Locality

- A principle that makes having a memory hierarchy a good idea
- If an item is referenced,
  - temporal locality: it will tend to be referenced again soon
  - spatial locality: nearby items will tend to be referenced soon.

Why does code have locality?

- Our initial focus: two levels (upper, lower)
  - block: minimum unit of data
  - hit: data requested is in the upper level
  - miss: data requested is not in the upper level

Cache

- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is one word of data
  - "direct mapped"
    - For each item of data at the lower level, there is exactly one location in the cache where it might be.
    - e.g., lots of items at the lower level share locations in the upper level

Direct Mapped Cache

- Mapping: address is modulo the number of blocks in the cache
Taking advantage of spatial locality:

Direct Mapped Cache

- Read hits: this is what we want!
- Read misses: stall the CPU, fetch block from memory, deliver to cache, restart
- Write hits: can replace data in cache and memory (write-through)
- Write misses: read the entire block into the cache, then write the word

Hardware Issues

- Make reading multiple words easier by using banks of memory
- It can get a lot more complicated...

Performance

- Increasing the block size tends to decrease miss rate:
- Use split caches because there is more spatial locality in code:

<table>
<thead>
<tr>
<th>Program</th>
<th>Block size</th>
<th>Instruction miss rate</th>
<th>Data miss rate</th>
<th>Effective combined miss rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>1</td>
<td>6.1%</td>
<td>2.1%</td>
<td>8.4%</td>
</tr>
<tr>
<td>spice</td>
<td>1</td>
<td>1.2%</td>
<td>1.3%</td>
<td>2.5%</td>
</tr>
<tr>
<td>gcc</td>
<td>4</td>
<td>2.0%</td>
<td>1.7%</td>
<td>3.7%</td>
</tr>
<tr>
<td>spice</td>
<td>4</td>
<td>0.3%</td>
<td>0.8%</td>
<td>0.4%</td>
</tr>
</tbody>
</table>
Performance

- Simplified model:
  - execution time = (execution cycles + stall cycles) × cycle time
  - stall cycles = # of instructions × miss ratio × miss penalty
- Two ways of improving performance:
  - decreasing the miss ratio
  - decreasing the miss penalty

What happens if we increase block size?

Decreasing miss ratio with associativity

Compared to direct mapped, give a series of references that:
- results in a lower miss ratio using a 2-way set associative cache
- results in a higher miss ratio using a 2-way set associative cache
assuming we use the “least recently used” replacement strategy
Decreasing miss penalty with multilevel caches

- Add a second level cache:
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in 2nd level cache

- Example:
  - CPI of 1.0 on a 5 GHz machine with a 5% miss rate, 100ns DRAM access
  - Adding 2nd level cache with 5ns access time decreases miss rate to .5%

- Using multilevel caches:
  - try and optimize the hit time on the 1st level cache
  - try and optimize the miss rate on the 2nd level cache

Cache Complexities

- Not always easy to understand implications of caches:

  Theoretical behavior of Radix sort vs. Quicksort
  Observed behavior of Radix sort vs. Quicksort

Virtual Memory

- Main memory can act as a cache for the secondary storage (disk)

  Advantages:
  - illusion of having more physical memory
  - program relocation
  - protection
Pages: virtual memory blocks

- Page faults: the data is not in memory, retrieve it from disk
  - huge miss penalty, thus pages should be fairly large (e.g., 4KB)
  - reducing page faults is important (LRU is worth the price)
  - can handle the faults in software instead of hardware
  - using write-through is too expensive so we use writeback