The memory hierarchy: Locality

Yipeng Huang

Rutgers University

April 5, 2022

Table of contents

Announcements

PA4 bomb lab

- Extended 24 hours. Due Wednesday, April 6 just before midnight.

PA 5 to be released very soon

Class session plan

- Today, Tuesday, 4/5: Locality (Book chapters 6.1, 6.2, and 6.3)
- Thursday, 4/7: Cache memories (Book chapter 6.4)
- Tuesday, 4/12: Cache-friendly code–cache blocking (Book chapters 6.5 and 6.6)
- Thursday, 4/14: Cache-friendly code–cache oblivious algorithms

Table of contents

Cache, memory, storage, and network hierarchy trends

- Assembly programming view of computer: CPU and memory.
- Full view of computer architecture and systems: +caches, +storage, +network

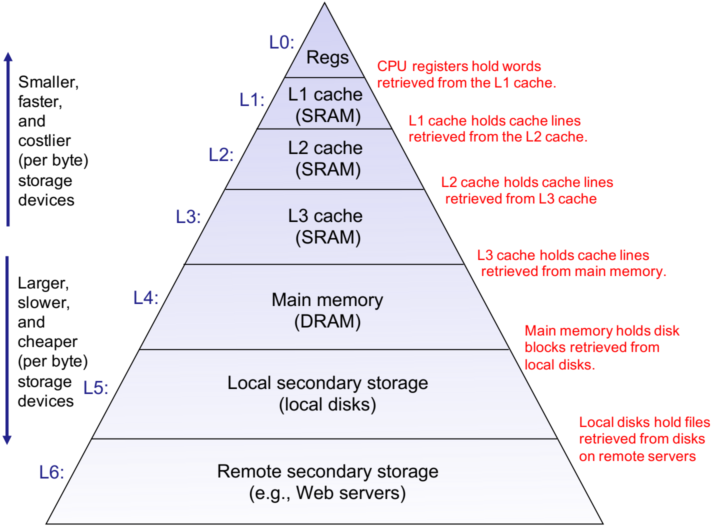


Figure: Memory hierarchy. Image credit CS:APP

Cache, memory, storage, and network hierarchy trends

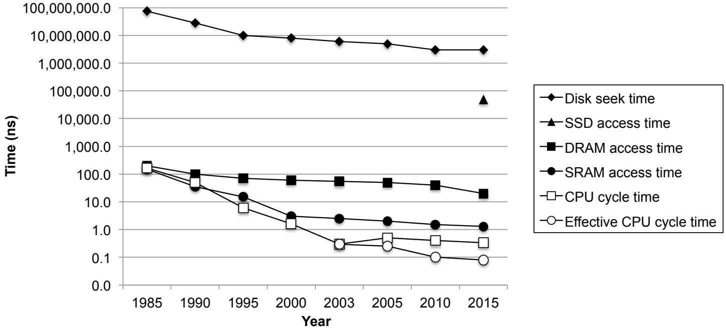


Figure: Widening gap: CPU processing time vs. memory access time. Image credit CS:APP

Topic of this chapter:

- Technology trends that drive CPU-memory gap.
- How to create illusion of fast access to capacious data.

Static random-access memory (caches)

- SRAM is bistable logic
- Access time: 1 to 10 CPU clock cycles
- Implemented in the same transistor technology as CPUs, so improvement has matched pace.

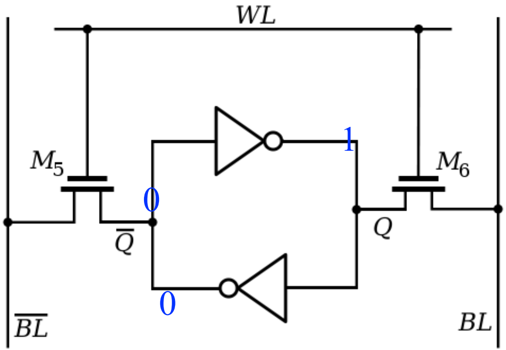


Figure: SRAM operating principle. Image credit Wikimedia

Dynamic random-access memory (main memory)

- Needs refreshing every 10s of milliseconds
- 8GB typical in laptop; 1TB on each ilab machine
- Access time: 100 CPU clock cycles
- Memory gap: DRAM technological improvement slower relative to CPU/SRAM.

