CSC290/420 ML Systems for Efficient AI Memory

Sreepathi Pai

September 22, 2025

URCS

Outline

Memory Technologies and Performance Metrics

Generic Memory Organization

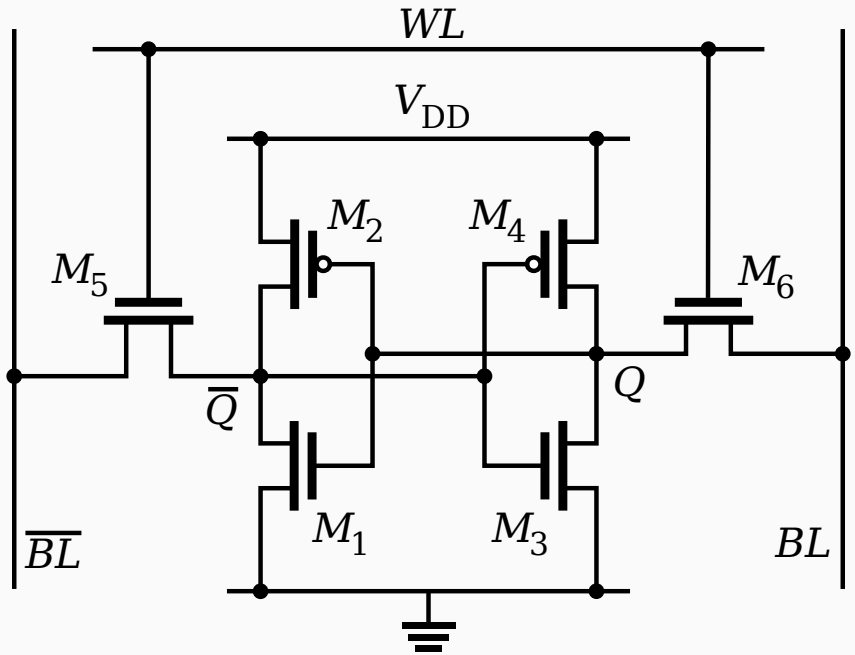
Caches and the Memory Hierarchy

Outline

Volatility

- Computer systems contain two types of memories based on volatility
  - Whether the memory retains its contents on loss of power
- Non-volatile memory: "Storage"
  - Hard disks, Flash Drives
  - But see also: Optane
- Volatile memory: "Main memory"
  - DRAM, VRAM
  - Caches
  - Registers

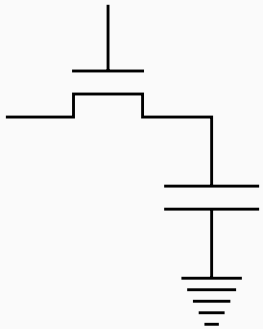
SRAM



- Uses 6 transistors to store 1 bit of data
- Used to implement registers (on GPUs) and caches
    - newer caches may be embedded DRAM

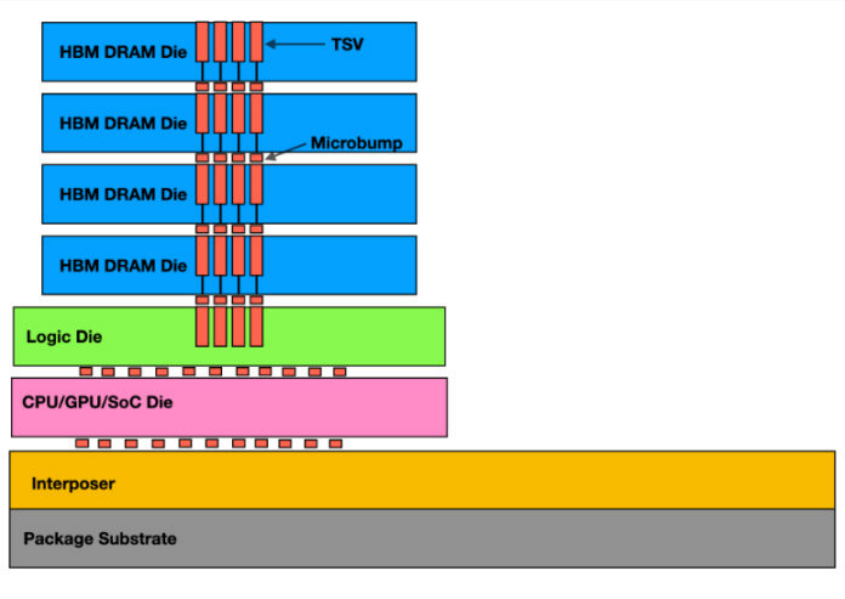
By Inductiveload - Own work, Public Domain, `https://commons.wikimedia.org/w/index.php?curid=114648423`

