Memory Hierarchy

Philipp Koehn

14 October 2019

Large and Fast

- We want: lots of memory and access it fast
- We really have: different speed/size tradeoffs
- Need methods to give illusion of large and fast memory

Locality

- What helps us is locality

- Temporal locality
  - same memory location often referenced repeatedly
  - example: instructions in loops

- Spatial locality
  - after an item is referenced
  - example: processing of sequential data

Example: Violation of Locality

- Consider this C code

```
#define size 32768
int matrix[size][size];
int main(void) {
  for(int i = 0; i<size; i++) {
    for(int j = 0; j<size; j++) {
      matrix[i][j] = 47;
    }
  }
  return 0;
}
```

- How fast does it run?

```
% time ./a.out
real 0m3.824s
user 0m2.533s
sys 0m1.118s
```

- Minor change

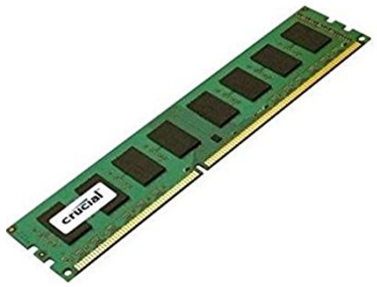
```
#define size 4096
int matrix[size][size];
int main(void) {
  for(int i = 0; i<size; i++) {
    for(int j = 0; j<size; j++) {
      matrix[j][i] = 47;
    }
  }
  return 0;
}
```

- How fast does it run?

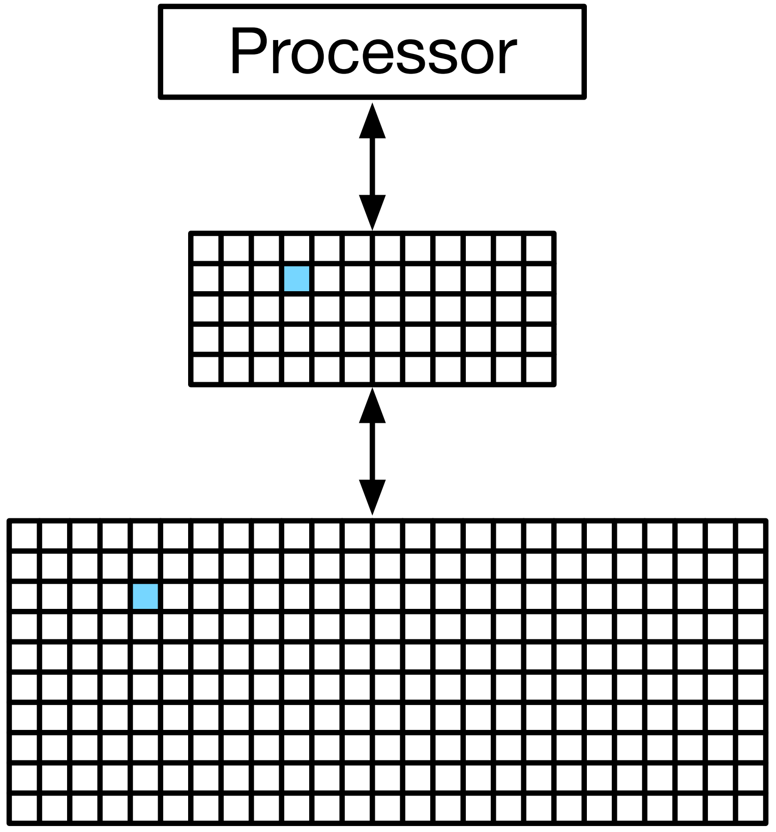
```
% time ./a.out
real 0m25.129s
user 0m23.841s
sys 0m1.272s
```

Memory Types

Technology	Speed	Capacity	Cost
SRAM on CPU	fastest	smallest	highest
DRAM on motherboard	...	...	...
Flash memory	...	...	....
Magnetic disk	slowest	biggest	lowest

