Introduction to Caches

Computer Architecture

Wei Wang

Text Book Chapters

- “Computer Organization and Design,” Chapter 5.3 and 5.4.
- “Computer Architecture: A Quantative Approach,” Appendix B.

Road Map

- Overview of Caches
- Caching Basics
- Cache Performance Equations

Overview of Caches

CPU vs Memory Performance

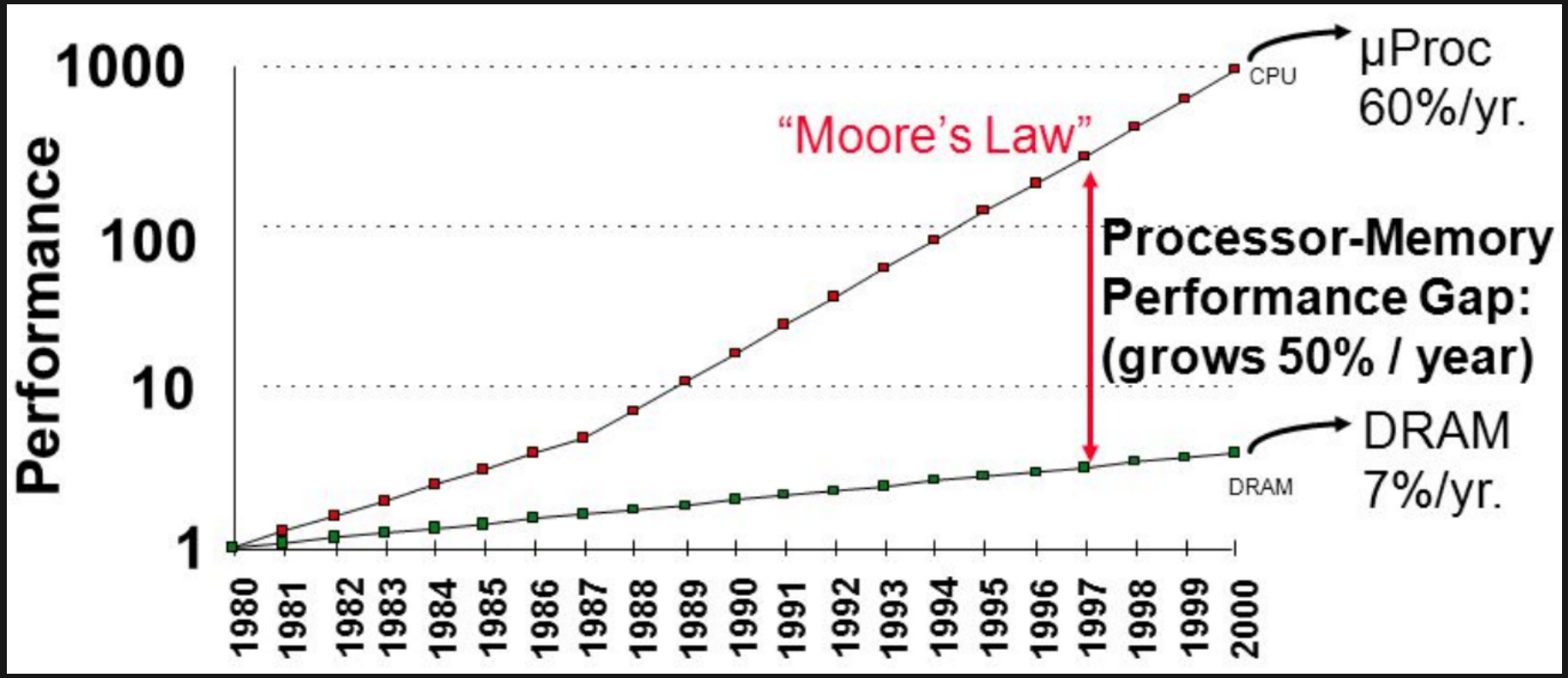


Figure: CPU vs memory performance growth

Memory Hierarchy Design

- Until now we have assumed a very ideal memory
  - All memory accesses take 1 cycle
- Assumes an unlimited size, very fast memory
  - Fast memory is very expensive
  - Large amounts of fast memory would be slow!
- Tradeoffs
  - Cost-speed and size-speed
- Solution:
  - Smaller, faster expensive memory close to core “cache”
  - Larger, slower, cheaper memory farther away

