




Recitation 7 Caching

By yzhuang

Announcements

- Pick up your exam from ECE course hub
 - Average is 43/60
 - Final Grade computation? See syllabus <http://www.cs.cmu.edu/~213/misc/syllabus.pdf>
- If you download cachelab before noon of September 30, you should re-download the tarball. See the writeup for details.

Memory Hierarchy

- Registers
 - SRAM
 - DRAM
 - Local Secondary storage
 - Remote Secondary storage
- 
- Today: we study this interaction to give you an idea how caching works

SRAM vs DRAM

- SRAM (cache)
 - Faster (L1 cache: 1 CPU cycle)
 - Smaller (Megabytes)
 - More expensive
- DRAM (main memory)
 - Relatively slower (100 CPU cycles)
 - Larger (Gigabytes)
 - Cheaper

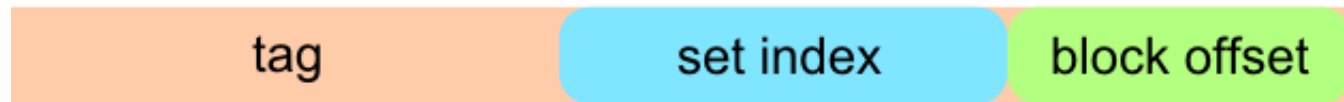
Caching

- 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

memory address



- 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

- Part (a) Building a cache simulator
- Part(b) Optimizing matrix transpose

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 \rightarrow B)

Matrix A

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

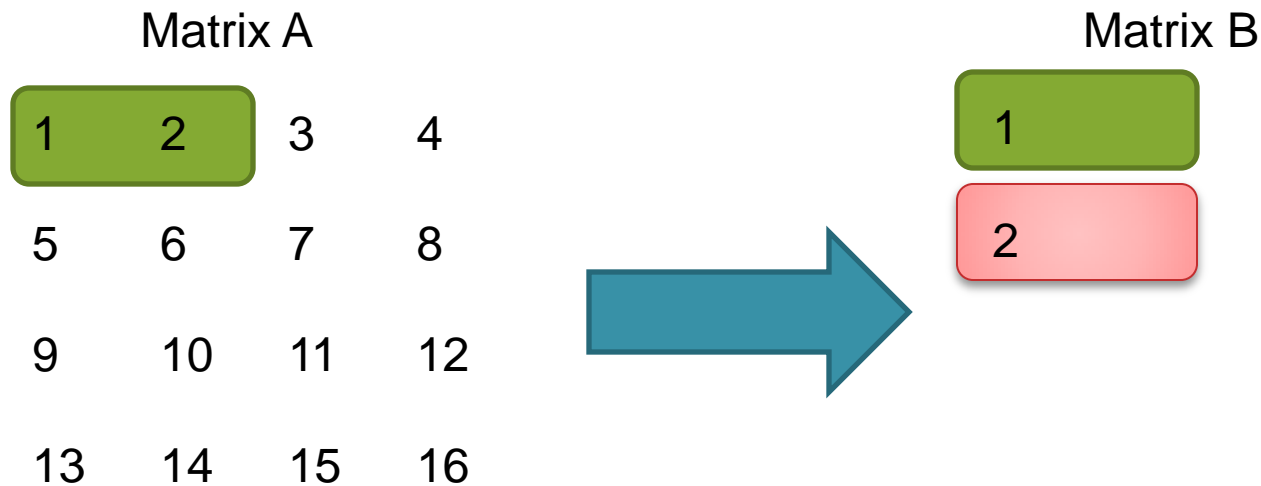


Matrix B

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

Part (b) Efficient Matrix Transpose

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



Access A[0][0] cache miss
Access B[0][0] cache miss
Access A[0][1] cache hit
Access B[1][0] cache miss

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

Part (b) Hint

- 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!