Storage Hierarchy

Instructor: Sanjeev Setia

Problem: Processor-Memory Bottleneck

Processor performance doubled about every 18 months

Bus bandwidth evolved much slower

Core 2 Duo:
- Can process at least 256 Bytes/cycle
- (1 SSE two operand add and mult)

Core 2 Duo:
- Bandwidth 2 Bytes/cycle
- Latency 100 cycles

Solution: Caches
Cache

**Definition:** Computer memory with short access time used for the storage of frequently or recently used instructions or data

---

General Cache Mechanics

Smaller, faster, more expensive memory caches a subset of the blocks

Data is copied in block-sized transfer units

Larger, slower, cheaper memory viewed as partitioned into “blocks”
**General Cache Concepts: Hit**

- **Request:** 14
- **Data in block b is needed**
- **Block b is in cache:** Hit!

**Cache**

| 8 | 9 | 14 | 3 |

**Memory**

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

**General Cache Concepts: Miss**

- **Request:** 12
- **Data in block b is needed**
- **Block b is not in cache:** Miss!
- **Block b is fetched from memory**

**Cache**

| 8 | 12 | 14 | 3 |

**Request:** 12

**Memory**

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

**Placement policy:** determines where b goes
**Replacement policy:** determines which block gets evicted (victim)
Cache Performance Metrics

■ Miss Rate
  ▪ Fraction of memory references not found in cache (misses / accesses)
    = 1 – hit rate
  ▪ Typical numbers (in percentages):
    ▪ 3-10% for L1
    ▪ can be quite small (e.g., < 1%) for L2, depending on size, etc.

■ Hit Time
  ▪ Time to deliver a line in the cache to the processor
    ▪ includes time to determine whether the line is in the cache
  ▪ Typical numbers:
    ▪ 1-2 clock cycle for L1
    ▪ 5-20 clock cycles for L2

■ Miss Penalty
  ▪ Additional time required because of a miss
    ▪ typically 50-200 cycles for main memory (Trend: increasing!)

Let's think about those numbers

■ Huge difference between a hit and a miss
  ▪ Could be 100x, if just L1 and main memory

■ Would you believe 99% hits is twice as good as 97%?
  ▪ Consider:
    cache hit time of 1 cycle
    miss penalty of 100 cycles

  ▪ Average access time:
    97% hits: 1 cycle + 0.03 * 100 cycles = 4 cycles
    99% hits: 1 cycle + 0.01 * 100 cycles = 2 cycles

■ This is why “miss rate” is used instead of “hit rate”
Types of Cache Misses

- **Cold (compulsory) miss**
  - Occurs on first access to a block

- **Conflict miss**
  - Most hardware caches limit blocks to a small subset (sometimes a singleton) of the available cache slots
    - e.g., block i must be placed in slot (i mod 4)
  - Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot
    - e.g., referencing blocks 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

Why Caches Work

- **Locality**: Programs tend to use data and instructions with addresses near or equal to those they have used recently

  - **Temporal locality**:
    - Recently referenced items are likely to be referenced again in the near future

  - **Spatial locality**:
    - Items with nearby addresses tend to be referenced close together in time
Example: Locality?

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

- **Data:**
  - Temporal: `sum` referenced in each iteration
  - Spatial: array `a[]` accessed in stride-1 pattern

- **Instructions:**
  - Temporal: cycle through loop repeatedly
  - Spatial: reference instructions in sequence

- Being able to assess the locality of code is a crucial skill for a programmer

Locality Example #1

```c
int sum_array_rows(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;
}
```
Localy Example #2

```c
int sum_array_cols(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;
}
```

Localy Example #3

```c
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;
    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[k][i][j];
    return sum;
}
```

- How can it be fixed?
Memory Hierarchies

- Some fundamental and enduring properties of hardware and software systems:
  - Faster storage technologies almost always cost more per byte and have lower capacity
  - The gaps between memory technology speeds are widening
    - True of registers $\leftrightarrow$ DRAM, DRAM $\leftrightarrow$ disk, etc.
    - Well-written programs tend to exhibit good locality

- These properties complement each other beautifully

- They suggest an approach for organizing memory and storage systems known as a memory hierarchy

An Example Memory Hierarchy

- CPU registers hold words retrieved from L1 cache
- L1 cache holds cache lines retrieved from L2 cache
- L2 cache holds cache lines retrieved from main memory
- Main memory holds disk blocks retrieved from local disks
- Local disks hold files retrieved from disks on remote network servers
- Remote secondary storage (tapes, distributed file systems, Web servers)
- L1: on-chip L1 cache (SRAM)
- L2: off-chip L2 cache (SRAM)
- L3: main memory (DRAM)
- L4: local secondary storage (local disks)
- L5: remote secondary storage (tapes, distributed file systems, Web servers)
Examples of Caching in the Hierarchy

<table>
<thead>
<tr>
<th>Cache Type</th>
<th>What is Cached?</th>
<th>Where is it Cached?</th>
<th>Latency (cycles)</th>
<th>Managed By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>4-byte words</td>
<td>CPU core</td>
<td>0</td>
<td>Compiler</td>
</tr>
<tr>
<td>TLB</td>
<td>Address translations</td>
<td>On-Chip TLB</td>
<td>0</td>
<td>Hardware</td>
</tr>
<tr>
<td>L1 cache</td>
<td>64-bytes block</td>
<td>On-Chip L1</td>
<td>1</td>
<td>Hardware</td>
</tr>
<tr>
<td>L2 cache</td>
<td>64-bytes block</td>
<td>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>

Memory Hierarchy: Core 2 Duo

L1/L2 cache: 64 B blocks

Throughput: 16 B/cycle 8 B/cycle 2 B/cycle 1 B/30 cycles
Latency: 3 cycles 14 cycles 100 cycles millions

Not drawn to scale