Caches

- Cache is a type of small, fast storage used to improve average access time to slow memory
- Hold a copy of the subset of the instructions and data used by program
- Exploits spacial and temporal locality

Caching is Everywhere

- In computer architecture, almost everything is a cache!
  - Registers are “a cache” on variables – software managed
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)
  - Translation-lookaside Buffer (TLB) a cache on page table
  - Branch target buffer a cache on branch targets.

Common Cache Hierarchy

- Most processors today have three levels of caches.
  - One major design constraint for caches is their physical sizes on CPU die. Limited by their sizes, we cannot have too many caches.
  - Some high-performance and/or embedded processors have L4 caches, which, however, use DRAM cells instead of common SRAM cells.
- L1 Cache
  - L1 caches are closest to the CPU computation logics.
  - To avoid pipeline structure hazards, L1 caches are usually partitioned into data cache (D-cache) and instruction cache (I-cache).
  - L1 caches are mostly 64KB per core, as bigger caches are too slow to access and will be physically too far away from computation logics.

Common Cache Hierarchy cont'd

- L2 caches
  - L2 caches are slightly further away from CPU. Thus L2 caches are slower than L1 caches, but can have larger storage space.
  - Typically, L2 caches are unified (for both data and instructions).
- L3 caches
  - L3 caches are the largest caches and typically aim at holding all of a working set in it.
  - L3 caches usually are located out side CPU core's die due to its bulky size.

