

CS3350B Computer Architecture

Memory Hierarchy: Why?

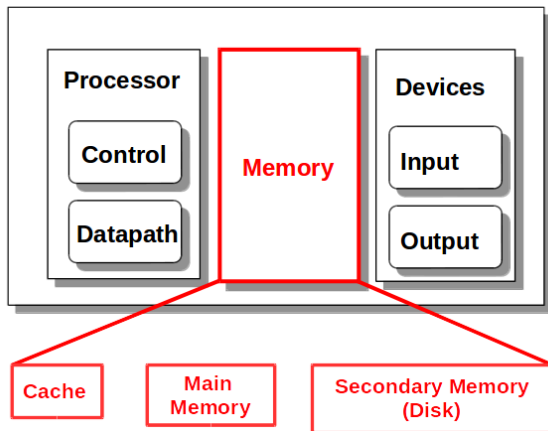
Marc Moreno Maza

http://www.csd.uwo.ca/~moreno/cs3350_moreno/index.html

Department of Computer Science
University of Western Ontario, Canada

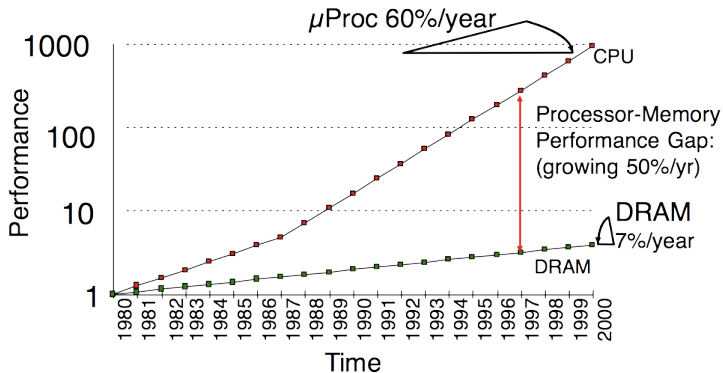
Thursday January 12, 2017

Components of a Computer



The "Memory Wall"

- Processor vs DRAM speed disparity continues to grow



Principles of Locality

- A program is likely to access a relatively small portion of the address space at any instant of time
 - **Temporal Locality** (locality in time): If a memory location is referenced then it is likely to be referenced again soon
 - **Spatial Locality** (locality in space): If a memory location is referenced, then the locations with nearby addresses are likely to be referenced soon.
- What program structures lead to temporal and spatial locality in code? In data?

Locality Example:

- Reference to array elements in succession

(**stride-1** reference pattern):

Spatial locality

- Reference to sum each iteration:

Temporal locality

```
sum = 0;
for (i=0; i<n; i++)
    sum += a[i];
return sum;
```

Locality Exercise 1

- **Question:** Does this function in C have good locality? If yes, which type?

```
int sumarrayrows(int a[M][N]) {
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Locality Exercise 2

- **Question:** Does this function in C have good locality? If yes, which type?

```
int sumarraycols(int a[M][N]) {
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Locality Exercise 3

- **Question:** Can you permute the loops so that the function scans the 3D array `a[]` with a stride-1 reference pattern (and thus has good spatial locality)?

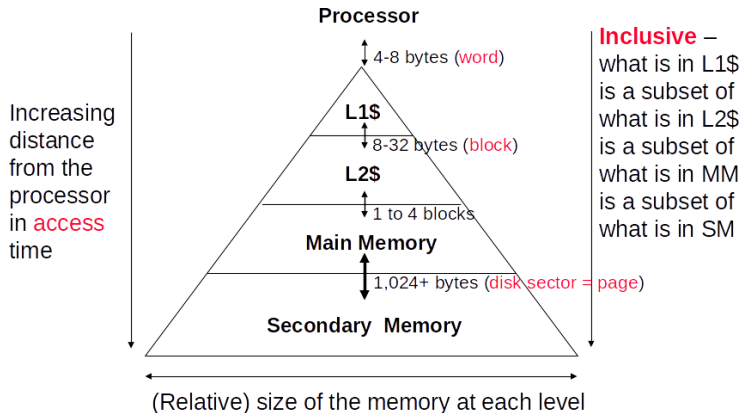
```
int sumarray3d(int a[M][N][N]) {
    int i, j, k, sum = 0;

    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < M; k++)
                sum += a[k][i][j];
    return sum;
}
```

Why Memory Hierarchies?