DRAM



- Data stored using a single transistor and a capacitor
- Charge in capacitor "leaks" over time
- Needs to be refreshed periodically
- Used for "main memory"
- Video RAM (VRAM) is actually just DRAM
    - technologically

HBM



- DRAM dies stacked on top of each other
- Connected using *thru-silicon vias*
- Connected to compute using a very wide interface
  - "Silicon Interposer"

By Blocks and Files `https://blocksandfiles.com/2025/06/26/four-more-hbm-generations/`

Disaggregated Memory

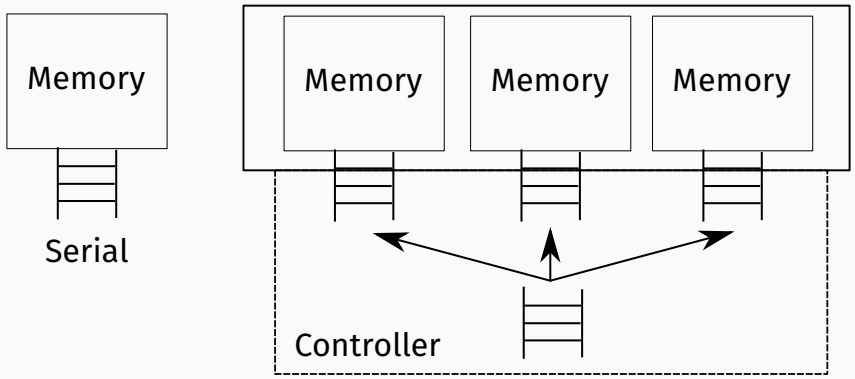
- What if we build a box full of memory chips and connect it to a computer through a special network?
  - You get "disaggregated" memory
- Special network is usually "Compute Express Link" (CXL)

Metrics

- Density
  - How many bits per unit area / volume
  - Higher is better
- Latency
  - Time to access data: clock cycles or nanoseconds
  - Lower is better
- Bandwidth
  - Data transferred per unit time, bits/cycle, bits/ns, bytes/s, etc.
  - Higher is better

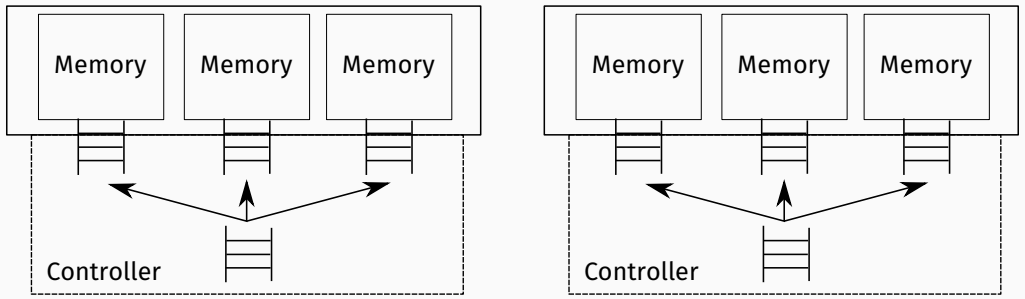
Outline

Building Blocks



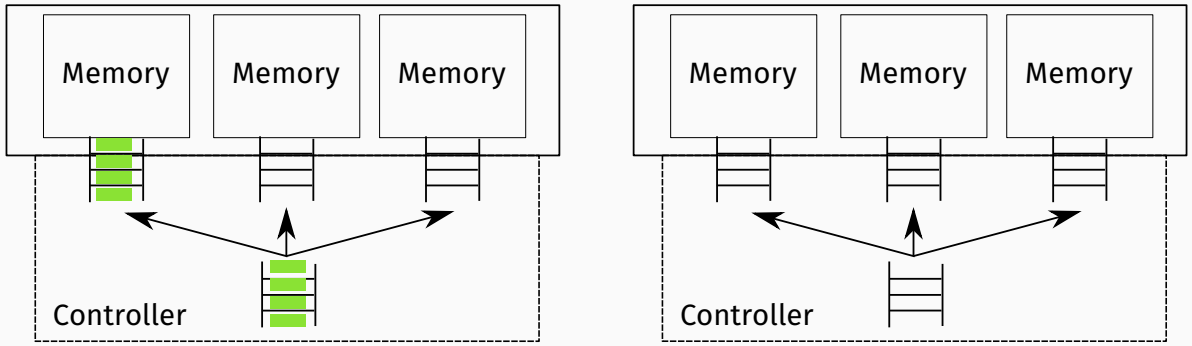
- Memory (systems) can be abstracted as a aggregation of serial memories
  - Each memory can process only one request at a time
- Access to each memory is mediated through a queue
  - May not physically exist for every memory
- Serial memories are called “banks”, (memory) “partitions”, etc.

