CS61C: Great Ideas in Computer Architecture (aka Machine Structures)

Lecture 20: Caches II: Performance, Multilevel, Coherence

Instructors: Rosalie Fang, **Charles Hong**, Jero Wang

Announcements

- P/NP deadline is Friday, July 28
    - Student support meetings available with course staff (see Ed)
    - As always, instructor OH open for support and course feedback!
- Discussion cancellations
    - Imran's Monday 12pm - 1pm discussion is cancelled
    - Imran's Monday 1pm - 2pm discussion will now be taught by Ben
    - Ben's Wednesday 12pm - 1pm discussion is cancelled starting August 2nd
- Labs 7 and 8 due today, Thursday 7/27
- Project 3B due next Tuesday 8/1

Agenda

- Analyzing Cache Performance
- Multi-level Caches
- Caches in Real CPUs
- Cache Coherence

Analyzing Cache Performance

Analyzing Cache Performance: Terminology

- The **hit rate** of a cache is the percentage of accesses that result in a hit
- The **miss rate** of a cache is the percentage of accesses that result in a miss
- **Hit time** is how long it takes to check the cache. The **miss penalty** is how long it takes to access main memory after a miss.
  - On cache hit: access time = hit time
  - On cache miss: access time = hit time + miss penalty
- The **Average Memory Access Time** (or **AMAT**) is the **average** amount of time it takes for **one memory access**, given a program.

Computing AMAT (1/2)

- If our hit rate was ¾ (in yesterday's example), a hit takes 10 cycles, and a miss takes 100 cycles, what speedup did our cache provide?
- First step: find new AMAT
    - **¾** of our memory accesses were **hits**, and therefore took **10 cycles each**
    - **¼** of our memory accesses were **misses**, and therefore took 100 + 10 = **110 cycles each**
    - The average time for **4 memory accesses** is 3*(hit time) + 1*(miss time) = **140 total cycles**
    - AMAT (for **1 memory access**) is 140/4 = **35 cycles**
- Our original AMAT would have been 100 cycles per access
    - Speedup is 100/35 = **2.86x**

Computing AMAT (2/2)

- Alternate formula:

AMAT = hit time + (miss rate * miss penalty)

  - AMAT = 10+(¼*100) = **35 cycles**
  - Speedup is 100/35 = **2.86x**

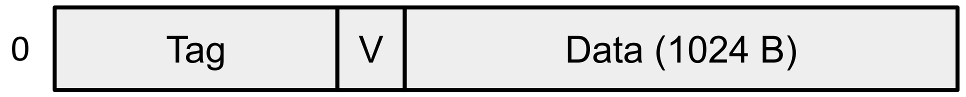
Computing Hit Rate

- Computing AMAT can get tricky for certain programs
    - Need to determine the hit rate of a program
- General Tips:
    - Typically hit rate starts low, because the cache is cold
    - After warming up, the cache normally reaches a "steady state", where a pattern of hits and misses appears every iteration of some outer loop, until we completely exhaust the memory in a block
    - Once we start a new block, the cache acts cold again, and the cycle repeats.
    - Once you find this cycle, it's possible to use that to get the total hit rate.
- More practice: Homework

Optimizing Cache Hardware: Block Size

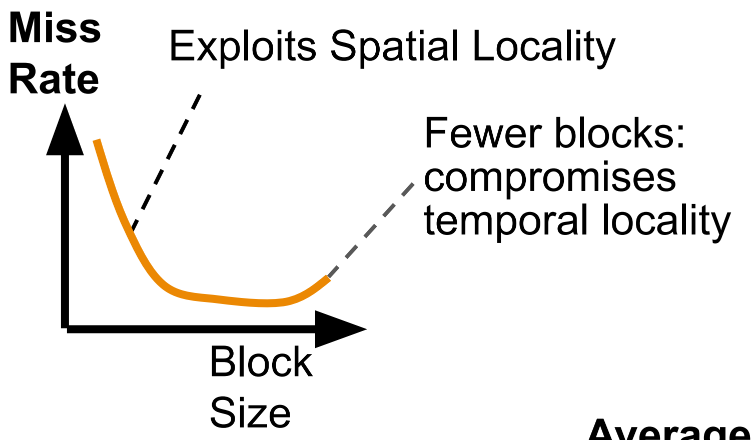
- Benefits of Larger Block Size
    - **Spatial locality**: if we access a given word, we're likely to access other nearby words soon
    - Reduces compulsory misses
    - Works well for programs
    - Works well for sequential array accesses
- Drawbacks of Larger Block Size
    - Larger block size means larger miss penalty
        - On a miss, takes longer time to load a new block from next level
    - Effect on miss rate?
        - Next slide →