Example: Skylake Due-core CPU

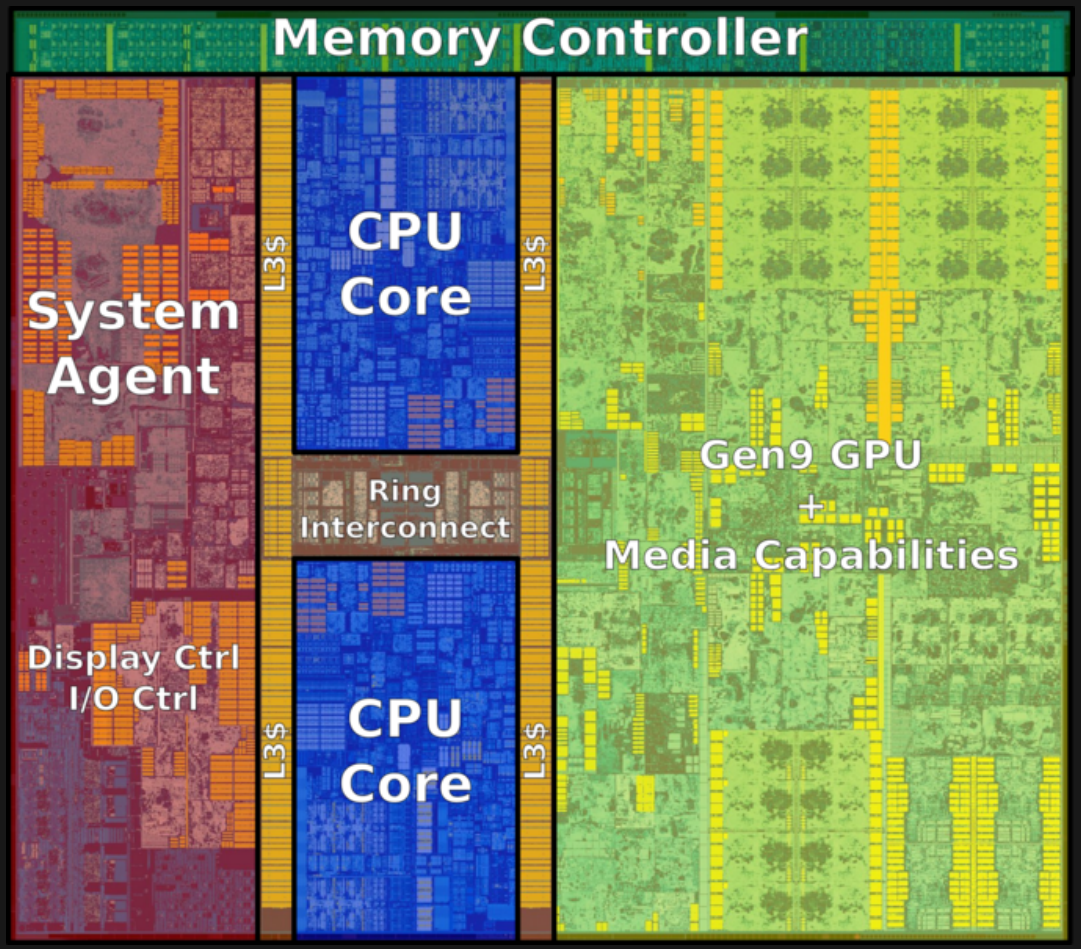


Figure: Floorplan for a Due-core Skylake CPU (figure from Wikichip.org)

Example: Skylake Due-core CPU cont'd

Figure: Floorplan for a Skylake CPU core (figure from Wikichip.org)

Memory Hierarchy Specs

Type	Capacity	Latency	Approx. Bandwidth
Register	<2KB	1ns	150GB/s
L1 Cache	<64KB	4ns	100GB/s
L2 Cache	<8MB	10ns	50GB/s
L3 Cache	<64MB	20ns	32GB/s
Memory	<4GB per rank	100ns	10GB/s
SSD	>1GB	5ms	300MB/s
HDD	>1GB	10ms	10MB/s

Program Locality is Why Caches Work

- Memory hierarchy exploit program locality:
  - Programs tend to reference parts of their address space that are local in time and space
    - Temporal locality: recently referenced addresses are likely to be referenced again (reuse)
    - Spatial locality: If an address is referenced, nearby addresses are likely to be referenced soon
- Programs that don't exploit locality won't benefit from caches
  - Machine-learning applications typically have low data locality.
  - These programs are called streaming programs. They are the main focus of today's system research.

An Example of Locality

```
1       j = val1;
2       k = val2;
3       for (i=0; i<10000;i++) {
4         A[i] += j;
5         B[i] += k;
6       }
```

- Data Locality: i, A, B, j, k?
  - i: reused and updated for all iterations; hence temporally local;
  - j and k: reused and stay constant for all iterations; hence temporally local;
  - A and B: if $A[i]$ and $B[i]$ is accessed, $A[i+1]$ and $B[i+1]$ will soon be accessed; hence spatially local.
- Instruction Locality?
  - The same loop body is executed over all iterations; hence temporally local.

Terminologies

- Lower levels in the hierarchy are closer to the CPU
  - L1-caches are right outside the CPU.
  - L2-caches usually surround the CPU cores.
  - L3-caches usually are located outside the physical die of the CPU cores (the uncore part of CPU).
- At each level a block is the minimum amount of data whose presence is checked at each level
  - Blocks are also often called cache lines or simply lines.
  - Block size is always a power of 2.
  - Contemporary processors usually have a cache line of 64 bytes.