Performance Modeling



- Most computer systems have multiple controllers
  - Cache controllers, Memory controllers (DRAM/HBM)
- Memory is *partitioned* among the controllers
- Controllers may be partitioned among CPU sockets/cores
  - Non-uniform Memory Access (NUMA)
- Queues have a queueing discipline: FCFS, FR-FCFS, etc.
  - Most systems also prioritize reads over writes (why?)
- The number of requests in queues and being serviced is termed *memory level parallelism* (MLP)

The 'Partition Camping' Problem



- What happens if a program only accesses a single partition of memory?
  - Or a single controller?
- How should addresses be distributed across partitions to prevent this?

Bank Conflicts in GPU Shared Memory

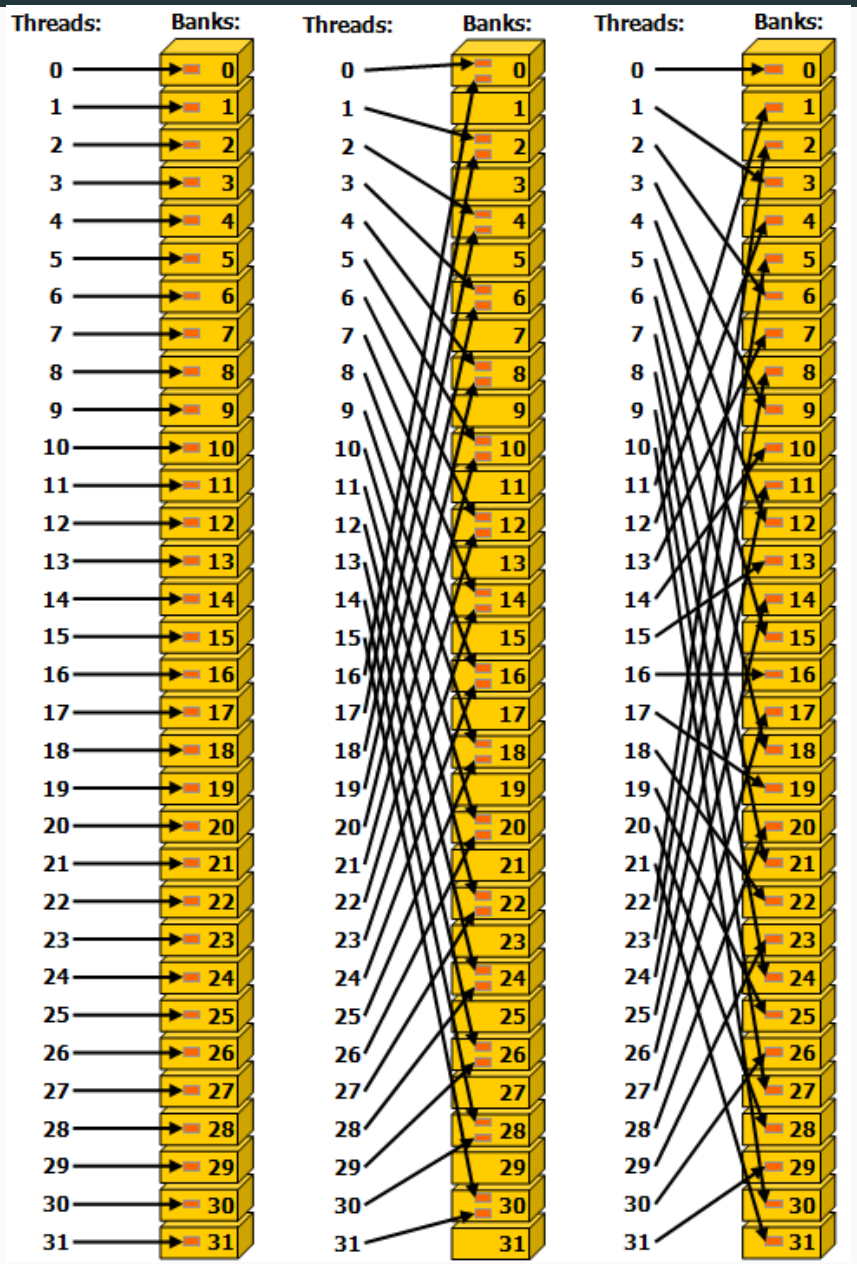


Figure from https://docs.nvidia.com/cuda/cuda-c-programming-guide/#compute-capability-5-x by NVIDIA