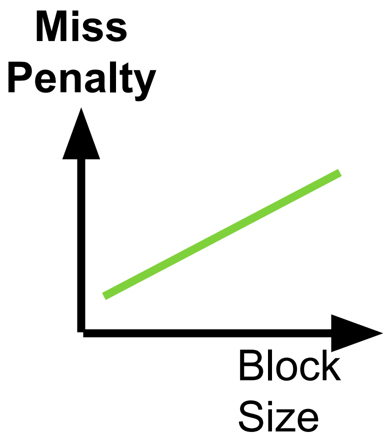
Extreme Example: One Big Block

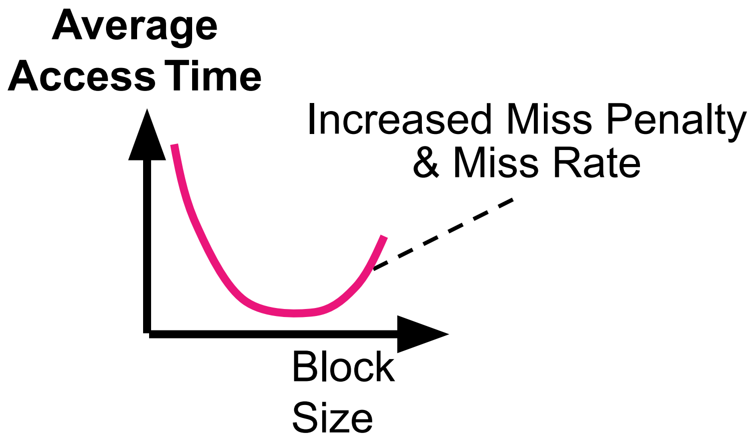


- Cache Size = 1024 bytes, Block Size = 1024 bytes
  - Only **ONE** entry (row) in the cache!
- If item accessed, likely accessed again soon
  - But unlikely will be accessed again immediately!
- The next access will likely to be a miss again
  - Continually loading data into the cache but discard data (force out) before it gets reused
  - Nightmare for cache designer: **Ping Pong Effect**
  - Increases conflict misses

Block Size Tradeoff Conclusions







Cache Optimizations: Hardware

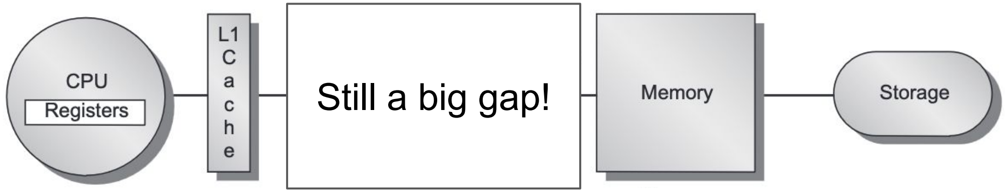
- Larger block size
  - + reduces compulsory misses
  - - increases conflict misses and miss penalty
- Larger cache size
  - + reduces capacity misses
  - - increases hit time
- Higher associativity
  - + reduces conflict misses
  - - increases hit time

Cache Optimizations: Software

- If **compulsory misses** are most common
  - Prefetching
- If **capacity misses** are most common
  - Using too much memory: reduce the "working set"
    - Split code into parts that access only as much memory as your cache can store
- If **conflict misses** are most common
  - Reduce the number of distinct chunks of memory you're accessing
  - E.g. By transposing your B matrix in matrix multiply

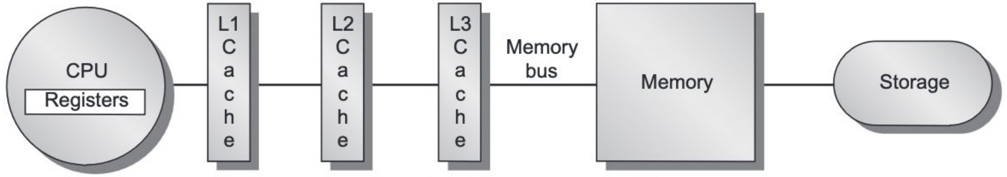
Multilevel Caches

Multilevel Caches (1/2)



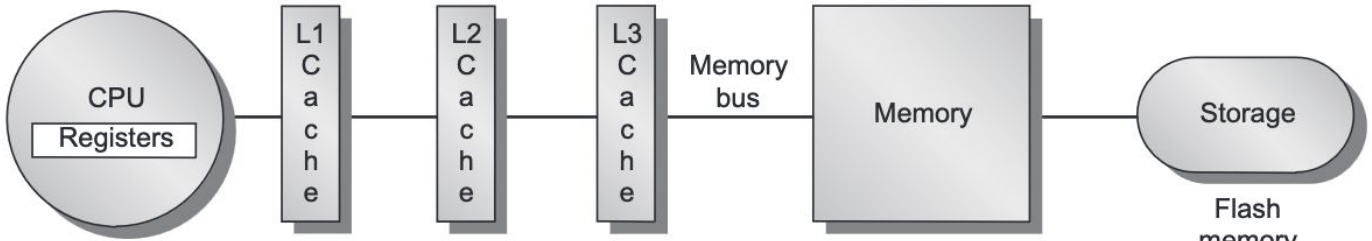
- Current setup: If cache hit, then fast access. If cache miss, then slow access
- Problem:
  - Making the cache bigger/more complex increases hit rate, but also increases hit time
  - With just one cache, we need to make some tough trade-offs.

Multilevel Caches (2/2)



- Solution: Add **multiple layers of caches**.
    - Check L1 cache. If hit, we're done
    - If miss, check L2 cache. If hit, we're done, and bring data up to L1 cache
    - If miss, check L3 cache. If hit, we're done, and bring data up to L1 and L2 cache
    - …
- Each level of cache is larger than the previous and holds all of the data in the previous cache
- Library analogy: We buy a bookshelf for right next to us, then we buy a larger bookshelf to put in our garage, then make a shed to store more books
    - Slight difference: to act exactly like a cache, each shelf/shed would need a copy of each book stored in the inner level.