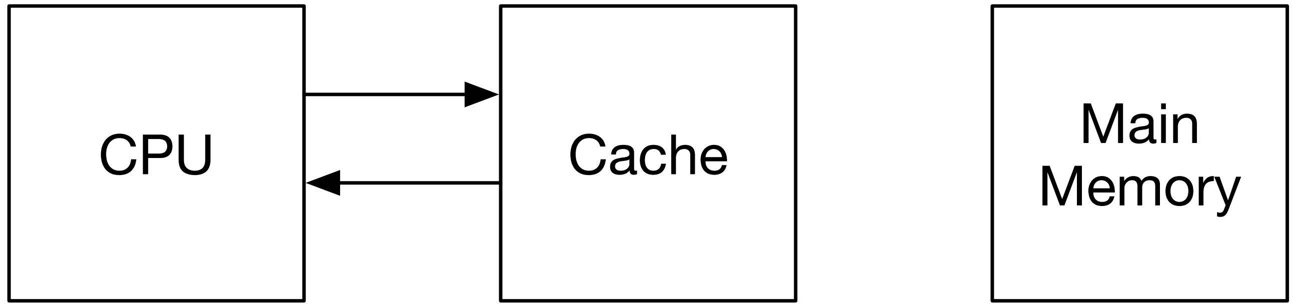


2 Level Memory



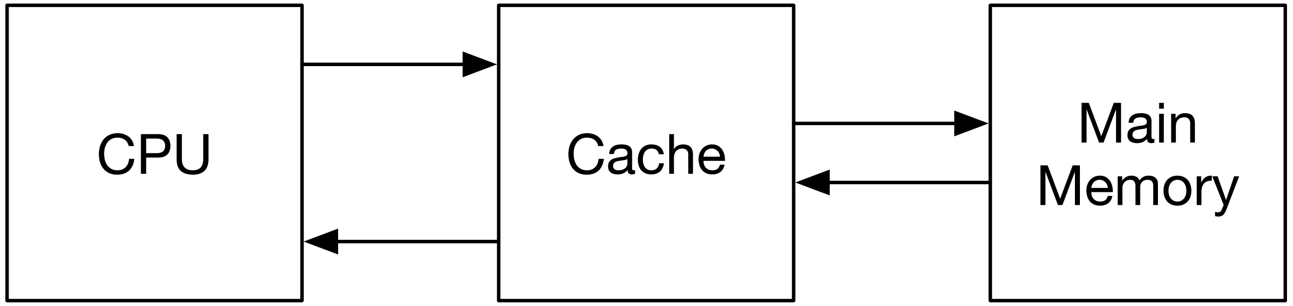
Smaller memory mirrors some of the large memory content

Cache Hit



- Memory request from CPU
- Data found in cache
- Send data to CPU

Cache Miss



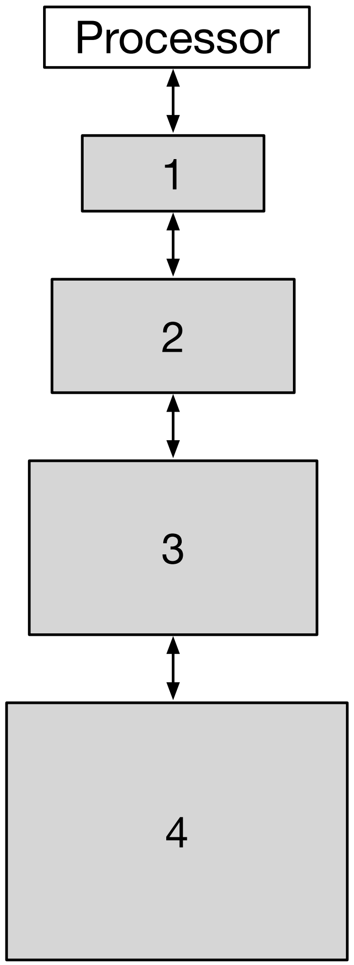
- Memory request from CPU
- Data **not** found in cache
- Memory request from cache to main memory
- Send data from memory to cache
- Store data in cache
- Send data to CPU

Concepts

- Memory has to be transferred from large memory to be used
- **Cache:** small memory connected to processor
- **Block:** unit of memory transferred
- **Hit rate:** fraction of memory lookups served by data already in cache
- **Miss rate:** fraction of memory lookups that require memory transfers
- **Hit time:** time to process a cache hit
- **Miss penalty:** time to process a cache miss

Memory Hierarchy

- More than 2 levels of memory
- Transfer between memory in level *i* and *i+1* follows same principle, regardless of *i*
- Hierarchy: if item in level *i*,
  then it is also in level *i+1*
- Hence, we restrict our discussion to 2 levels



memory technologies

Current Technologies

Technology	Access Time	Price per GB
SRAM semiconductor	0.5-2.5ns	$300
DRAM semiconductor	50-70ns	$6
Flash semiconductor	5,000-50,000ns	$0.40
Magnetic disk	5,000,000-20,000,000ns	$0.02
Magnetic tape	-	$0.008

(prices from 2018)