MLP Code Example 1

```
for(row=0; row < NROWS; row++)
   for(col=0; col < NCOLS; col++)
       out[row * NCOLS + col] = in[col * NROWS + row];
```

MLP Code Example 2: Indirect Memory Access

Sparse graph using CSR representation with data on each edge stored in a edgedata array.

```
for(row = 0; row < NROWS; row++) {
  for(j = row_start[row]; j < row_start[row+1]; j++) {
      sum += edgedata[col[j]]
    }
}
```

MLP Code Example 3: Pointer chasing

Linked list search:

```
current = head;

while(current) {
    if(current->value == search) return current;
    current = current->next;
}
```

Outline

Cache

(Roughly) a cache is a storage location that is faster to access than the original location.

- Cache type: Cache location, Original location
  - Browser: Disk, Website
  - Google: Google, Website
  - OS Disk Cache: RAM, Disk

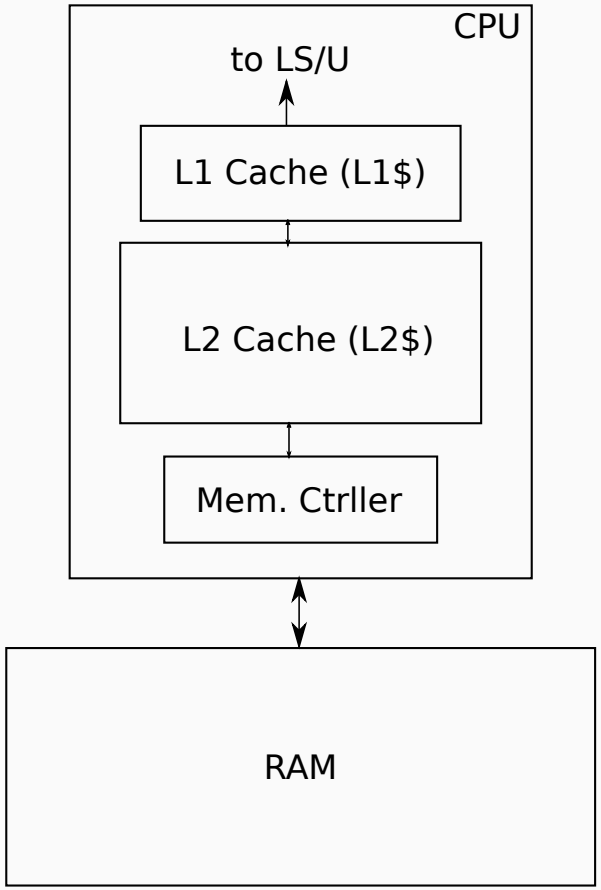
Hardware Cache

A hardware cache (specifically a CPU cache) is a small amount of fast (usually SRAM) memory in the CPU.

Q: Why not use this fast memory for building all of memory?

Using Hardware Caches

- The LSU asks the L1 cache for data at a specific address
    - if found, the L1 cache returns the data
- Otherwise, the L1 cache asks the L2 cache for the data
- And so on, until RAM is queried for the data
    - Actually on modern systems, disk is the last level (later lectures)
- Caches are transparent to the programmer



Cache Performance Benefits

- Cache hit: Cache contains the data you're looking for
- Cache miss: Cache must query the next level of memory

$$T_{memavg} = H_{L1} \times T_{L1} + (1 - H_{L1}) \times (H_{L2} \times T_{L2} + (1 - H_{L2}) T_{RAM})$$

- H_{L1} is the L1 hit rate, H_{L2} is L2 hit rate, etc.
  - Number of memory accesses that hit / Number of memory accesses
- Assume you have an average L1 hit rate of 100%
  - T_{L1} is 1 cycles[1]
  - T_{L2} is around 10 cycles
  - T_{RAM} is 100 cycles
- What is average time for a memory access?