Multilevel Cache Sizes/Latencies

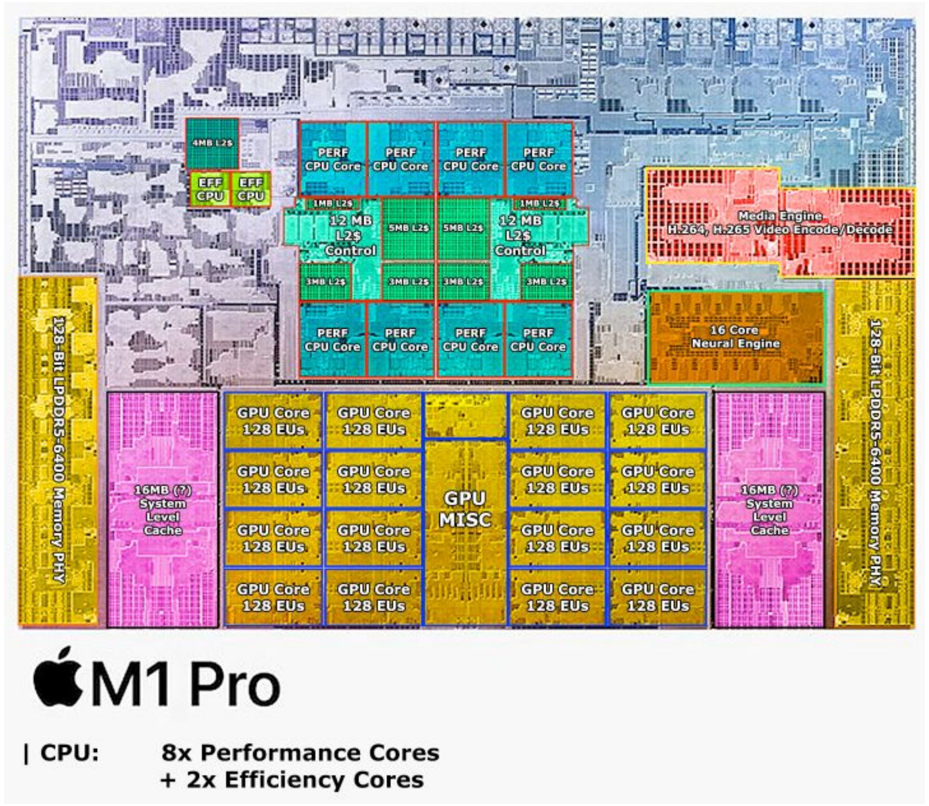


		Register reference	Level 1 Cache reference	Level 2 Cache reference	Level 3 Cache reference	Memory reference	Flash memory reference
Laptop	Size:	1000 bytes	64 KB	256 KB	4-8 MB	4–16 GB	256 GB-1 TB
	Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	50-100 uS
Desktop	Size:	2000 bytes	64 KB	256 KB	8-32 MB	8–64 GB	256 GB-2 TB
	Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	50-100 uS

From "Computer Architecture: A Quantitative Approach" (also by P&H)

Multilevel Cache Organization

- Generally, the L1 cache is small, but attached to the core
  - Each core gets its own L1 cache
  - Fast access, but also fairly small
  - Split into instruction and data caches
- L2 cache is larger and slightly slower, but usually a bit farther away
  - Sometimes shared, sometimes on the core.


4MB L2$
EFF CPU
EFF CPU
PERF CPU Core
1MB L2$
12 MB L2$ Control
5MB L2$
3MB L2$
Media Engine H.264, H.265 Video Encode/Decode
16 Core Neural Engine
128-Bit LPDDR5-6400 Memory PHY
16MB (?) System Level Cache
GPU Core 128 EUs
GPU MISC
M1 Pro
| CPU: 8x Performance Cores + 2x Efficiency Cores

Multilevel Cache Hit/Miss Times

- Using the numbers from the previous slide as examples:
  - L1 hit time = 1 ns
  - L2 hit time = 1 ns + 10 ns = 11 ns
  - L3 hit time = 1 ns + 10 ns + 20 ns = 31 ns
  - L3 miss time = 1 ns + 10 ns + 20 ns + 100 ns = 131 ns
- Hit/miss rate is calculated only for accesses that "reach" that cache
  - L2 hit rate is % of L2 accesses that hit, so L1 hits don't count as L2 hits OR L2 misses.

AMAT in Multilevel Caches

- Let's say we have a memory system with the following properties. What would be the AMAT of this system?

- 50% of accesses are L1 hits
  - 50% of accesses take 1ns
- 50% of accesses are L1 misses
  - 75% of this 50% are L2 hits (37.5% of total)
  - 87.5% of accesses are now accounted for
- 12.5% of accesses are L2 misses
  - 80% of 12.5% = 10% are L3 hits
  - 97.5% of accesses are now accounted for
- 2.5% of accesses are L3 misses
  - And therefore access DRAM

- Total: 1 ns + 0.5*10 ns + 0.125 * 20 ns + 0.025 * 100 ns

- 11 ns access time ≈ 9x speedup.

	Hit time	Hit rate
L1 Cache	1 ns	50%
L2 Cache	10 ns	75%
L3 Cache	20 ns	80%
DRAM	100 ns	100%

AMAT in Multilevel Caches

- Another way to think about this:

AMAT = hit time + miss rate * **miss penalty**

miss penalty = AMAT of next level

1 ns + 0.5*(10 ns + 0.25*(20 ns + 0.2*100 ns))

