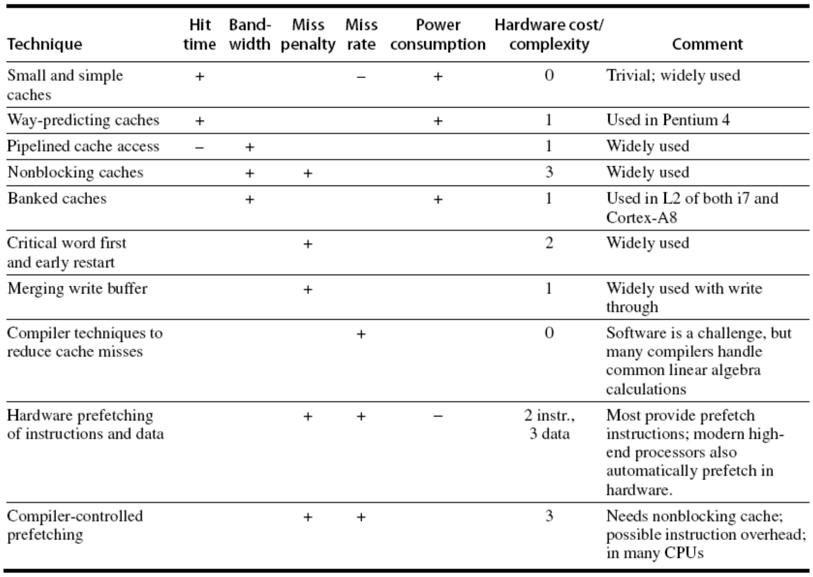
11. Reducing Misses by Software Prefetching Data

- Insert Prefetch instructions to request data before the processor needs it
- Data prefetch
  - Register 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
- Issuing prefetch instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

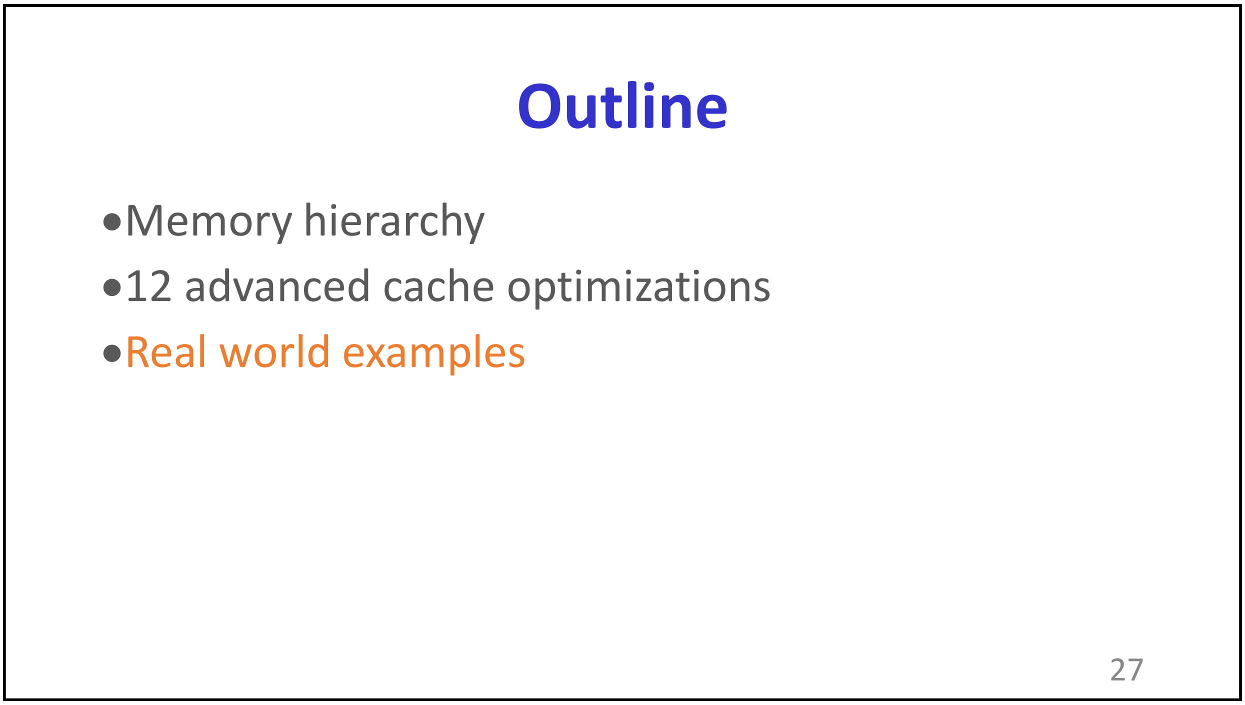
12. Reducing Misses by Compiler Optimizations