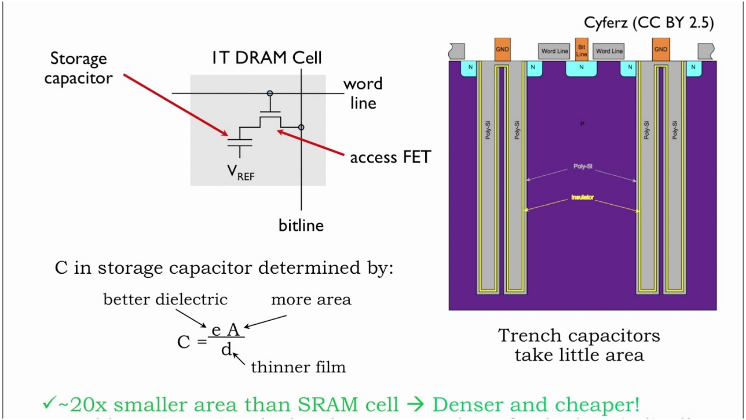


Figure: DRAM operating principle. Image credit ocw.mit.edu

CPU / DRAM main memory interface

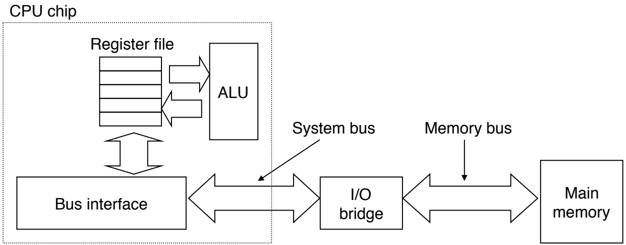


Figure: Memory Bus. Image credit CS:APP

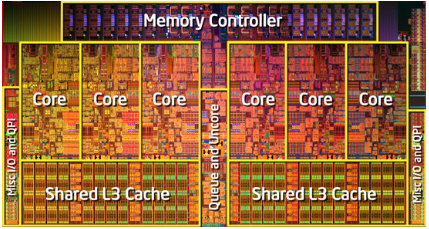


Figure: Intel 2020 Gulftown die shot. Image credit AnandTech

- DDR4 bus standard supports 25.6GB/s transfer rate

Solid state and hard disk drives (storage)

Technology

- SSD: flash nonvolatile memory stores data as charge.
- HDD: magnetic orientation.
- Access time: 100K CPU clock cycles

For in-depth on storage, file systems, and operating systems, take:

- CS214 Systems Programming
- CS416 Operating Systems Design

Since summer 2021, LCSR (admins of iLab) have moved the storage systems that supports everyone's home directories to SSD. `https://resources.cs.rutgers.edu/docs/file-storage/storage-technology-options/`

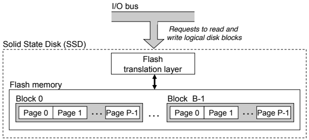


Figure: SSD. Image credit CS:APP

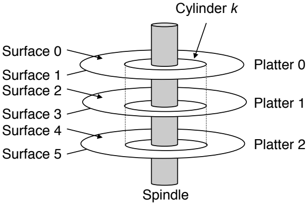


Figure: HDD. Image credit CS:APP

I/O interfaces

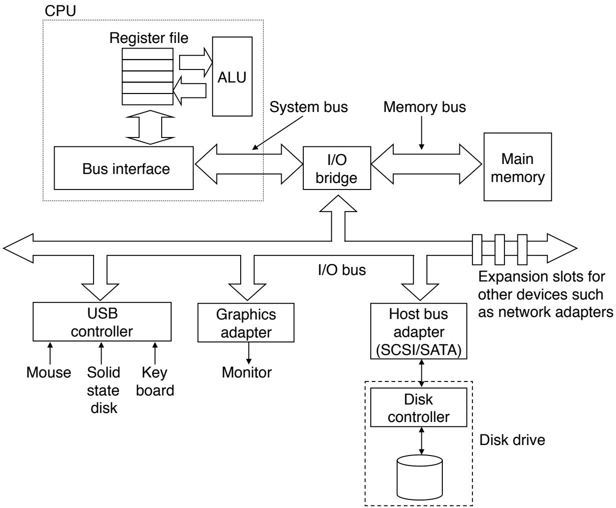


Figure: I/O Bus. Image credit CS:APP

Storage interfaces

- SATA 3.0 interface (6Gb/s transfer rate) typical
- PCIe (15.8 GB/s) becoming commonplace for SSD
- But interface data rate is rarely the bottleneck.

For in-depth on computer network layers, take:

- CS352 Internet Technology

Cache, memory, storage, and network hierarchy trends

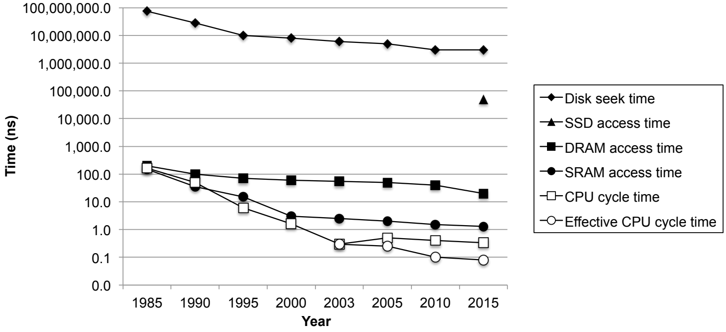


Figure: Widening gap: CPU processing time vs. memory access time. Image credit CS:APP

Topic of this chapter:

- Technology trends that drive CPU-memory gap.
- How to create illusion of fast access to capacious data.