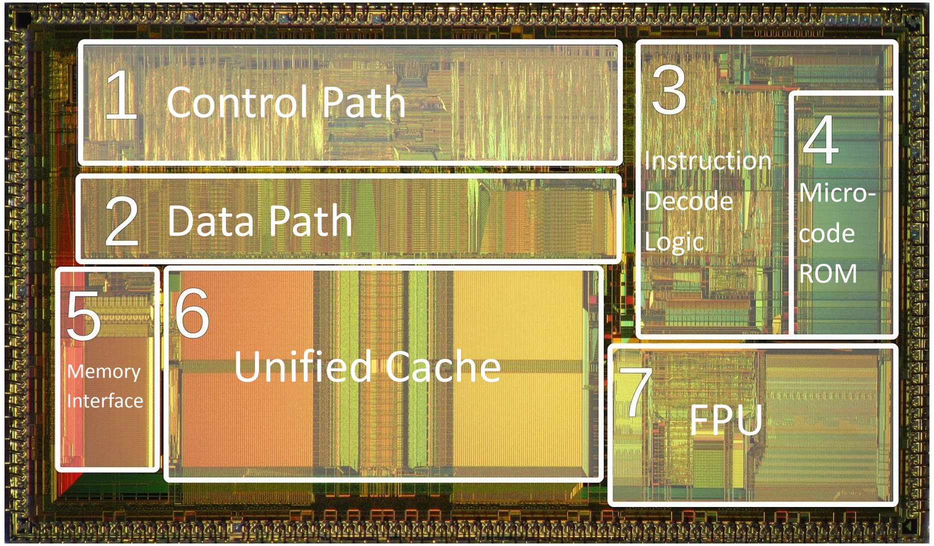
= 11 ns also!

	Hit time	Hit rate
L1 Cache	1 ns	50%
L2 Cache	10 ns	75%
L3 Cache	20 ns	80%
DRAM	100 ns	100%

Caches (and More!) in Real CPUs

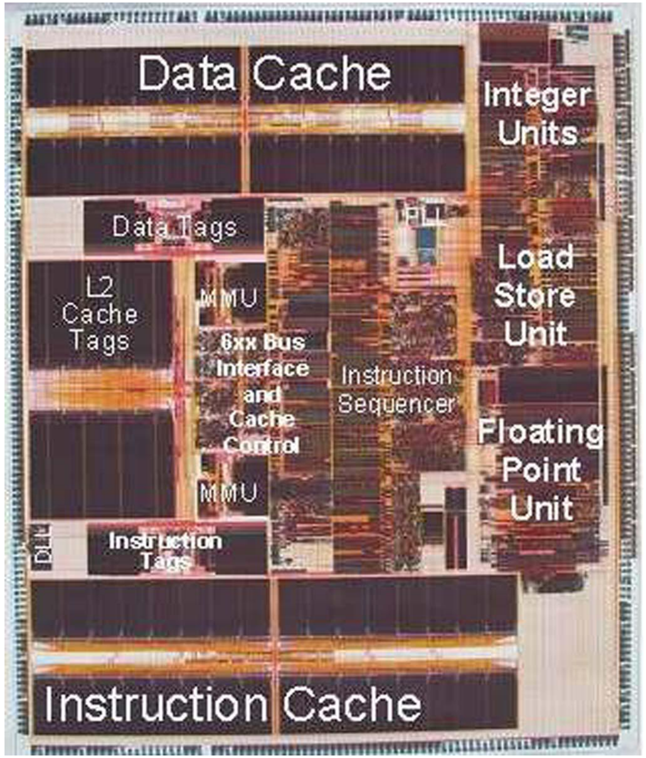
Intel 486 (1989)

- First major CPU to include on-chip cache!
- **8KiB unified L1**
- Compatible with additional external cache (acted like L2)
  - These existed before on-chip caches
- <10 cycle memory latency
  - 20-33 MHz clock frequency
- 5-stage pipelining
- Chief architect: Pat Gelsinger.



Early PowerPC (2001)

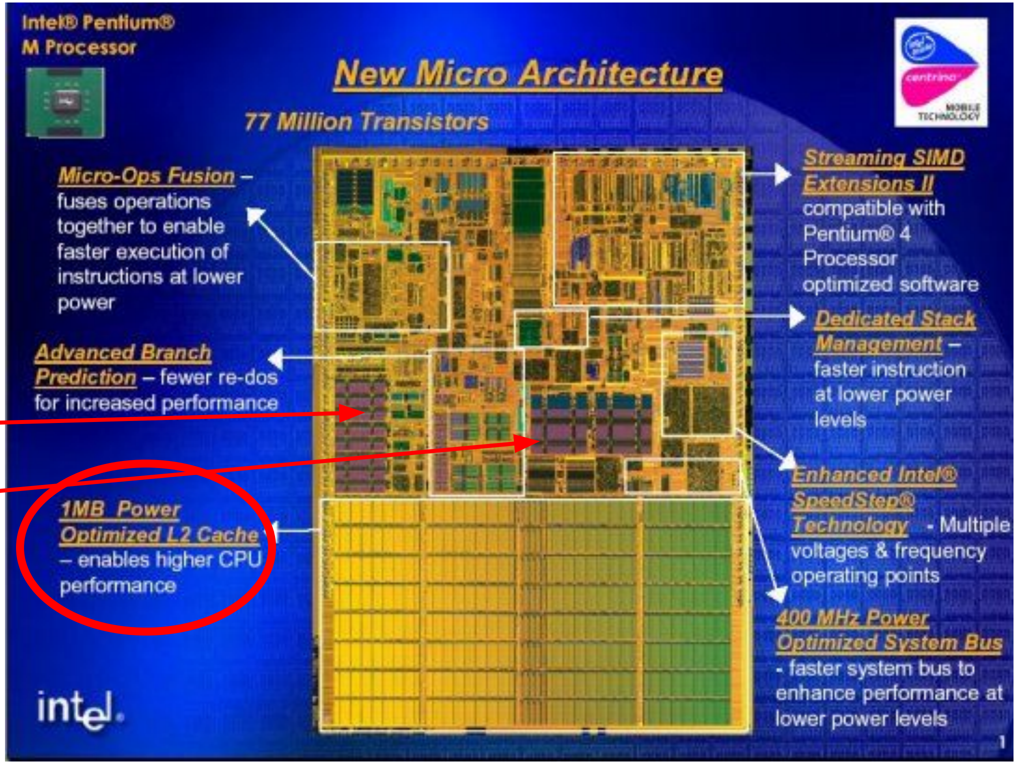
- Cache
  - **32 KiB Instruction L1 & 32 KiB Data L1 caches**
  - **External L2 cache interface** with integrated controller and cache tags, supports up to 1 MiB external L2 cache
  - Dual Memory Management Units (MMU) with Translation Lookaside Buffers (TLB)
    - ■ TLBs are used for virtual memory (next unit)
- Pipelining
  - Superscalar (3 inst/cycle)
  - 6 execution units (2 integer and 1 double precision IEEE floating point)