- McFarling [1989] reduced cache misses by 75% on 8 KB direct mapped cache, 4 Byte blocks in Software
- Instructions
  - Reorder procedures in memory to reduce conflict misses
  - Profiling to look at conflicts (using custom tools)
- Data
  - Merging Arrays: Improve spatial locality by single array of compound elements vs. 2 arrays
  - Loop Interchange: Change nesting of loops to access data in order stored in memory
  - Loop Fusion: Combine 2 independent loops that have same looping and some variables overlap
  - Blocking: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows

Summary: Advanced Cache Optimizations

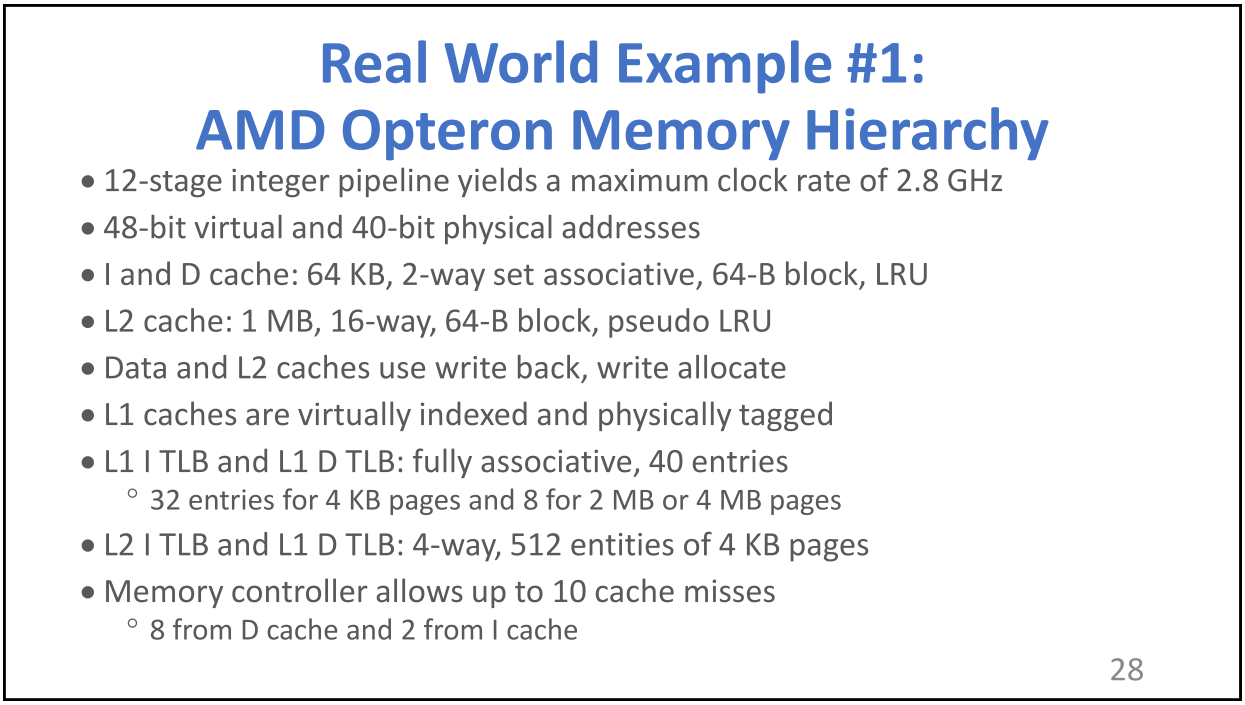


Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	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