Terminologies cont'd

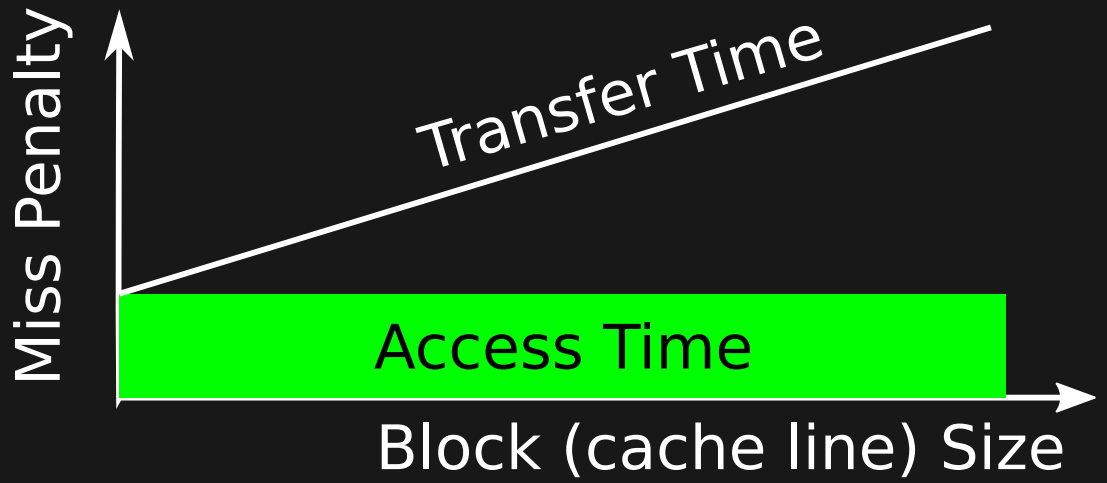
- A reference is said to hit at a particular level if the cache line is found at that level. A reference is said to miss at a particular level if the cache line is NOT found at that level.
  - Hit rate (HR): $Hit_{Rate} = \frac{Hits}{References}$
  - Miss rate (MR): $Miss_{Rate} = \frac{Misses}{References}$
- Access time of a hit is the hit time
- The additional time to fetch a block on a miss is called the miss penalty.
  - If there is a cache miss, then data has to be fetched from higher level of caches or memory.
  - If the cache miss happens at level x, then data will be fetched from caches with levels larger than x or from memory. The data will be stored in the level-x cache after fetching.
  - Since a cache miss can take a long time, pipeline may be stalled during a cache miss.

Terminologies cont'd

- Miss penalty = Access Time + Transfer time
- Access time is a function of latency
  - "Time for memory to get request and process it"
- Transfer time is a function of bandwidth
  - "Time for all of a block to get back"
  - Even if the data to be accessed is only 1 byte, a whole cache line is fetched into cache. This is based on the expectation of spatial locality. That is, if one byte is accessed, the next byte is likely to be accessed soon.

Terminologies cont'd

- Access time is typically constant (i.e., the latency), while the transfer time depends on the size of a cache line.



Access Time vs. Transfer Time

- Time decomposition for a cache miss to the memory.
  - t_1: time to send memory request (i.e., address) to the memory device
  - t_2: time for the memory device to locate the data
  - t_3: time for sending the data block back to cache
- Access time = $t_1 + t_2$
- Transfer time = t_3
- Cache miss penalty = $t_1 + t_2 + t_3$

Latency vs. Bandwidth

There is an old network saying: Bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed. – David Clark, MIT

- Latency is about the time for completing one task.
- Bandwidth is about the number of tasks that can be done within a time window.
  - Bandwidth can be increased by simply having more workers working on multiple tasks simultaneously.
  - Bandwidth is equally affected by latency and parallelization.
- With memory, for bandwidth we can:
  - Wider buses, larger block sizes, more DRAM channels
- Latency is still much harder:
  - Have to get request from cache to memory (off-chip)
  - Have to do memory lookup
  - Have to have bits travel on wire back on-chip to cache

Caching Bascis

Caching Basics

- Most basic caching questions:
  - How do we know if a piece of data is in the cache?
  - If it is, how do we find it?
  - If it isn't, how do we get it?

More Detailed Questions

- Cache line placement policy?
  - Where does a cache line go when it is fetched?
- Cache line identification policy?
  - How do we find a cache line in the cache?
- Cache line replacement policy?
  - When fetching a cache line into a full cache, how do we decide what other cache line gets kicked out?
- Write strategy?
  - Does any of this differ for reads vs. writes?