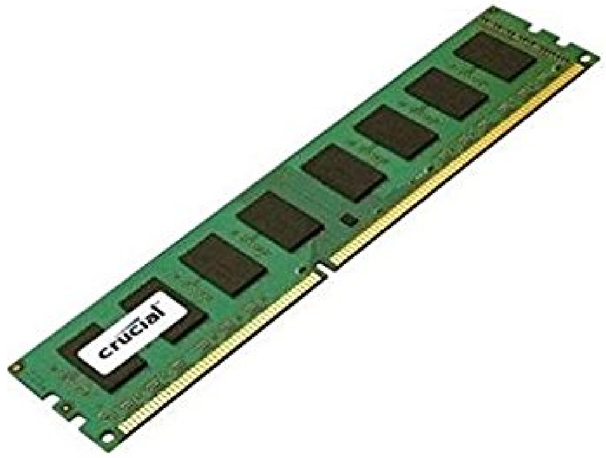
SRAM

- Integrated in CPU, runs at similar clock speeds
- Implemented using flip flops
- Uses more transistors than DRAM

DRAM

- Separate chips on the motherboard
- In PCs and servers, multiple chips on a module (DIMM)
- Implemented using capacitors

  lose charge $\rightarrow$ need to be frequently refreshed
- Lose charge when power is turned off



Flash Memory

- A type of EEPROM

  (electrically erasable programmable read-only memory)
  - allows read of individual bytes
  - writes require erase of a block, rewrite of bytes
- Writes can wear out the memory
- Currently becoming standard storage memory for laptops

Hard Drives

- Magnetic charge on spinning disk
- Read/write requires read head at the right place
- Sequential data reads are relatively fast
- Random access slow $\rightarrow$ not practical as process memory

cache basics

Cache

- All data is in large main memory
- Data for processing has to moved to cache
- Caching strategies
  - mapping between cache and main memory
  - which data to read / keep / write

Direct Mapping

- Idea: keep mapping from cache to main memory simple

⇒ Use part of the address as index to cache

- Address broken up into 3 parts
  - memory position in block (offset)
  - index
  - tag to identify position in main memory

- If blocks with same index are used, older one is overwritten

Direct Mapping: Example

- Main memory address (32 bit)

```
0010 0011 1101 1100 0001 0011 1010 1111
```

- Block size: 1KB (10 bits)
- Cache size: 1MB (20 bits)

0010 0011 1101	1100 0001 00	11 1010 1111
Tag	Index	Offset

Cache Access Example

- Cache content

Index	Valid	Tag	Mapped Memory
000	no		
001	no		
010	no		
011	no		
100	no		
101	no		
110	no		
111	no		

Cache Access Example

- Cache content

Index	Valid	Tag	Mapped Memory
000	no		
001	no		
010	no		
011	no		
100	no		
101	yes	10	10101
110	no		
111	no		

- Operation: read 10101
  - cache miss
  - retrieve value from main memory

Cache Access Example

- Cache content

Index	Valid	Tag	Mapped Memory
000	no		
001	no		
010	yes	11	11010
011	no		
100	no		
101	yes	10	10101
110	no		
111	no		

- Operation: read 11010
  - cache miss
  - retrieve value from main memory

Cache Access Example

- Cache content

Index	Valid	Tag	Mapped Memory
000	no		
001	no		
010	yes	11	11010
011	no		
100	no		
101	yes	10	10101
110	no		
111	no		

- Operation: read 10101
  - cache hit

Cache Access Example

- Cache content

Index	Valid	Tag	Mapped Memory
000	no		
001	no		
010	yes	11	11010
011	no		
100	no		
101	yes	10	10101
110	no		
111	no		

- Operation: read 11010
  - cache hit

Cache Access Example

- Cache content

Index	Valid	Tag	Mapped Memory
000	yes	10	10000
001	no		
010	yes	11	11010
011	no		
100	no		
101	yes	10	10101
110	no		
111	no		

- Operation: read 10000
  - cache miss
  - retrieve value from main memory

Cache Access Example

- Cache content

Index	Valid	Tag	Mapped Memory
000	yes	10	10000
001	no		
010	yes	11	11010
011	yes	00	00011
100	no		
101	yes	10	10101
110	no		
111	no		

- Operation: read 00011
  - cache miss
  - retrieve value from main memory

Cache Access Example

- Cache content

Index	Valid	Tag	Mapped Memory
000	yes	10	10000
001	no		
010	yes	11	11010
011	yes	00	00011
100	no		
101	yes	10	10101
110	no		
111	no		

- Operation: read 10000
  - cache hit

Cache Access Example

- Cache content

Index	Valid	Tag	Mapped Memory
000	yes	10	10000
001	no		
010	yes	10	10010
011	yes	00	00011
100	no		
101	yes	10	10101
110	no		
111	no		

- Operation: read 10010
  - cache miss
  - retrieve value from main memory
  - overwrite existing cache value

Block Size Tradeoffs

- Larger block size
  - fewer cache misses due to spatial locality
  - longer transfer times of block
  - fewer blocks in cache $\rightarrow$ more competition for cache

- In practice
  - optimal value somewhere in the middle
  - depends on running process