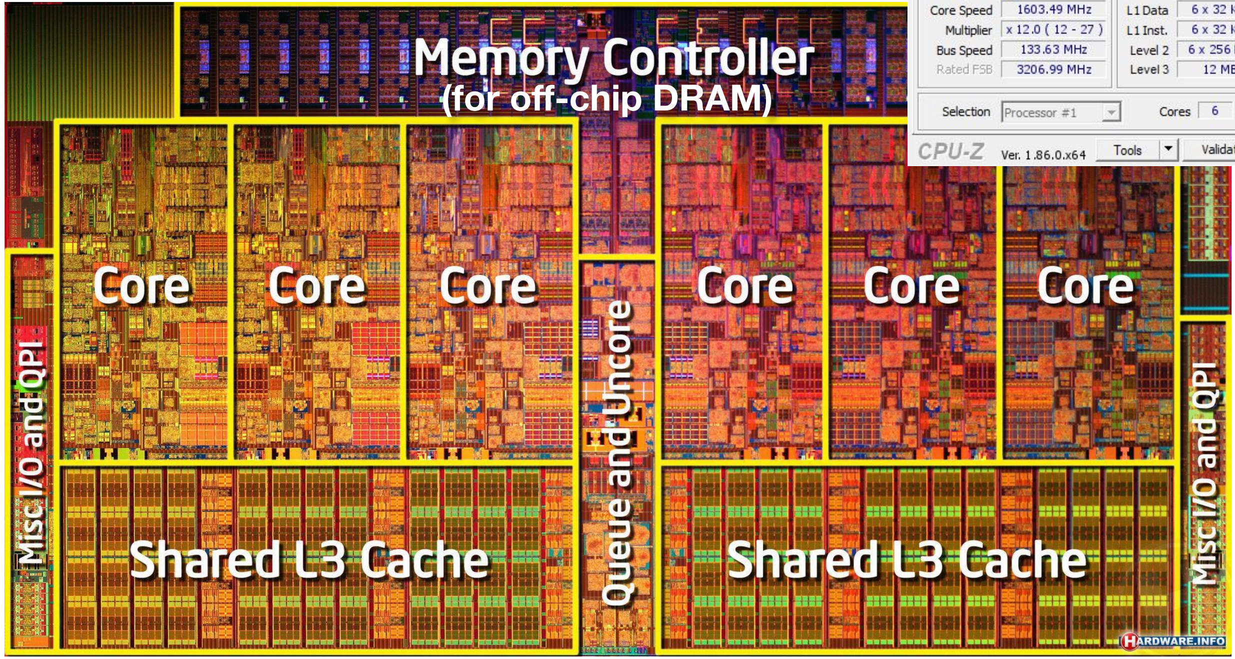
Intel Pentium M (2003)

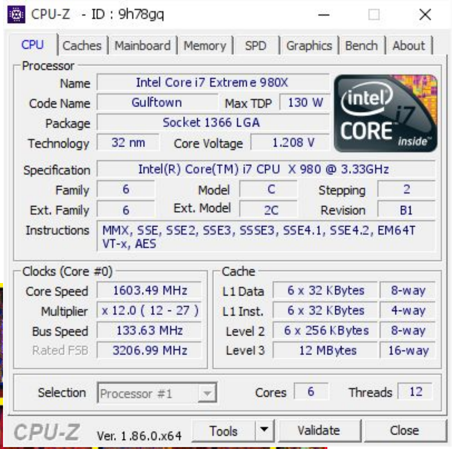
32KiB L1 I$

32KiB L1 D$


Intel® Pentium® M Processor
New Micro Architecture
77 Million Transistors
Micro-Ops Fusion – fuses operations together to enable faster execution of instructions at lower power
Advanced Branch Prediction – fewer re-dos for increased performance
1MB Power Optimized L2 Cache – enables higher CPU performance
Streaming SIMD Extensions II compatible with Pentium® 4 Processor optimized software
Dedicated Stack Management – faster instruction at lower power levels
Enhanced Intel® SpeedStep® Technology - Multiple voltages & frequency operating points
400 MHz Power Optimized System Bus - faster system bus to enhance performance at lower power levels
centrino MOBILE TECHNOLOGY
intel.

