Cache Basics
Adapted from a presentation by Beth Richardson
bethr@ncsa.uiuc.edu

Cache Historical Note
Cache hardware first appeared on production computers in the late 1960’s.
Before that processor/memory communication looked like:

Processors designed without cache were simpler because every memory access took the same amount of time.

Main Memory Improvements (1)
A hardware improvement named interleaving reduces main memory access time.
Interleaving Defined
Main memory is divided into partitions or segments named memory banks.
Consecutive data elements are spread across the banks.
Each bank supplies one data element per bank cycle.
Multiple data elements are read in parallel, one from each bank.

Main Memory Improvements (2)
Interleaving Problem
The memory interleaving improvement assumes that memory is accessed sequentially.
If we have 2-way memory interleaving, but the code accesses every other location, there is no benefit.
Regardless of the above problem the bank cycle time is 4-8 times the CPU clock cycle time. The main memory can’t keep up with the fast CPU and keep it busy with data.
Large main memory with a cycle time comparable to the processor is not affordable.

Purpose of Cache
The purpose of cache is to improve the memory access time to the processor.
There is an overhead associated with cache, but the benefits outweigh the costs.

CPU
logic
Cache
Main Memory
Today’s Computers

Today almost every computer, large or small, has a cache. The CPU must be able to handle variable memory access times.

Registers

Purpose of Registers
- Registers are the sources and destinations of CPU operations.

Description of Registers
- They hold one data element and are 32 bits or 64 bits wide.
- They are on-chip and built from SRAM.

Speed of Registers
- Register access speeds are comparable to processor speeds.

Memory Hierarchy

The different memory subsystems in the memory hierarchy have different speeds, sizes, and costs.

Memory technology:
- Smaller memory is faster
- Slower memory is cheaper

The memory hierarchy is built so that the fastest memory is closest to the CPU, and the slower memories are further away from the CPU.

Memory Hierarchy (Cont.)

It’s called a hierarchy because every level is a subset of a level further away.

All data in one level is found in the level below.

Performance is the reason for having a memory hierarchy.

Analogy for Memory Hierarchy
- The library books on your desk are a subset of the books in the LUMS library, which in turn is a subset of the books in the Library of Congress.

Principle of Locality

Temporal Locality
- When an item is referenced, it will be referenced again soon.

Spatial Locality
- When an item is referenced, items whose addresses are nearby will tend to be referenced soon (library analogy).
Cache Line or Block

The overhead of the cache can be reduced by fetching a chunk or block of data elements. When a main memory access is made, a cache line (or block) of data is brought into the cache instead of a single data element. A cache line is defined in terms of a number of bytes. For example, we say that the cache line is 32 bytes, or 128 bytes. This takes advantage of spatial locality. The additional elements in the cache line will most likely be needed soon.

Cache Line Size

How large should computer designers make the cache line? The cache miss rate falls as the size of the cache line increases. But there is a point of negative returns on cache line size. When the cache line size becomes too large, the transfer time increases.

Cache Hit/Miss

A cache hit occurs when the data element requested by the processor is in the cache. You want to maximize cache hits. Cache Hit Rate

It’s the fraction of time that the requested data is found in the cache.

A cache miss occurs when the data element requested by the processor is not in the cache. You want to minimize cache misses. Cache Miss Rate

Defined as 1.0 - Hit Rate

Miss Penalty (miss time)

The time needed to retrieve the data from a lower level (downstream) of the memory hierarchy.

Two Levels of Cache

An on-chip cache performs the fastest. But the computer designer makes a trade-off between die size and cache size. Hence on-chip cache has a small size. When the on-chip cache has a cache miss, the time to access the slower main memory is very large. A cache miss is very costly. To solve this problem, computer designers have implemented a larger, slower off-chip cache. It speeds up the on-chip cache miss time.

Memory Hierarchy

```
  CPU
  |    |
  | Registers
  |    |
  |    L1 Cache
  |    |
  |    L2 Cache
  |    |
  |    Main Memory
  |    |
  |    Disk
```
Split or Unified Cache

Unified Cache
The cache is a combined instruction-data cache.

Split Cache
The cache is split into 2 parts.
One for the instructions, the instruction cache.
Another for the data, named the data cache
The 2 caches are independent of each other, and they can have independent properties.

Disadvantage of a Unified Cache
When the data access and instruction access conflict with each other, the cache may thrash.

Cache Mapping

Cache Mapping Defined
Cache mapping determines which cache location should be used to store a copy of a data element from main memory.
There are 2 mapping strategies - direct mapped cache, and set associative cache.

Direct Mapped Cache
There is a one to one correspondence between main memory addresses and cache addresses.
cache address = main memory address MOD (size of cache)
Cache lines are mapped to unique addresses.

Direct Mapped Cache Diagram

<table>
<thead>
<tr>
<th>M</th>
<th>E</th>
<th>M</th>
<th>O</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>...</td>
<td>128</td>
<td>129</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Direct Mapped Cache Diagram

<table>
<thead>
<tr>
<th>M</th>
<th>E</th>
<th>M</th>
<th>O</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

N-way Set Associative Cache
Can think of cache as being divided into N vertical strips (usually N is 2 or 4).
A cache line is assigned to just one of the strips.

Set Associative Cache

<table>
<thead>
<tr>
<th>C</th>
<th>A</th>
<th>C</th>
<th>H</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td></td>
</tr>
</tbody>
</table>

Cache Block Replacement

With Direct Mapped Cache
A cache line can only be mapped to one unique place in cache. The new cache line replaces the cache block at that address.

With Set Associative Cache
There is a choice. We’ll look at 3 strategies named Random, LRU, and FIFO.

Random
There is a uniform random replacement within the set of cache blocks.
The advantage of random replacement is that it’s simple and inexpensive to implement.

LRU (Least Recently Used)
The block that gets replaced is the one that hasn’t been used for the longest time.
The principle of temporal locality tells us that recently used data are likely to be used again soon.
An advantage of LRU is that it preserves temporal locality.
A disadvantage of LRU is that it’s expensive to keep track of cache access patterns.
In empirical studies there was little performance difference between LRU and Random.
### Cache Block Replacement (3)

**FIFO (First In First Out)**
- Replace the block that was used N accesses ago, regardless of the access pattern.
- In empirical studies Random replacement generally outperformed FIFO.

### Cache Thrashing

**Thrashing Definition**
- Cache thrashing is a problem that happens when a frequently used cache line gets displaced by another frequently used cache line.
- Cache thrashing can happen for both instruction and data caches.
- The CPU can’t find the data element it wants in the cache and must make another main memory cache line access.
- The same data elements are repeatedly fetched into and displaced from the cache.

### Cache Thrashing (2)

**Why does thrashing happen?**
- The computational code statements have too many variables and arrays for the needed data elements to fit in cache. Cache lines are discarded and later retrieved.
- The arrays are dimensioned too large to fit in cache.
- The arrays are accessed with indirect addressing, e.g. a(k(j)).

**How to Reduce Thrashing**
- The computer designer can reduce cache thrashing by increasing the cache’s set associativity.