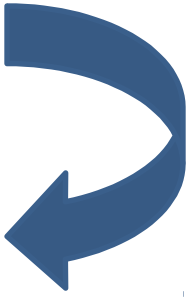
Cache

Outline

- Memory organization
- Caching
  - Different types of locality
  - Cache organization
- Cachelab
  - Warnings are errors
  - Part (a) Building Cache Simulator
  - Part (b) Efficient Matrix Transpose
- Blocking

Memory Hierarchy

- Registers
- SRAM



Today: we study this interaction to give you an idea how caching works

- DRAM
- Local Secondary storage

SRAM vs DRAM Trade-off

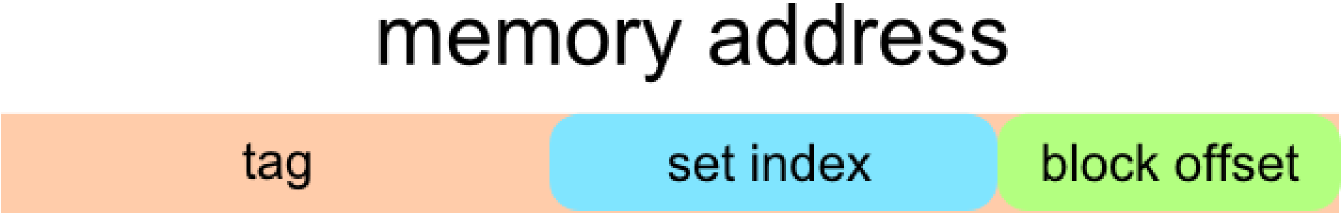
- SRAM (cache)
  - Faster (L1 cache: 1 CPU cycle)
  - Smaller (Kilobytes (L1) or Megabytes (L2))
  - More expensive and “energy-hungry”
- DRAM (main memory)
  - Relatively slower (hundreds of CPU cycles)
  - Larger (Gigabytes)
  - Cheaper

Caching

- Temporal locality
  - A memory location accessed is likely to be accessed again multiple times in the future
  - After accessing address X in memory, save the bytes in cache for future access
- Spatial locality
  - If a location is accessed, then nearby locations are likely to be accessed in the future.
  - After accessing address X, save the block of memory around X in cache for future access

Memory Address

- 64-bit on shark machines



- Block offset: b bits
- Set index: s bits

Cache

- A cache is a set of 2^s *cache sets*

- A *cache set* is a set of E *cache lines*
  - E is called associativity
  - If E=1, it is called “direct-mapped”

- Each *cache line* stores a block
  - Each block has 2^b bytes

Cachelab

- Warnings are errors!
- Include proper header files
- Part (a) Building a cache simulator
- Part (b) Optimizing matrix transpose

Warnings are Errors

- Strict compilation flags
- Reasons:
  - Avoid potential errors that are hard to debug
  - Learn good habits from the beginning

Part (a) Cache simulator

- A cache simulator is NOT a cache!
  - Memory contents NOT stored
  - Block offsets are NOT used
  - Simply counts hits, misses, and evictions

- Your cache simulator need to work for different s, b, E, given at run time.
- Use LRU replacement policy

Cache simulator: Hints

- A cache is just 2D array of *cache lines*:
  - struct cache_line cache[S][E];
  - S = 2^s, is the number of sets
  - E is associativity
- Each cache_line has:
  - Valid bit
  - Tag
  - LRU counter

Part (b) Efficient Matrix Transpose

- Matrix Transpose (A -> B)

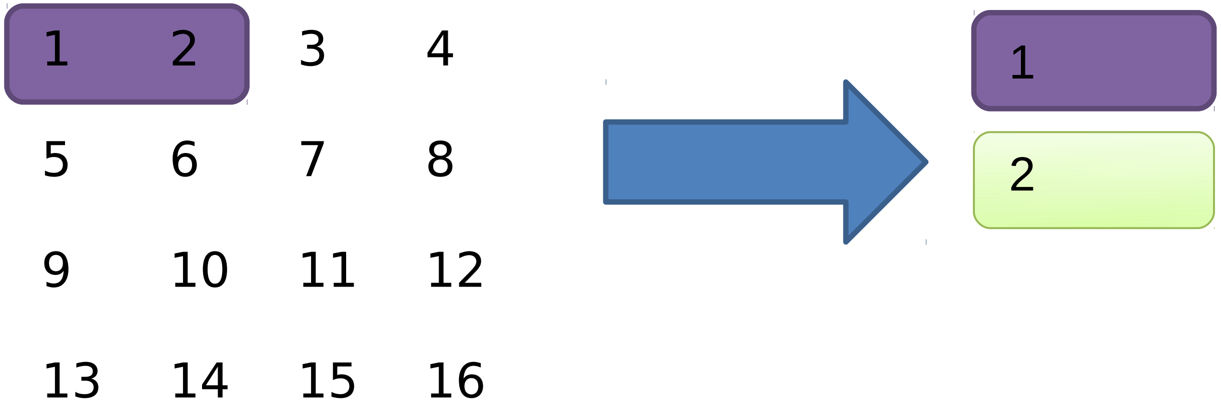
Matrix A

Matrix B

$$\begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{matrix} \quad \Rightarrow \quad \begin{matrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{matrix}$$

Part (b) Efficient Matrix Transpose

- Matrix Transpose (A -> B)
- Suppose block size is 8 bytes (2 ints)



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Question: After we handle 1&2. Should we handle 3&4 first, or 5&6 first ?

Blocking

- What inspiration do you get from previous slide ?
  - Divide matrix into sub-matrices
  - This is called **blocking** (CSAPP2e p.629)
  - Size of sub-matrix depends on
    - cache block size, cache size, input matrix size
  - Try different sub-matrix sizes
- We hope you invent more tricks to reduce the number of misses !

Part (b)

- Cache:
  - You get 1 kilobytes of cache
  - Directly mapped (E=1)
  - Block size is 32 bytes (b=5)
  - There are 32 sets (s=5)
- Test Matrices:
  - 32 by 32, 64 by 64, 61 by 67

The End

- Good luck!