Intel Core i7 (2010)





CPU-Z - ID : 9h78gq
CPU | Caches | Mainboard | Memory | SPD | Graphics | Bench | About
Tools
Validate
Close

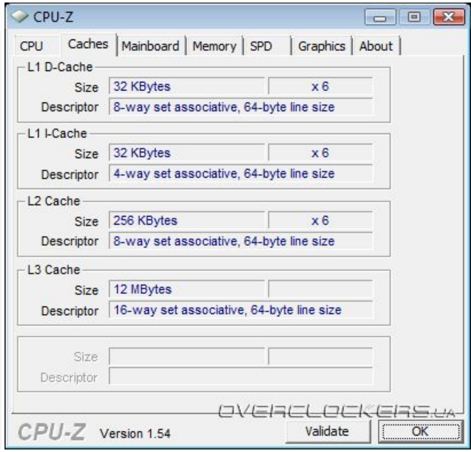
Processor

Name	Intel Core i7 Extreme 980X				
Code Name	Gulftown	Max TDP	130 W		
Package	Socket 1366 LGA				
Technology	32 nm	Core Voltage	1.208 V		
Specification	Intel(R) Core(TM) i7 CPU X 980 @ 3.33GHz				
Family	6	Model	C	Stepping	2
Ext. Family	6	Ext. Model	2C	Revision	B1
Instructions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T VT-x, AES				

Clocks (Core #0)		Cache		
Core Speed	1603.49 MHz	L1 Data	6 x 32 KBytes	8-way
Multiplier	x 12.0 (12 - 27)	L1 Inst.	6 x 32 KBytes	4-way
Bus Speed	133.63 MHz	Level 2	6 x 256 KBytes	8-way
Rated FSB	3206.99 MHz	Level 3	12 MBytes	16-way

Selection Processor #1 Cores 6 Threads 12

CPU-Z Ver. 1.86.0.x64



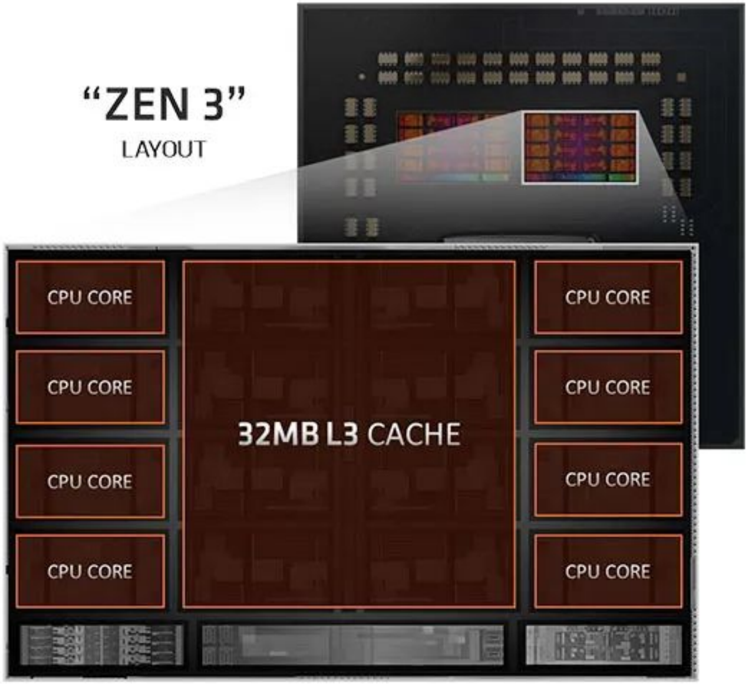
L1 D-Cache	Size	32 KBytes	x 6
	Descriptor	8-way set associative, 64-byte line size	
L1 I-Cache	Size	32 KBytes	x 6
	Descriptor	4-way set associative, 64-byte line size	
L2 Cache	Size	256 KBytes	x 6
	Descriptor	8-way set associative, 64-byte line size	
L3 Cache	Size	12 MBytes	
	Descriptor	16-way set associative, 64-byte line size	
	Size		
	Descriptor		

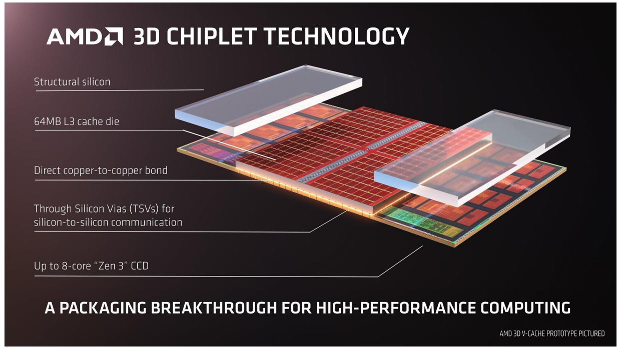
CPU-Z Version 1.54

- 6 cores, 12 threads
  - Hyperthreading
- Per core:
  - **32 KiB L1 I$, 32KiB L1 D$**
  - **256 KiB L2**
- **12 MiB shared L3 cache**

AMD Zen 3 (2020)

- "Chiplet" architecture enables multiple dies to be fabricated and integrated on one CPU
- Each chiplet has its own **32 MiB L3 cache**
  - Each core still has its own L1 and L2 caches
- "3D stacking" enables **up to 96 MiB L3 cache**!




AMD 3D CHIPLET TECHNOLOGY
Structural silicon
64MB L3 cache die
Direct copper-to-copper bond
Through Silicon Vias (TSVs) for silicon-to-silicon communication
Up to 8-core "Zen 3" CCD
A PACKAGING BREAKTHROUGH FOR HIGH-PERFORMANCE COMPUTING
AMD 3D V-CACHE PROTOTYPE PICTURED

Cache Coherence

Caching with multithreading (1/2)

- Each core has its own L1 cache, and sometimes its own L2 cache
- Most caches are write-back, so it doesn't update main memory until the block gets evicted
- If multiple threads are running with shared memory, how do we guarantee that the correct version of data is being used?