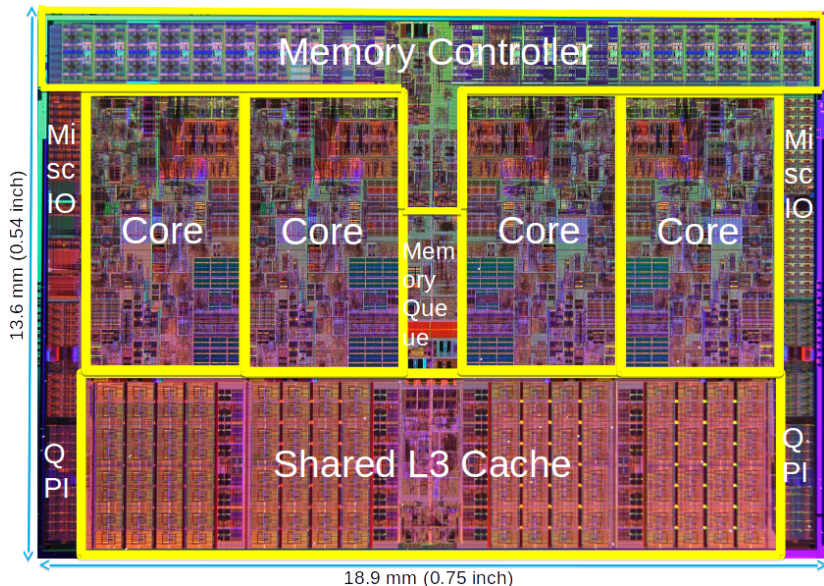
- Some fundamental and enduring properties of hardware and software:
 - Fast storage technologies (SRAM) cost more per byte and have less capacity
 - Gap between CPU and main memory (DRAM) speed is widening
 - Well-written programs tend to exhibit good locality
- These fundamental properties complement each other beautifully.
- They suggest an approach for organizing memory and storage systems known as a **memory hierarchy**, to obtain the effect of a **large, cheap, fast memory**.

Characteristics of the Memory Hierarchy



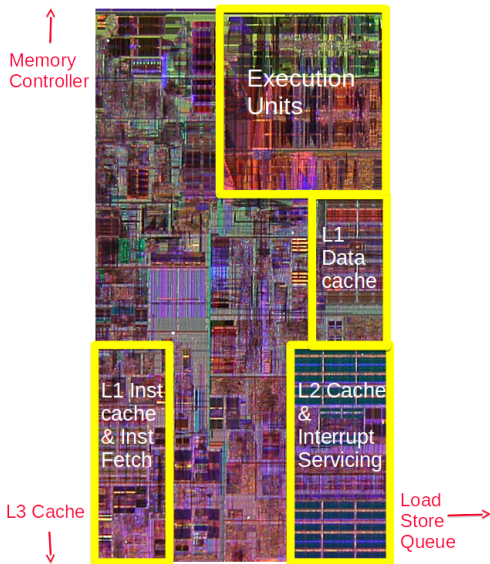
- CPU looks first for data in L1, then in L2, ..., then in main memory.

Photo of a CPU: Nehalem Die



Core Area Breakdown

32KB I\$ per core
32KB D\$ per core
512KB L2\$ per core
Share one 8-MB L3\$

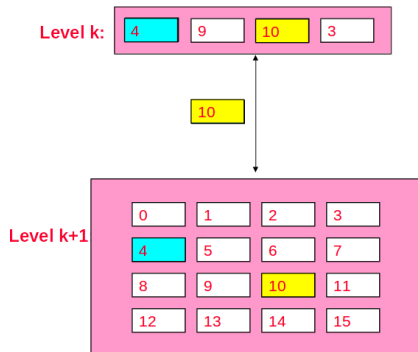


Cache Parameters of Two Processors

	Intel Nehalem	AMD Barcelona
L1 cache size & organization	32KB for each per core; 64B blocks; Split I\$ and D\$	64KB for each per core; 64B blocks; Split I\$ and D\$
L1 associativity	4-way (I), 8-way (D) set assoc.; ~LRU replacement	2-way set assoc.; LRU replacement
L1 write policy	write-back, write-allocate	write-back, write-allocate
L2 cache size & organization	256MB (0.25MB) per core; 64B blocks; Unified	512KB (0.5MB) per core; 64B blocks; Unified
L2 associativity	8-way set assoc.; ~LRU	16-way set assoc.; ~LRU
L2 write policy	write-back, write-allocate	write-back, write-allocate
L3 cache size & organization	8192KB (8MB) shared by cores; 64B blocks; Unified	2048KB (2MB) shared by by cores; 64B blocks; Unified
L3 associativity	16-way set assoc.	32-way set assoc.; evict block shared by fewest cores
L3 write policy	write-back, write-allocate	write-back, write-allocate

