Problem: Processor-Memory Bottleneck

Processor performance doubled about every 18 months
Bus bandwidth doubled every 36 months

Solution: Caches/Memory hierarchy

Typical Memory Hierarchy

L0: registers
CPU registers hold words retrieved from L1 cache
L1: on-chip L1 cache (SRAM)
L2 cache holds cache lines retrieved from L2 cache
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
(tapes, distributed file systems, Web servers)

Abstracted Microarchitecture: Example Core i7 Haswell (2013) and Sandybridge (2011)

Part of these slides are adapted from the course associated with this book

The next slide is from the course “How to Write Fast Numerical Code”
http://people.inf.ethz.ch/markusp/teaching/263-2300-ETH-spring17/course.html
It contains additional information on latency and throughput of caches
Why Caches Work: Locality

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

- **History of locality**
  - **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?

- **Data**:
  - Temporal: `sum` referenced in each iteration
  - Spatial: `array` accessed consecutively

- **Instructions**:
  - Temporal: loops cycle through the same instructions
  - Spatial: instructions referenced in sequence

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

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

Cache

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

  - Naturally supports **temporal locality**
  - **Spatial locality** is supported by transferring data in blocks
    - Core family: one block = 64 B = 8 doubles

Example (S=4, E=2)

- Add associativity (E = 2, B = 32 bytes, S = 8)
- Show how elements are mapped into cache

Cache Structure

Example (S=4, E=2)

- Assume: cold (empty) cache, `a[0][0]` goes here
- `B = 32 byte = 4 doubles`
### General Cache Organization (S, E, B)

- **E** = 2<sup>e</sup> lines per set
- **E** = associativity, E=1: direct mapped
- **S** = 2<sup>s</sup> sets

### Cache Read

- **E** = 2<sup>e</sup> lines per set
- **E** = associativity, E=1: direct mapped

### Terminology

- **Direct mapped cache**:
  - Cache with **E** = 1
  - Means every block from memory has a unique location in cache

- **Fully associative cache**:
  - Cache with **S** = 1 (i.e., maximal **E**)
  - Means every block from memory can be mapped to any location in cache
  - In practice expensive to build
  - One can view the register file as a fully associative cache

- **LRU (least recently used) replacement**
  - when selecting which block should be replaced (happens only for **E** > 1), the least recently used one is chosen

### Types of Cache Misses (The 3 C’s)

- **Compulsory (cold) miss**
  - Occurs on first access to a block

- **Capacity miss**
  - Occurs when working set is larger than the cache

- **Conflict miss**
  - Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot
  - Not a clean classification but still useful

### Example: (Blackboard)

- **z** = **x** + **y**, **x**, **y**, **z** vector of length **n**
- assume they fit jointly in cache + cold cache
- memory traffic Q(n)?
- operational intensity I(n)?

### What about writes?

- **What to do on a write-hit?**
  - **Write-through**: write immediately to memory
  - **Write-back**: defer write to memory until replacement of line

- **What to do on a write-miss?**
  - **Write-allocate**: load into cache, update line in cache
  - **No-write-allocate**: writes immediately to memory
Summary

- It is important to assess temporal and spatial locality in the code
- Cache structure is determined by three parameters
  - block size
  - number of sets
  - associativity
- You should be able to roughly simulate a computation on paper