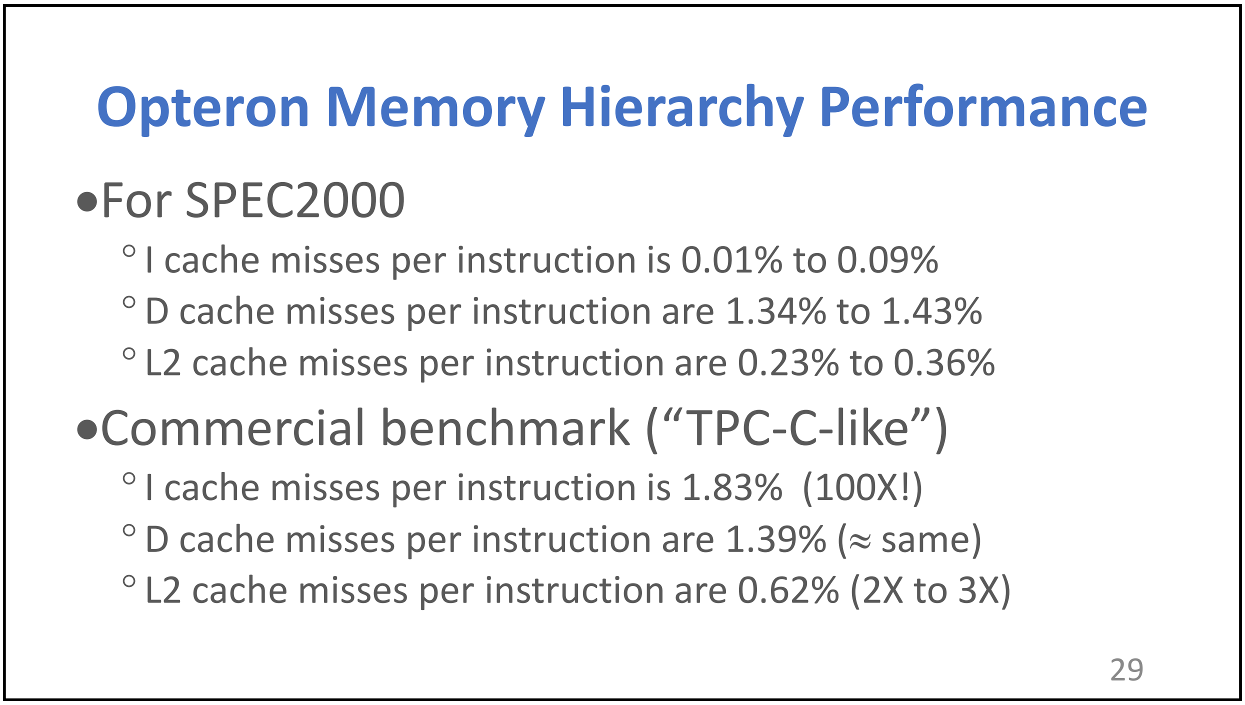
Outline

- Memory hierarchy
- 12 advanced cache optimizations
- Real world examples



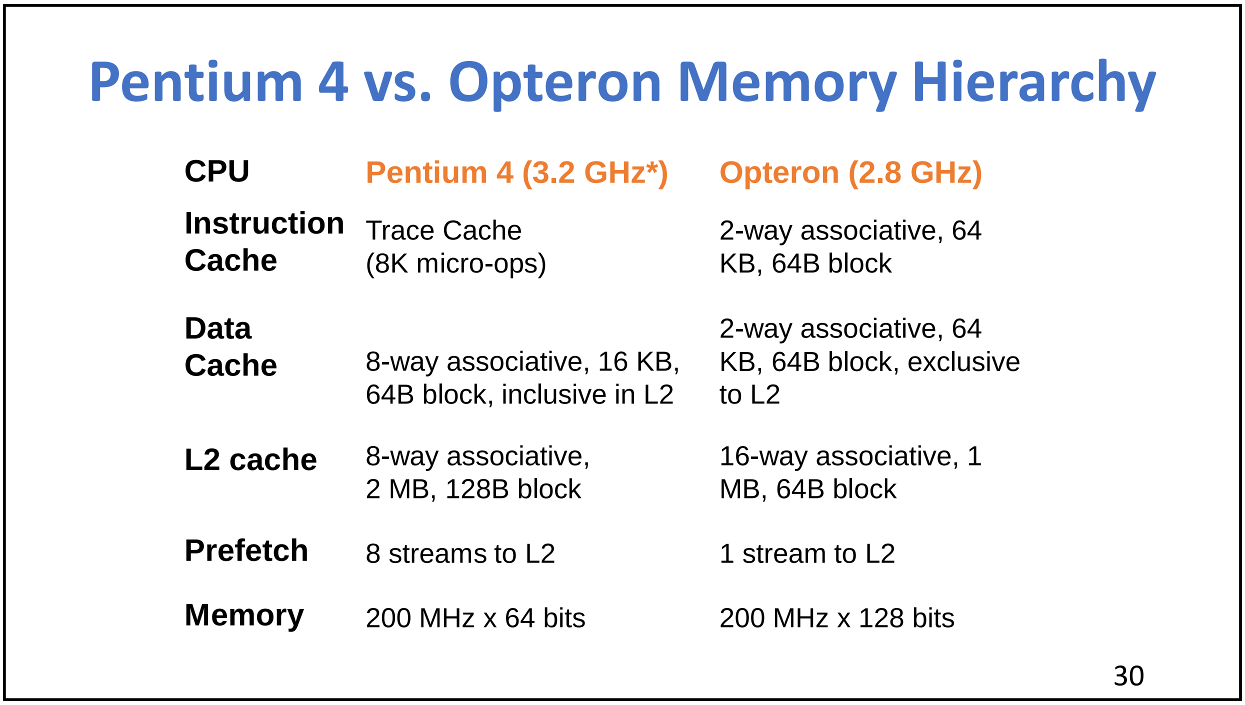
Real World Example #1: AMD Opteron Memory Hierarchy

- 12-stage integer pipeline yields a maximum clock rate of 2.8 GHz
- 48-bit virtual and 40-bit physical addresses
- I and D cache: 64 KB, 2-way set associative, 64-B block, LRU
- L2 cache: 1 MB, 16-way, 64-B block, pseudo LRU
- Data and L2 caches use write back, write allocate
- L1 caches are virtually indexed and physically tagged
- L1 I TLB and L1 D TLB: fully associative, 40 entries