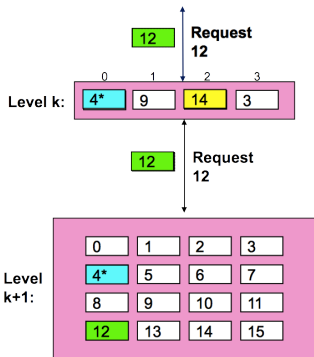
- **Cache:** A small-and-fast storage device that acts as a staging area for subset of data in a larger-and-slower device
- Fundamental idea of a memory hierarchy:
 - For each k , the fast-and-small device at level k serves as cache for the larger-and-slower device at level $k + 1$.
- **Why do memory hierarchies work?**
 - Programs tend to access (thus find) data at level k more often than they access data at level $k + 1$
 - Thus, storage at level $k + 1$ can be slower, and thus larger and cheaper per bit.
 - **Net effect:** Large pool of memory that costs as little as the cheap storage near the bottom, but that serves data to programs at \approx rate of the fast storage near the top.

Caching in a Memory Hierarchy



- Each level of memory is partitioned into blocks of consecutive bytes and of equal size (which depends on the level)
- The smaller, faster, more expensive storage-device at level k caches a subset of the blocks from level $k + 1$
- Data is copied between levels in block-sized transfer units.

General Caching Concepts



- A Program needs an object d , which is stored in some block b
- **Cache hit (at level k)**
 - The Program finds b in the cache at level k . e.g., block 14
- **Cache miss (at level k)**
 - b is not at level k , so the level k cache must fetch it from level $k + 1$. e.g., block 12
 - If the level k cache is full, then some current block must be replaced (evicted). Which one is the “victim”?
 - **Placement (mapping) policy**: where can the new block go? e.g., $b \bmod 4$
 - **Replacement policy**: which block should be evicted? e.g., LRU (least recently used).

General Caching Concepts

- Types of cache misses:
 - **Cold (compulsory) miss at level k**
 - A cold miss occurs at level k for a block b when this block is missing for the first time in the level k cache
 - **Conflict miss at level k**
 - Most caches limit block positions at level k to a small subset (sometimes a singleton) of the block positions at level $k + 1$
 - e.g. block i at level $k + 1$ must be placed in block $i \bmod 4$ at level k
 - Conflict misses occur at level k when multiple data items from level $k + 1$ all map to the same block at level k
 - e.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time, with a $i \mapsto i \bmod 4$ mapping
 - **Capacity miss at level k**
 - Occurs when the set of active blocks (that is, the data set with which the Program is working) is larger than the cache at level k

More Caching Concepts

- **Hit Rate:** the percentage of memory accesses found in a level of the memory hierarchy
- **Hit Time:** Time to access that level which consists of:
 - Time to determine hit/miss + Time to access the block.
- **Miss Rate:** the percentage of memory accesses not found in a level of the memory hierarchy, that is, $1 - (\text{Hit Rate})$.
- **Miss Penalty:** Time to replace a block in that level with the corresponding block from a lower level which consists of:
 - Time to determine hit/miss
 - Time to access the block in the lower level
 - + Time to transmit that block to the level that experienced the miss
 - + Time to insert the block in that level
 - + Time to pass the block to the requester

Hit Time \ll Miss Penalty

Examples of Cache Memories in the Hierarchy

Cache Type	What Cached	Where Cached	Latency (cycles)	Managed By
Registers	4-byte word	CPU registers	0.5	Compiler
TLB	Address translations	On-Chip TLB	0.5	Hardware
L1 cache	32-byte block	On-Chip L1	0.5	Hardware
L2 cache	32-byte block	On/Off-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main memory	100	Hardware+OS
Buffer cache	Parts of files	Main memory	100	OS
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

- The TLB (Translation lookaside buffer) stores the recent translations of virtual memory to physical memory and can be called an address-translation cache
- The Andrew File System (AFS) and Network File System (NFS) are distributed file system protocols
- A proxy server is a server (a computer system or an application) that acts as an intermediary for requests from clients seeking resources from other servers.