General View of Caches

- Cache is made of frames
  - Frame = data + tag + state bits
  - Tag is the memory address of currently stored cache line
  - State bits: Valid bit (has valid data in frame?), Dirty (data is written?)
- Cache line matching algorithm
  - Find frame(s)
  - If (incoming tag != stored tag) then a cache miss occurs
    - Evict cache line currently in frame
    - Read requested data from memory or higher level of caches.
    - Replace with cache line read from memory or higher level of caches.
- Return appropriate word within cache line

Simple Cache Example

- Direct-mapped cache: Each cache line has a specific spot in the cache.
  - That is, if the cache line is in the cache, only one slot for it based on its tag (memory address).
- Makes cache line placement, ID, and replacement policies easy
  - Cache line placement
    - It goes to its one assigned slot based on its tag (address).
  - Cache line identification:
    - We look at the tag for that one assigned slot
  - Cache line replacement: What gets kicked out?
    - Whatever is in its assigned spot
  - Write strategy:
    - “Allocate on write” (more on write strategies later)

Simple Cache Example cont'd

- 4 locations in our cache
- Block Size = 1 byte
- Data reference stream:
- References to memory addresses (Tags):
  - 0, 1, 2, 3, 4, 5, 2, 3, 7
- Tag to cache slot mapping
  - $slot = address \% cache\_size$

Slot	tag	valid	data
0			
1			
2			
3			

Table: 4-slot cache layout

Simple Cache Example cont'd

- Initially, the cache is empty. So all slots are invalid (valid bit is 0).

Slot	tag	valid	data
0		0	
1		0	
2		0	
3		0	

Table: 4-slot cache layout

Simple Cache Example cont'd

- Read data at address 0.
  - Data should be mapped to slot $0\%4 = 0$.
  - Slot 0 is invalid, data is not in cache, a cache miss occurred.
  - Read data from memory, and store data in slot 0.

Slot	tag	valid	data
0	0	1	data@addr 0
1		0	
2		0	
3		0	

Table: 4-slot cache layout

Simple Cache Example cont'd

- Read data at address 1.
  - Data is not in cache, a cache miss occurred.
  - Data is mapped to slot $1 \% 4 = 1$.

Slot	tag	valid	data
0	0	1	data@addr 0
1	1	1	data@addr 1
2		0	
3		0	

Table: 4-slot cache layout

Simple Cache Example cont'd

- Read data at address 2.
  - Data is not in cache, a cache miss occurred.
  - Data is mapped to slot $2\%4 = 2$.

Slot	tag	valid	data
0	0	1	data@addr 0
1	1	1	data@addr 1
2	2	1	data@addr 2
3		0	

Table: 4-slot cache layout

Simple Cache Example cont'd

- Read data at address 3.
  - Data is not in cache, a cache miss occurred.
  - Data is mapped to slot $2\%4 = 3$.

Slot	tag	valid	data
0	0	1	data@addr 0
1	1	1	data@addr 1
2	2	1	data@addr 2
3	3	1	data@addr 3

Table: 4-slot cache layout

Simple Cache Example cont'd

- Read data at address 4.
  - Data should be mapped to slot $4\%4 = 0$.
  - Slot 0 has tag 0. Thus, data from address 4 is not in cache, a cache miss occurred.
  - Since data at address 0 is in the slot 0, it is evicted.

Slot	tag	valid	data
0	4	1	data@addr 4
1	1	1	data@addr 1
2	2	1	data@addr 2
3	3	1	data@addr 3

Table: 4-slot cache layout

Simple Cache Example cont'd

- Read data at address 5.
  - Data is not in cache, a cache miss occurred.
  - Data is mapped to slot $5\%4 = 1$.
  - Since data at address 1 is in the slot 1, it is evicted.