  - 32 entries for 4 KB pages and 8 for 2 MB or 4 MB pages
- L2 I TLB and L1 D TLB: 4-way, 512 entities of 4 KB pages
- Memory controller allows up to 10 cache misses
  - 8 from D cache and 2 from I cache



Opteron Memory Hierarchy Performance

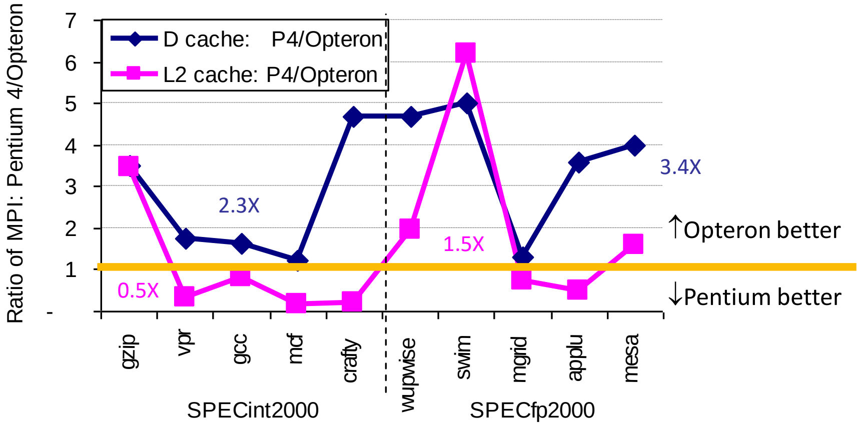
- For SPEC2000
  - I cache misses per instruction is 0.01% to 0.09%
  - D cache misses per instruction are 1.34% to 1.43%
  - L2 cache misses per instruction are 0.23% to 0.36%
- Commercial benchmark (“TPC-C-like”)
  - I cache misses per instruction is 1.83% (100X!)
  - D cache misses per instruction are 1.39% (≈ same)
  - L2 cache misses per instruction are 0.62% (2X to 3X)