- Being able to look at code and get a qualitative sense of its **locality** properties is a key skill for professional programmer.
- **Examples of projects driven by data locality** (and other features):
 - BLAS (Basic Linear Algebra Subprograms)
<http://www.netlib.org/blas/>
 - SPIRAL, Software/Hardware Generation for DSP Algorithms
<http://www.spiral.net/>
 - FFTW, by Matteo Frigo and Steven G. Johnson
<http://www.fftw.org/>
 - Cache-Oblivious Algorithms, by Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran, 1999
https://en.wikipedia.org/wiki/Cache-oblivious_algorithm
 - ...

Memory Performance

- **Cache Miss Rate:** number of cache misses/total number of cache references (accesses)
Miss rate + hit rate = 1.0 (100%)
- **Miss Penalty:** the difference in access time of a given memory level, where a block is missing, and the lower level, where the block is found.
- **Average Memory Access Time (AMAT)** is the average time to access memory considering both hits and misses
$$\text{AMAT} = \text{Time for a Hit} + \text{Miss Rate} * \text{Miss Penalty}$$
- What is the AMAT for a processor with a 200 ps clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache access time of 1 clock cycle?

Memory Performance

- **Cache Miss Rate:** number of cache misses/total number of cache references (accesses)
Miss rate + hit rate = 1.0 (100%)
- **Miss Penalty:** the difference in access time of a given memory level, where a block is missing, and the lower level, where the block is found.
- **Average Memory Access Time (AMAT)** is the average time to access memory considering both hits and misses
 $AMAT = \text{Time for a Hit} + \text{Miss Rate} * \text{Miss Penalty}$
- What is the AMAT for a processor with a 200 ps clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache access time of 1 clock cycle?
 $1 + 0.02 * 50 = 2 \text{ clock cycles, or } 2 * 200 = 400 \text{ ps}$

Measuring Cache Performance - Effect on CPI

- Assuming that the cache hit costs are included as part of the *normal CPU execution cycle*, we have:

$$\begin{aligned}\text{CPU time} &= \text{IC} \times \text{CPI} \times \text{CC} \\ &= \text{IC} \times \underbrace{(\text{CPI}_{\text{ideal}} + \text{Average memory stall cycles})}_{\text{CPI}_{\text{stall}}} \times \text{CC}\end{aligned}$$

- A simple definition for *memory-stall cycles*:

Memory-stall cycles = #accesses/program × miss rate × miss penalty

This ignores extra costs of write misses.

Impacts of Cache Performance

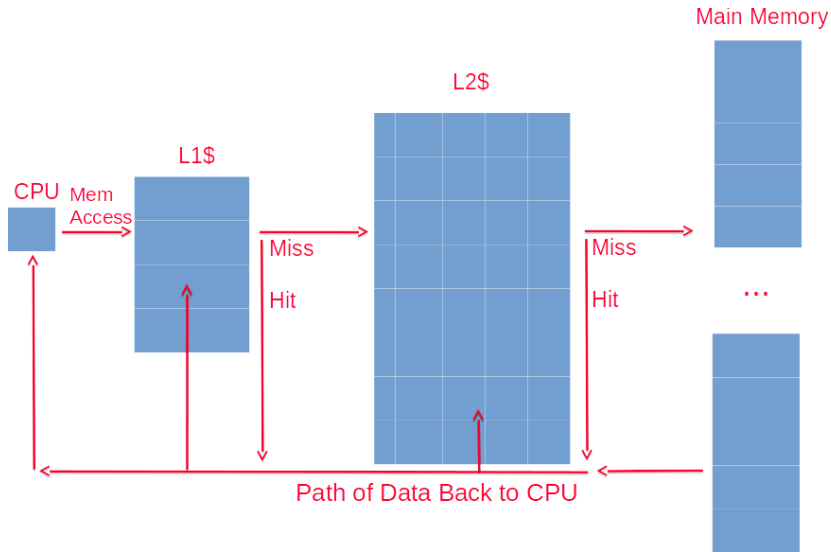
- Relative cache miss penalty increases as processor performance improves (faster clock rate and/or lower CPI). Indeed:
 - Memory speed unlikely to improve as fast as processor cycle time.
 - When calculating CPI_{stall} , the cache miss penalty is measured in processor clock cycles needed to handle a miss.
 - The lower is CPI_{ideal} , the more pronounced is the impact of stalls
- Processor with a CPI_{ideal} of 2, a 100 cycle miss penalty, 36% load/store instr's, and 2% instruction-cache miss rate, and 4% data-cache miss rate.
 - Memory-stall cycles = $2\% \times 100 + 36\% \times 4\% \times 100 = 3.44$
 - So $CPI_{stall} = 2 + 3.44 = 5.44$
 - More than twice the CPI_{ideal} !
- What if the CPI_{ideal} is reduced to 1?
- What if the data cache miss rate went up by 1%?

