Systems I

Locality and Caching

Topics

- Locality of reference
- Cache principles
- Multi-level caches
Locality

Principle of Locality:

- Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.
- **Temporal locality**: Recently referenced items are likely to be referenced in the near future.
- **Spatial locality**: Items with nearby addresses tend to be referenced close together in time.

Locality Example:

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

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

Claim: Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.

Question: Does this function 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;
}
```
Locality Example

Question: Does this function 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 3-d array $a[\cdot]$ 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 < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[k][i][j];

    return sum;
}
```
Memory Hierarchies

Some fundamental and enduring properties of hardware and software:

- Fast storage technologies cost more per byte and have less capacity.
- The gap between CPU and main memory 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.
An Example Memory Hierarchy

L0: registers
CPU registers hold words retrieved from L1 cache.

L1: on-chip L1 cache (SRAM)
L1 cache holds cache lines retrieved from the L2 cache memory.

L2: off-chip L2 cache (SRAM)
L2 cache holds cache lines retrieved from main memory.

L3: main memory (DRAM)
Main memory holds disk blocks retrieved from local disks.

L4: local secondary storage (local disks)
Local disks hold files retrieved from disks on remote network servers.

L5: remote secondary storage (distributed file systems, Web servers)

Smaller, faster, and costlier (per byte) storage devices

Larger, slower, and cheaper (per byte) storage devices
Caches

Cache: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

Fundamental idea of a memory hierarchy:

- For each $k$, the faster, smaller device at level $k$ serves as a cache for the larger, slower device at level $k+1$.

Why do memory hierarchies work?

- Programs tend to access the data at level $k$ more often than they access the data at level $k+1$.
- Thus, the storage at level $k+1$ can be slower, and thus larger and cheaper per bit.
- Net effect: A large pool of memory that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.
- Use combination of small fast memory and big slow memory to give illusion of big fast memory.
Caching in a Memory Hierarchy

Level k:

Data is copied between levels in block-sized transfer units

Level k+1:

Smaller, faster, more expensive device at level k caches a subset of the blocks from level k+1

Larger, slower, cheaper storage device at level k+1 is partitioned into blocks.
General Caching Concepts

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 policy: where can the new block go? E.g., \( b \) mod 4
  - Replacement policy: which block should be evicted? E.g., LRU
General Caching Concepts

Types of cache misses:

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

- **Conflict miss**
  - Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
  - E.g. Block i at level k+1 must be placed in block (i mod 4) at level k+1.
  - 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.
# 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</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>32-byte block</td>
<td>On-Chip L1</td>
<td>1</td>
<td>Hardware</td>
</tr>
<tr>
<td>L2 cache</td>
<td>32-byte 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>
Cache Memories

Cache memories are small, fast SRAM-based memories managed automatically in hardware.

- Hold frequently accessed blocks of main memory

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

Typical bus structure:
Inserting an L1 Cache Between the CPU and Main Memory

The transfer unit between the CPU register file and the cache is a 4-byte block.

The transfer unit between the cache and main memory is a 4-word block (16 bytes).

The big slow main memory has room for many 4-word blocks.

The small fast L1 cache has room for two 4-word blocks.

The tiny, very fast CPU register file has room for four 4-byte words.
Multi-Level Caches

Options: separate **data and instruction caches**, or a **unified cache**

<table>
<thead>
<tr>
<th></th>
<th>L1 d-cache</th>
<th>L1 i-cache</th>
<th>Unified L2 Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>size:</strong></td>
<td>200 B</td>
<td>8-64 KB</td>
<td>1-4MB SRAM</td>
<td>128 MB DRAM</td>
</tr>
<tr>
<td><strong>speed:</strong></td>
<td>3 ns</td>
<td>3 ns</td>
<td>6 ns</td>
<td>60 ns</td>
</tr>
<tr>
<td><strong>$/Mbyte:</strong></td>
<td>$100/MB</td>
<td>$1.50/MB</td>
<td>$1.50/MB</td>
<td>$0.05/MB</td>
</tr>
<tr>
<td><strong>line size:</strong></td>
<td>8 B</td>
<td>32 B</td>
<td>32 B</td>
<td>8 KB</td>
</tr>
</tbody>
</table>

larger, slower, cheaper

Processor

disk
Intel Pentium Cache Hierarchy

- **L1 Data**
  - 1 cycle latency
  - 16 KB
  - 4-way assoc
  - Write-through
  - 32B lines

- **L1 Instruction**
  - 16 KB
  - 4-way
  - 32B lines

- **L2 Unified**
  - 128KB--2 MB
  - 4-way assoc
  - Write-back
  - Write allocate
  - 32B lines

- **Main Memory**
  - Up to 4GB
Find the Caches...
Summary

Today
- Locality: Spatial and Temporal
- Cache principles
- Multi-level cache hierarchies

Next Time
- Cache organization
- Replacement and writes
- Programming considerations