[1] Latency Numbers Every Programmer Should Know

Why Cache Memories work: Locality

- Most programs do not access memory randomly
- Their memory access patterns usually demonstrate *locality* (of reference)
  - Spatial locality: nearby memory locations will be accessed together
  - Temporal locality: a memory location will be accessed multiple times
- Spatial locality: transfer data in blocks
- Temporal locality: keep data around waiting for it to be reaccessed

Cache Addressing

- A cache is much smaller than the next level of memory
  - Assume it can store C entries, while next level has N entries, $N >> C$
- Mapping N entries to C can be done as follows:
  - Direct-mapped: Each cache entry can store one of a set of fixed addresses from N at one time
  - Associative: Each cache entry can store any address in N
  - Set-Associative: First-level direct-mapped, Second-level Associative

Cache Misses: The 3C Model

- Compulsory Misses
  - The miss that occurs when the data is first brought in
  - Can't avoid this miss (?)
- Conflict Misses
  - A miss that occurs in a direct-mapped cache, but would not occur in a similarly-sized associative cache
- Capacity Miss
  - A miss that occurs in an associative cache that would not occur in a larger sized associative cache

Cache Replacement

- When a cache fills up, an existing entry must be replaced to make room
- OPT policy
  - Remove entry that will furthest in the future to be accessed next
- LRU policy
  - Remove entry that was least recently used in the past
- MRU policy
  - Remove entry that was most recently used in the past
- Lots of other policies

Caches and Writes

When a program writes to a location, two policies can apply:

- Write-through: the write is sent back to the next level
- Write-back: the write is kept in cache to be written to the next level at some opportune time

On some caches, writes to a location not previously read will result in bringing that block to cache before the write is carried out.

Code that shows locality

```
clock_gettime(CLOCK_MONOTONIC_RAW, &start);

for(i = 0; i < N; i++) {
  if(a[i] > max) max = a[i];
}

clock_gettime(CLOCK_MONOTONIC_RAW, &end);
```

About 13ms on my laptop, with `perf stat -e cache-misses,cache-references` showing:

```
Performance counter stats for './locality' (10 runs):

           622,559      cache-misses              #   87.861 % of all c
           708,570      cache-references
```

Code that may not show locality

```
clock_gettime(CLOCK_MONOTONIC_RAW, &start);

for(i = 0; i < N; i++) {
  if(a[b[i]] > max) max = a[i];
}

clock_gettime(CLOCK_MONOTONIC_RAW, &end);
```

- In the above code, `b[i] = i` (i.e. identity)
- About 14ms on my laptop, with `perf stat -e cache-misses,cache-references` showing:

```
Performance counter stats for './nolocality1' (10 runs):

         1,222,210      cache-misses              #   86.575 % of all c
         1,411,733      cache-references
```

Code that does not show locality

```
clock_gettime(CLOCK_MONOTONIC_RAW, &start);

// b[i] is now a random permutation of numbers from 0 to N-1
for(i = 0; i < N; i++) {
  if(a[b[i]] > max) max = a[i];
}

clock_gettime(CLOCK_MONOTONIC_RAW, &end);
```

- In the above code, `b` is a random permutation of the numbers 0 to N-1
- About 65ms on my laptop, with `perf stat -e cache-misses,cache-references` showing:

```
Performance counter stats for './nolocality2' (10 runs):

        10,857,423      cache-misses              #   70.655 % of all c
        15,366,835      cache-references
```

Locality in code

- Keep data you want to access together near each other (spatial locality)
  - e.g., use arrays
  - use a custom memory allocator for pointer-based structures if possible
- Reuse data as much as possible (temporal locality)
  - techniques called “blocking” / “tiling”

Inverted Memory Hierarchies

- The size of each memory level usually decreases as we get closer to the CPU
  - Size of registers is smaller than L1 which is smaller than L2, and so on.
- On most GPUs, this is not true
  - Size of registers (64K * 4 bytes = 256KBytes) is much larger than L1 (around 64-128K)
  - On GPUs, data can probably be kept in registers
  - Unfortunately, registers are not transparent to programmers

Scratchpad Memory

- On-chip memory, sometimes called scratchpad memory, is a small amount of memory on chip.
- Very fast.
- Special load/store instructions
- Programmer must load data into scratchpad and store it back to main memory (if needed)
- On NVIDIA GPUs, this is called "shared memory"

Beyond Load/Store: DMA

- Reading and writing memory is slow for a CPU
- Offload work to a *direct memory access* (DMA) controller
  - Program start address, destination address, and length
- DMA engine will carry out copy *asynchronously* and let the CPU know when its done
  - CPU can work on other tasks

Next Class

- Cache coherence
- Virtual memory