Impacts of Cache Performance

- Relative cache miss penalty increases as processor performance improves (faster clock rate and/or lower CPI). Indeed:
 - Memory speed unlikely to improve as fast as processor cycle time.
 - When calculating CPI_{stall} , the cache miss penalty is measured in processor clock cycles needed to handle a miss.
 - The lower is CPI_{ideal} , the more pronounced is the impact of stalls
- Processor with a CPI_{ideal} of 2, a 100 cycle miss penalty, 36% load/store instr's, and 2% instruction-cache miss rate, and 4% data-cache miss rate.
 - Memory-stall cycles = $2\% \times 100 + 36\% \times 4\% \times 100 = 3.44$
 - So $CPI_{stall} = 2 + 3.44 = 5.44$
 - More than twice the CPI_{ideal} !
- What if the CPI_{ideal} is reduced to 1?
- What if the data cache miss rate went up by 1%?

$$\text{Memory-stall cycles} = 2\% \times 100 + 36\% \times 4\% \times 100 = 3.800$$

Multiple Cache Levels



Multiple Cache Levels

- With advancing technology, the CPU has more room on die for bigger L1 caches and for second level cache - normally a **unified** L2 cache (i.e., it holds both instructions and data,) and in some cases even a unified L3 cache.
- **New AMAT Calculation:**

$$\begin{aligned} \mathbf{AMAT} &= \text{L1 Hit Time} + \text{L1 Miss Rate} * \mathbf{L1 Miss Penalty}, \\ \mathbf{L1 Miss Penalty} &= \text{L2 Hit Time} + \text{L2 Miss Rate} * \text{L2 Miss Penalty} \end{aligned}$$

and so forth (final miss penalty is Main Memory access time)

New AMAT Example

- 1 cycle L1 hit time, 2% L1 miss rate, 5 cycle L2 hit time, 5% L2 miss rate.
- 100 cycle main memory access time
- Without L2 cache:

- With L2 cache:

New AMAT Example

- 1 cycle L1 hit time, 2% L1 miss rate, 5 cycle L2 hit time, 5% L2 miss rate.
- 100 cycle main memory access time
- Without L2 cache:
 $AMAT = 1 + .02*100 = 3$
- With L2 cache:
 $AMAT = 1 + .02*(5 + .05*100) = 1.2$

- Wanted: effect of a **large, cheap, fast memory**
- Approach: **Memory Hierarchy**
 - Successively lower levels contain “most used” data from next higher level
 - Exploits **temporal & spatial** locality of programs
 - Do the common case fast, worry less about the exceptions (RISC design principle)
- **Challenges to programmer:**
 - Develop cache friendly (efficient) programs
- From Wikipedia: Reduced instruction set computing, or RISC, is a CPU design strategy based on the insight that a simplified instruction set provides higher performance when combined with a microprocessor architecture capable of executing those instructions using fewer microprocessor cycles per instruction.[1] A computer based on this strategy is a reduced instruction set computer, also called RISC.

Layout of C Arrays in Memory (hints for the exercises)

- C arrays allocated in row-major order
 - Each row in contiguous memory locations
- Stepping through columns in one row:
 - `for (i = 0; i < N; i++)`
 `sum += a[0][i];`
 - Accesses successive elements of size k bytes
 - If block size (B) $>$ k bytes, exploit spatial locality
 Compulsory miss rate = $k \text{ bytes} / B$.
 - Typically $k = 8$ and $B = 8k$ or $B = 16k$.
- Stepping through rows in one column:
 - `for (i = 0; i < n; i++)`
 `sum += a[i][0];`
 - Accesses distant elements
 - No spatial locality!
 Compulsory miss rate = 1 (i.e. 100%)