Slot	tag	valid	data
0	4	1	data@addr 4
1	5	1	data@addr 5
2	2	1	data@addr 2
3	3	1	data@addr 3

Table: 4-slot cache layout

Simple Cache Example cont'd

- Read data at address 2.
  - Data should be mapped to slot $2\%4 = 2$.
  - Slot 2 has data from address 2, a cache hit

Slot	tag	valid	data
0	4	1	data@addr 4
1	5	1	data@addr 5
2	2	1	hit!=>data@addr 2
3	3	1	data@addr 3

Table: 4-slot cache layout

Simple Cache Example cont'd

- Read data at address 3.
  - Data should be mapped to slot $3\%4 = 3$.
  - Slot 3 has data from address 3, a cache hit

Slot	tag	valid	data
0	4	1	data@addr 4
1	5	1	data@addr 5
2	2	1	data@addr 2
3	3	1	hit!=>data@addr 3

Table: 4-slot cache layout

Simple Cache Example cont'd

- Read data at address 7.
  - Data should be mapped to slot $7\%4 = 3$.
  - Slot 3 does not have data from address 7, a cache miss occurred.
  - Since data at address 3 is in the slot 3, it is evicted.

Slot	tag	valid	data
0	4	1	data@addr 4
1	5	1	data@addr 5
2	2	1	data@addr 2
3	7	1	data@addr 7

Table: 4-slot cache layout

Cache Performance Equations

Hit Rate and Miss Rate

- Miss Rate: the percentage of data accesses are cache misses.
- Hit Rate: the percentage of data accesses are cache hits.
- For the example in slide 26, there are 9 accesses in total, and 2 of them are hits.
  - The hit rate is then $2/9 = 22.2\%$.
  - The miss rate is then $7/9 = 77.8\%$.

Average Memory Access Time

- Average memory access time (AMAT) is used to represent the average memory latency for a series of memory accesses.
- $AMAT = Latency_{hit} \times Rate_{hit} + Latency_{miss} \times Rate_{miss}$
  - Typically, $Latency_{miss} = Latency_{hit} + Miss\_Penalty$.
- For the example in slide 26, assume the hit latency is $1ns$, miss penalty is $100ns$. The AMAT for these example is then,

$$\begin{aligned} AMAT &= Latency_{hit} \times Rate_{hit} + Latency_{miss} \times Rate_{miss} \\ &= Latency_{hit} \times Rate_{hit} + (Latency_{hit} + Miss\_Penalty) \times Rate_{miss} \\ &= 1ns \times 22.2\% + (1ns + 100ns) * 77.8\% \\ &= 78.8ns \end{aligned} \tag{1}$$

Memory Time and Execution Time

- In general, an application's execution time can be partitioned into memory access time and computation time.
- Cache misses typically cause pipeline stalls, which constitute the majority of memory access time.
- Therefore we can roughly decompose execution time into

$$\begin{aligned} Time_{exec} &= Time_{comp} + Time_{memory} \\ &= Time_{comp} + Time_{mem\_stalls} \\ &= Time_{comp} + Latency_{miss} \times Rate_{miss} \end{aligned} \tag{2}$$

Memory Time and Execution Time cont'd

- For better accuracy, it is also better to consider read and write accesses separately in the above equations.
- Strictly speaking, the execution time here is just CPU time. Actual execution time also includes I/O wait time and OS scheduling overheads.

Hit Latency, Miss Latency and Miss Rate

- Miss latency is mostly determined by
  - The speed of DRAM;
  - The latency of the data path from CPU to DRAM.
- Hit latency is mostly determined by
  - The latency of the data path from CPU to cache.
  - The size of the cache. Larger caches are slower to probe.
  - The time it takes to compare tags.
- Miss rate is the major cache optimization metric. Most cache optimizations aim at reducing miss rates.
  - Since the miss latency is much higher than the hit latency, cache misses dominate memory access time. Therefore, reducing miss rates can significantly help memory performance.
  - Some cache optimizations, however, may reduce miss rates but increase cache latency (e.g., use bigger caches).

Acknowledgment

This lecture is based on the slides from Dr. David Brooks.