Pentium 4 vs. Opteron Memory Hierarchy

CPU	Pentium 4 (3.2 GHz*)	Opteron (2.8 GHz)
Instruction Cache	Trace Cache (8K micro-ops)	2-way associative, 64 KB, 64B block
Data Cache	8-way associative, 16 KB, 64B block, inclusive in L2	2-way associative, 64 KB, 64B block, exclusive to L2
L2 cache	8-way associative, 2 MB, 128B block	16-way associative, 1 MB, 64B block
Prefetch	8 streams to L2	1 stream to L2
Memory	200 MHz x 64 bits	200 MHz x 128 bits

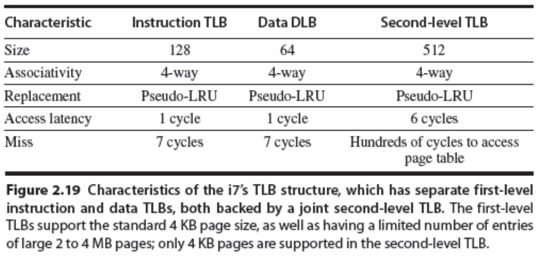
Misses Per Instruction (MPI): Pentium 4 vs. Opteron



- D cache miss: P4 is 2.3X to 3.4X vs. Opteron
- L2 cache miss: P4 is 0.5X to 1.5X vs. Opteron
- Note: Same ISA, but not same instruction count

Real World Example #2:
Intel core i7 Memory Hierarchy

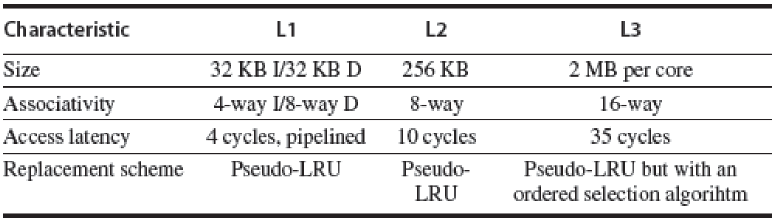
- Out-of-order execution processor that includes four cores and supports the x86-64 instruction set architecture
  - Each core can execute up to four 80x86 instructions per clock cycle using a multiple issue, dynamically scheduled, 16-stage pipeline
  - Can also support up to two simultaneous threads per processor, using a technique called simultaneous multithreading; 3.3 GHz clock rate
  - Uses 48-bit virtual addresses and 36-bit physical addresses
    - Yielding a maximum physical memory of 36 GB.
  - Memory management is handled with a two level TLB



Characteristic	Instruction TLB	Data DLB	Second-level TLB
Size	128	64	512
Associativity	4-way	4-way	4-way
Replacement	Pseudo-LRU	Pseudo-LRU	Pseudo-LRU
Access latency	1 cycle	1 cycle	6 cycles
Miss	7 cycles	7 cycles	Hundreds of cycles to access page table

Figure 2.19 Characteristics of the i7's TLB structure, which has separate first-level instruction and data TLBs, both backed by a joint second-level TLB. The first-level TLBs support the standard 4 KB page size, as well as having a limited number of entries of large 2 to 4 MB pages; only 4 KB pages are supported in the second-level TLB.

Intel core i7 Memory Hierarchy



Characteristic	L1	L2	L3
Size	32 KB I/32 KB D	256 KB	2 MB per core
Associativity	4-way I/8-way D	8-way	16-way
Access latency	4 cycles, pipelined	10 cycles	35 cycles
Replacement scheme	Pseudo-LRU	Pseudo-LRU	Pseudo-LRU but with an ordered selection algorihtm

- 3-level cache hierarchy
  - L1 is virtually indexed and physically tagged while the L2 and L3 caches are physically indexed
  - L1 and L2 are separate for each core; L3 is shared (max. 2 MB/core)
  - All three caches are nonblocking and allow multiple outstanding writes
  - A merging write buffer is used for the L1 cache, which holds data in the event that the line is not present in L1 when it is written. (That is, an L1 write miss does not cause the line to be allocated)
  - L3 is inclusive of L1 and L2
  - Replacement is by a variant on pseudo-LRU; in the case of L3 the block replaced is always the lowest numbered way whose access bit is turned off. This is not quite random but is easy to compute.