Caching with multithreading (2/2)

- Let’s look at the potentially problematic scenarios:
- Simultaneous reads:
  - Should be allowed; as long as no one changes data, things are safe
- Simultaneous writes:
  - Once two different cores have conflicting data in their caches, it's difficult to return to a reasonable state
  - Example: git merge conflicts
  - Therefore, we should not allow these to happen
  - Only one thread should be able to write to a block at a time!
- Simultaneous read and write:
  - Get the read the new value after writing

MSI Protocol (1/2)

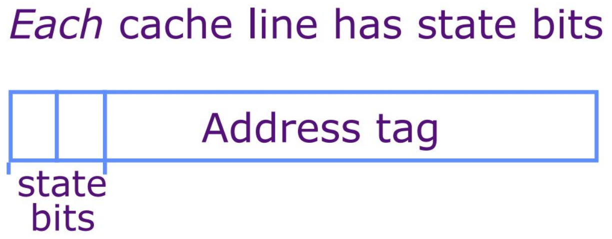
- Recall: In our cache, we keep track of whether a line is valid or not, and whether it’s dirty or not
- Can we use this metadata to handle cache coherence?
- On a read:
  - Check if any other core has its dirty bit set for that block; tell that core to write back its data
- On a write:
  - If that block is valid in any other core, invalidate it (and if dirty, write back)
- Requirement: Set up a system where you can "**snoop**" on other caches to see if they have a cache block with the same tag.

MSI Protocol (2/2)

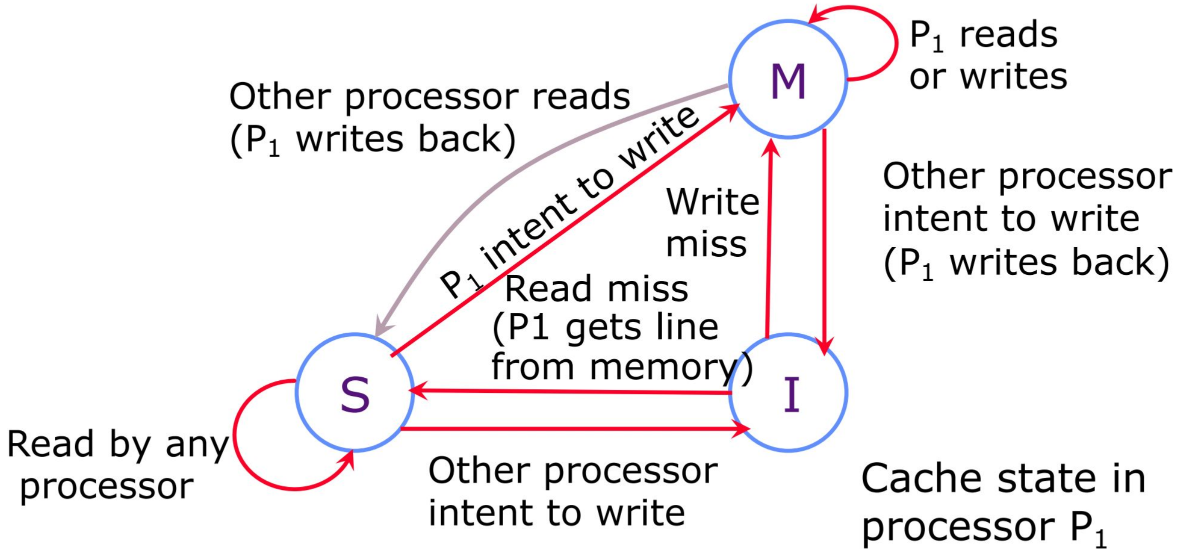
- **Invalid:** Same as valid bit off in regular cache; the block isn't in the cache
- **Shared:**
  - The block is in some other cache
  - We haven't modified this block, and we're not allowed to make modifications
    - If we need to make modifications, evict the block from everyone's cache and move to modified
  - Allows for simultaneous reads
- **Modified:** Same as valid bit on, dirty bit on in regular cache
  - The block has been read, and modified
  - Further, no other cache has this block.

