Components of a Computer

Processor
  Control
  Datapath

Memory

Devices
  Input
  Output

Cache
Main Memory
Secondary Memory (Disk)
The “Memory Wall”

- Processor vs DRAM speed disparity continues to grow

![Graph showing the growth of processor and DRAM performance gap over time.](image-url)
The Principle of Locality

- Program 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 will tend to be referenced again soon
  - **Spatial Locality** (locality in space): If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon
- What program structures lead to temporal and spatial locality in code? In data?

Locality Example:

- **Data**
  - Reference array elements in succession (**stride-1** reference pattern):
    - Spatial locality
  - Reference sum each iteration:
    - Temporal locality

- **Instructions**
  - Reference instructions in sequence: **Spatial locality**
  - Cycle through loop repeatedly: **Temporal locality**

```
sum = 0;
for (i=0; i<n; i++)
    sum += a[i];
return sum;
```
Question: Does this function in C have good locality?

```c
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;
}
```
Question: Does this function in C have good locality?

```c
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;
}
```
- **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)?

```c
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.
CPU looks first for data in L1, then in L2, ..., then in main memory.
Core Area Breakdown

- 32KB I$ per core
- 32KB D$ per core
- 512KB L2$ per core
- Share one 8-MB L3$
## Two Machines’ Cache Parameters

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

- **Cache**: Smaller, faster storage device that acts as staging area for subset of data in a larger, slower device
- Fundamental idea of a memory hierarchy:
  - For each $k$, the faster, smaller device at level $k$ serves as cache for larger, slower device at level $k+1$
- **Why do memory hierarchies work?**
  - Programs tend to access 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.

Ning Xie (http://www.csd.uwo.ca/courses/CS3350b) Department of Computer Science University of Western Ontario, Canada

Caching in a Memory Hierarchy

- Smaller, faster, more expensive device at level k caches a subset of the blocks from level k+1.
- Data is copied between levels in block-sized transfer units.
- Larger, slower, cheaper storage device at level k+1 is partitioned into blocks.
- Program needs object d, which is stored in some block b
- **Cache hit**
  - Program finds b in the cache at level k. e.g., block 14
- **Cache miss**
  - b is not at level k, so level k cache must fetch it from level k+1. e.g., block 12
  - If 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 mod 4
    - **Replacement policy**: which block should be evicted? e.g., LRU into blocks.
Types of cache misses:

- **Cold (compulsory) miss**
  - Cold misses occur because the cache is empty

- **Conflict miss**
  - Most caches limit blocks 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 mod 4) at level k
  - Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block
  - e.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time

- **Capacity miss**
  - Occurs when the set of active cache blocks (working set) is larger than the cache
More Caching Concepts

- **Hit Rate**: the fraction of memory accesses found in a level of the memory hierarchy
  - **Hit Time**: Time to access that level which consists of
    - Time to access the block + Time to determine hit/miss

- **Miss Rate**: the fraction of memory accesses not found in a level of the memory hierarchy ⇒ 1 - (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 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 requestor

Hit Time « Miss Penalty
### Examples of Caching in the Hierarchy

<table>
<thead>
<tr>
<th>Cache Type</th>
<th>What Cached</th>
<th>Where Cached</th>
<th>Latency (cycles)</th>
<th>Managed By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>4-byte word</td>
<td>CPU registers</td>
<td>0.5</td>
<td>Compiler</td>
</tr>
<tr>
<td>TLB</td>
<td>Address translations</td>
<td>On-Chip TLB</td>
<td>0.5</td>
<td>Hardware</td>
</tr>
<tr>
<td>L1 cache</td>
<td>32-byte block</td>
<td>On-Chip L1</td>
<td>0.5</td>
<td>Hardware</td>
</tr>
<tr>
<td>L2 cache</td>
<td>32-byte block</td>
<td>On/Off-Chip L2</td>
<td>10</td>
<td>Hardware</td>
</tr>
<tr>
<td>Virtual Memory</td>
<td>4-KB page</td>
<td>Main memory</td>
<td>100</td>
<td>Hardware+ OS</td>
</tr>
<tr>
<td>Buffer cache</td>
<td>Parts of files</td>
<td>Main memory</td>
<td>100</td>
<td>OS</td>
</tr>
<tr>
<td>Network buffer cache</td>
<td>Parts of files</td>
<td>Local disk</td>
<td>10,000,000</td>
<td>AFS/NFS client</td>
</tr>
<tr>
<td>Browser cache</td>
<td>Web pages</td>
<td>Local disk</td>
<td>10,000,000</td>
<td>Web browser</td>
</tr>
<tr>
<td>Web cache</td>
<td>Web pages</td>
<td>Remote server disks</td>
<td>1,000,000,000</td>
<td>Web proxy server</td>
</tr>
</tbody>
</table>
- Being able to look at code and get qualitative sense of its **locality** is key skill for professional programmer

- **Examples:**
  - BLAS (Basic Linear Algebra Subprograms)
  - SPIRAL, Software/Hardware Generation for DSP Algorithms
  - FFTW, by Matteo Frigo and Steven G. Johnson
  - ...
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 between lower level access time and cache access time

- **Average Memory Access Time (AMAT)** is the average time to access memory considering both hits and misses
  
  AMAT = Time for a Hit + Miss Rate * 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?
  
  $1 + 0.02 \times 50 = 2$ clock cycles, or $2 \times 200 = 400$ ps
Assuming cache hit costs are included as part of the normal CPU execution cycle, then

\[
\text{CPU time} = \text{IC} \times \text{CPI} \times \text{CC} \\
= \text{IC} \times (\text{CPI}_{\text{ideal}} + \text{Average memory stall cycles}) \times \text{CC}
\]

A simple model for Memory-stall cycles:

Memory-stall cycles = \#accesses/program \times \text{miss rate} \times \text{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)
  - Memory speed unlikely to improve as fast as processor cycle time. When calculating $CPI_{\text{stall}}$, cache miss penalty is measured in processor clock cycles needed to handle a miss
  - Lower the $CPI_{\text{ideal}}$, more pronounced impact of stalls
- Processor with a $CPI_{\text{ideal}}$ of 2, a 100 cycle miss penalty, 36% load/store instr’s, and 2% instruction cache and 4% data cache miss rates
  - Memory-stall cycles = $2\% \times 100 + 36\% \times 4\% \times 100 = 3.44$
  - So $CPI_{\text{stall}} = 2 + 3.44 = 5.44$
  - More than twice the $CPI_{\text{ideal}}$
- What if the $CPI_{\text{ideal}}$ is reduced to 1?
- What if the data cache miss rate went up by 1%?
Multiple Cache Levels

- CPU
- Mem Access
- L1$
- Miss
- Hit
- Path of Data Back to CPU
- L2$
- Miss
- Hit
- Main Memory

...
With advancing technology, have 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:**

\[
\text{AMAT} = \text{L1 Hit Time} + \text{L1 Miss Rate} \times \text{L1 Miss Penalty}, \\
\text{L1 Miss Penalty} = \text{L2 Hit Time} + \text{L2 Miss Rate} \times \text{L2 Miss Penalty}
\]

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:
  AMAT = 1 + 0.02*100 = 3
- With L2 cache:
  AMAT = 1 + 0.02*(5 + 0.05*100) = 1.2
Summary

- **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

Ning Xie ([http://www.csd.uwo.ca/courses/CS3350b](http://www.csd.uwo.ca/courses/CS3350b) Department of Computer Science University of Western Ontario, Canada)
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 bytes / B

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%)