Table of contents

Locality: How to create illusion of fast access to capacious data

From the perspective of memory hierarchy, locality is using the data in at any particular level more frequently than accessing storage at next slower level.

First, let's experience the puzzling effect of locality in `sumArray.c`

- sumArrayRows()
- sumArrayCols()

Well-written programs maximize locality

- Spatial locality
- Temporal locality

Spatial locality

```
double dotProduct (
  double a[N],
  double b[N],
) {
  double sum = 0.0;
  for(size_t i=0; i<N; i++){
    sum += a[i] * b[i];
  }
  return sum;
}
```

Spatial locality

- Programs tend to access adjacent data.
- Example: stride 1 memory access in `a` and `b`.

Temporal locality

```c
double dotProduct (
  double a[N],
  double b[N],
) {
  double sum = 0.0;
  for(size_t i=0; i<N; i++){
    sum += a[i] * b[i];
  }
  return sum;
}
```

Temporal locality

- Programs tend to access data over and over.
- Example: `sum` gets accessed N times in iteration.

Table of contents

CPU / cache / DRAM main memory interface

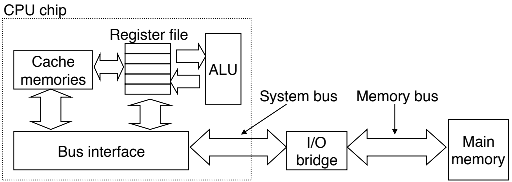
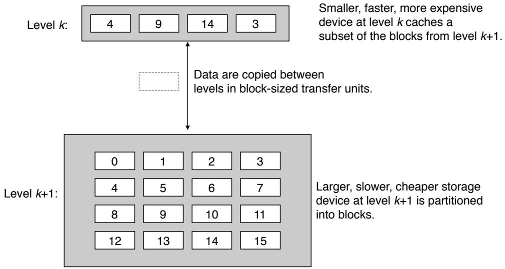


Figure: Cache resides on CPU chip close to register file. Image credit CS:APP



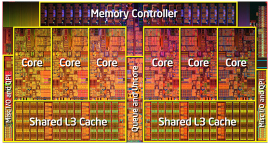


Figure: Intel 2020 Gulftown die shot. Image credit AnandTech

Figure: Cache stores a temporary copy from the slower main memory. Image credit CS:APP

CPU / cache / DRAM main memory interactions

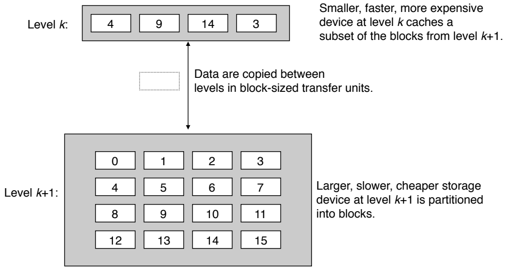


Figure: Cache stores a temporary copy from the slower main memory. Image credit CS:APP

When CPU loads (LD) from memory

- Cache read hit
- Cache read miss

When CPU stores (ST) to memory

- Cache write hit
- Cache write miss

Table of contents

Cache placement policy (how to find data at address for read and write hit)

Several designs for caches

- Fully associative cache
- Direct-mapped cache
- N-way set-associative cache

Cache design options use m-bit memory addresses differently

- t-bit tag
- s-bit set index
- b-bit block offset

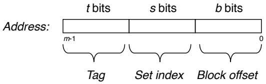


Figure: Memory addresses. Image credit CS:APP

Fully associative cache

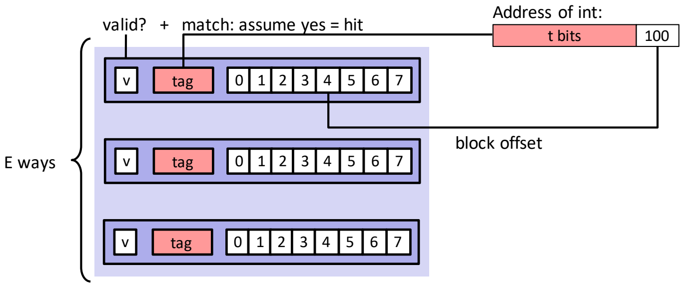


Figure: Fully associative cache. Image credit CS:APP

m-bit memory address split into:

- t-bit tag
- b-bit block offset

Fully associative cache

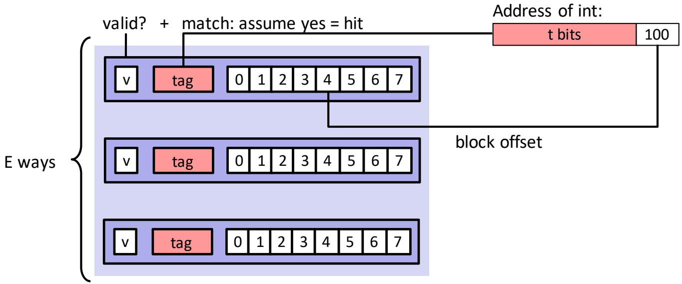


Figure: Fully associative cache. Image credit CS:APP

b-bit block offset

- here, $b = 3$
- The number of bytes in a block is $B = 2^b = 2^3 = 8$
- A block is the minimum number of bytes that can be cached
- Good for capturing spatial locality, short strides

Fully associative cache

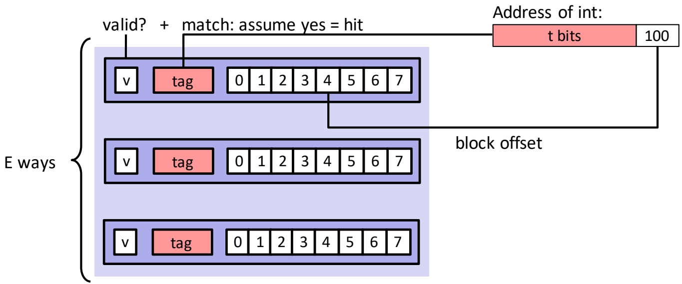


Figure: Fully associative cache. Image credit CS:APP

t-bit tag

- here, $t = m - b = m - 3$
- When CPU wants to read from or write to memory, all t-bits in tag need to match for read/write hit.

Fully associative cache

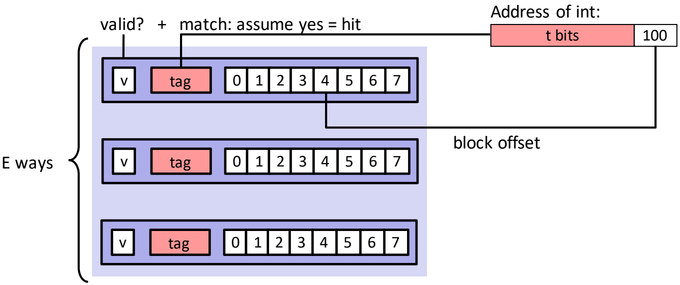


Figure: Fully associative cache. Image credit CS:APP

Full associativity

- Blocks can go into any of E ways
- Here, $E = 3$
- Good for capturing temporal locality: cache hits can happen even with intervening reads and writes to other tags.

Fully associative cache

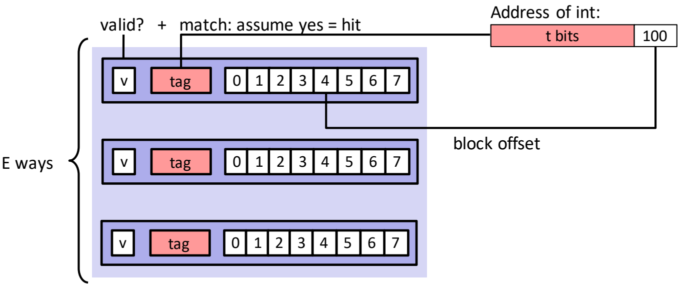


Figure: Fully associative cache. Image credit CS:APP

Capacity of cache

- Total capacity of fully associative cache in bytes: $C = EB = E * 2^b$
- Here, $C = E * 2^b = 3 * 2^3 = 24$ bytes

Fully associative cache

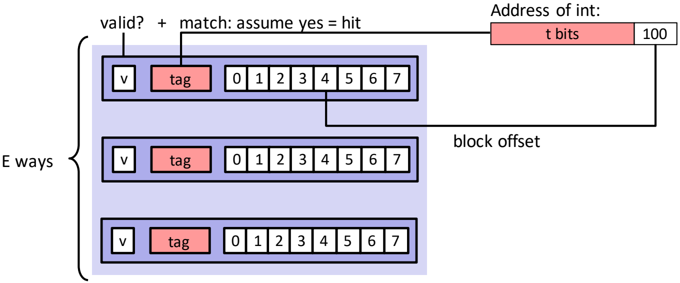


Figure: Fully associative cache. Image credit CS:APP

Strengths

- Blocks can go into any of *E*-ways.
- Hit rate is only limited by total capacity.

Weaknesses

- Searching across all valid tags is expensive.
- Figuring out which block to evict on read/write miss is also expensive.
- Requires a lot of