Intel core i7 Cache Performance

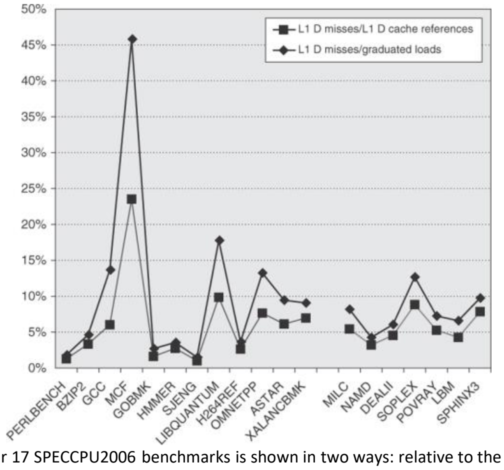


Figure 2.22 The L1 data cache miss rate for 17 SPECCPU2006 benchmarks is shown in two ways: relative to the actual loads that complete execution successfully and relative to all the references to L1, which also includes prefetches, speculative loads that do not complete, and writes, which count as references, but do not generate misses. These data, like the rest in this section, were collected by Professor Lu Peng and Ph.D. student Ying Zhang, both of Louisiana State University, based on earlier studies of the Intel Core Duo and other processors (see Peng et al. [2008]).

Intel core i7 Cache Performance

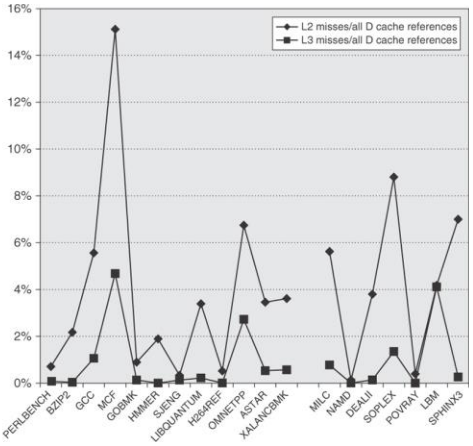


Figure 2.24 The L2 and L3 data cache miss rates for 17 SPECCPU2006 benchmarks are shown relative to all the references to L1, which also includes prefetches, speculative loads that do not complete, and program–generated loads and stores. These data, like the rest in this section, were collected by Professor Lu Peng and Ph.D. student Ying Zhang, both of Louisiana State University.

Intel core i7 Cache Performance

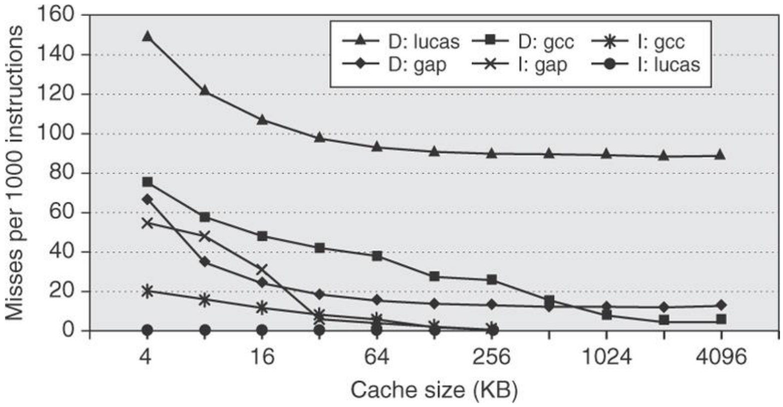


Figure 2.26 Instruction and data misses per 1000 instructions as cache size varies from 4 KB to 4096 KB. Instruction misses for gcc are 30,000 to 40,000 times larger than lucas, and, conversely, data misses for lucas are 2 to 60 times larger than gcc. The programs gap, gcc, and lucas are from the SPEC2000 benchmark suite.