Credit: CS 152 slides

MSI State Transitions



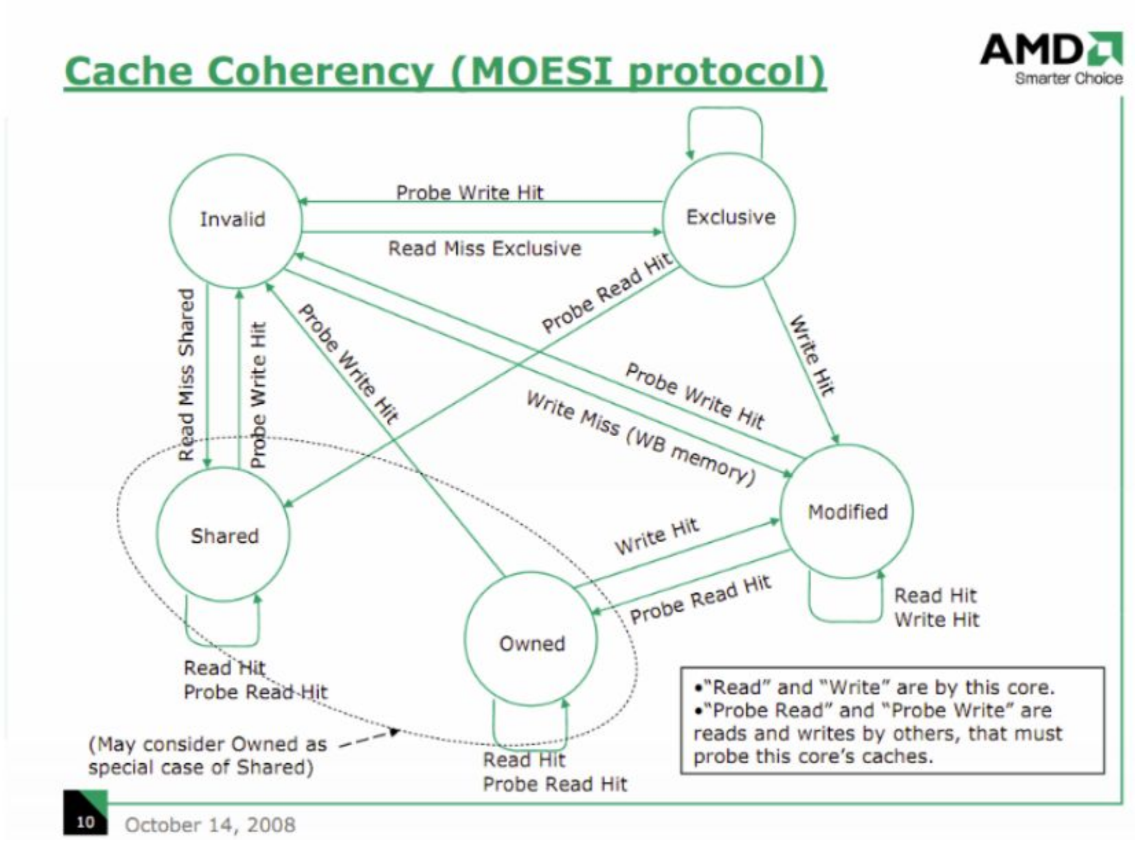
M: Modified
S: Shared
I: Invalid


M
S
I
P_1 reads or writes
Other processor reads (P_1 writes back)
P_1 intent to write
Write miss
Other processor intent to write (P_1 writes back)
Read miss (P1 gets line from memory)
Read by any processor
Other processor intent to write
Cache state in processor P_1

MOESI

- Adds two more states:
  - Can read then write without traffic to other cores
- **Exclusive:** Same as valid bit on, dirty bit off in regular cache
  - The block has been read, but not modified
  - Further, no other cache has this block
- **Owner:**
  - The block is in some other cache
  - If we make modifications, it's our responsibility to tell all the other caches in shared state about these changes
  - Allows for writing while other threads read the same data.

MOESI State Transitions



Cache Coherency (MOESI protocol)

AMD
Smarter Choice

•"Read" and "Write" are by this core.
•"Probe Read" and "Probe Write" are reads and writes by others, that must probe this core's caches.

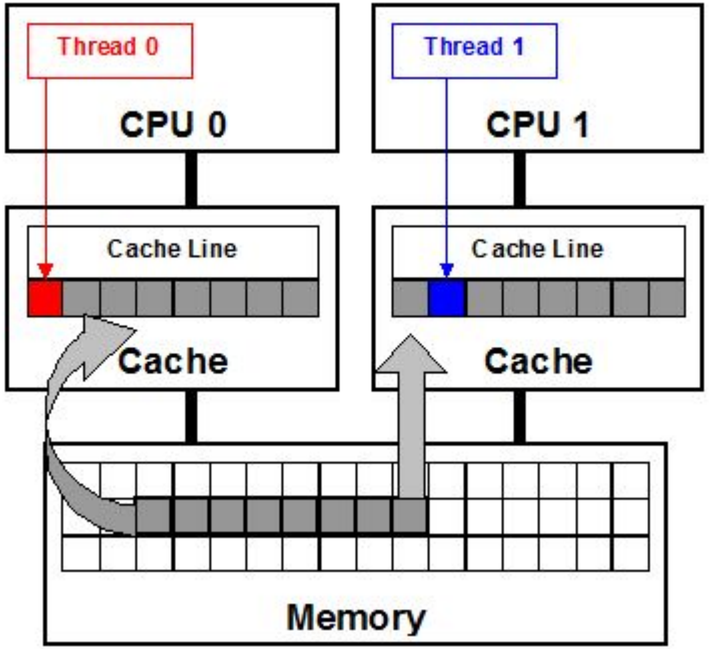
10 October 14, 2008

Coherence Misses

- **Coherence misses**: misses caused by blocks invalidated due to other cores
- Since only one thread is allowed to have a dirty block at a time, two threads writing to the same block cause a lot of coherence misses
  - This is why the interleaved pragma omp was much slower than the blocked pragma omp
  - Another form of **thrashing**
- Therefore, when writing multithreaded code, we want to have each thread work on a separate block as much as possible
  - But also close enough so that the shared L3 cache gets hits.

False Sharing

- Coherence misses due to interleaving are an example of **false sharing**
- The entire block is invalidated and must be reloaded, even though technically no data is shared.



https://www.codeproject.com/Articles/85356/Avoiding-and-Identifying-False-Sharing-Among-Threa

In conclusion…

- We’ve discussed memory caching in detail. Caching in general shows up over and over in computer systems
    - Filesystem cache, Web page cache, Game databases / tablebases, Software memoization, Others?
- Big idea: if something is expensive but we want to do it repeatedly, do it once and cache the result.
- Cache design choices:
    - Block size
    - Size of cache: speed v. capacity
    - Associativity (direct-mapped v. set v. fully associative)
    - Replacement policy (LRU, MRU, FIFO, …)
    - Write Policy (write through v. write back)
    - Extra:
        - Multilevel cache
        - Cache coherence protocol
- Understanding caches allows us to make correct hardware and software design choices, depending on programs, technology, budget, etc.
- Next week: upending everything you (think you) know about memory :)