15-213 Recitation 7
Caches and Blocking

9 October 2017

Agenda

- Reminders
- Revisiting Cache Lab
- Caching Review
- Blocking to reduce cache misses
- Cache alignment

Reminders

- Cache Lab is due Thursday!
- Exam1 is just a week away!
- Start doing practice problems.
- Come to the review session.

Reminders: Cache Lab

- Two parts
- Write a cache simulator
  - Hopefully you've started this part by now
- Optimize some code to minimize cache misses
- Programming style will be graded starting now
  - Worth about a letter grade on this assignment
- See the appendix. But that’s incomplete, so also see the style guide on the course website.
- Other details are in the writeup.

Cache Lab: Parsing Input with fscanf

- fscanf() is exactly like scanf() except that you specify a stream to use (i.e. an open file) instead of always reading from standard input

- The parameters to fscanf are
- 1: a stream pointer of type FILE*, e.g. from fopen()
- 2: a format string specifying how to parse the input
- 3+: pointers to each of the variables that will store the parsed data

- fscanf() returns -1 if the data does not match the format string or there is no more input
- Use it to parse the trace files

fscanf() Example

```
FILE *pFile; /* pointer to FILE object */
pFile = fopen("trace.txt","r"); /* open trace file for reading */
/* verify that pFile is non-NULL! */
char access_type;
unsigned long address;
int size;
/* line format is " S 2f,1" or " L 7d0,3" */
/* so we need to read a character, a hex number, and a decimal number */
/* put those in the format string along with the fixed formatting */
while (fscanf(pFile," %c %lx,%d", &access_type, &address, &size) > 0) {
   /* do stuff */
}
fclose(pFile);  /* always close file when done */
```

Cache Lab: Cache Simulator Hints

- You are only counting hits, misses, and evictions
- Use LRU (Least Recently Used) replacement policy
- Structs are a great way to bundle up the different parts of a cache line (valid bit, tag, LRU counter, etc.)
- A cache is like a 2D array of cache lines
  - One dimension represents set associativity E, the other the number of sets S:

```
struct cache_line cache[S][E];
```

- Your simulator needs to handle different values of S, E, and b (block size) given at run time
  - Allocate your space dynamically

Class Question / Discussions

- We’ll work through a series of questions
- Write down your answer for each question
- You can discuss with your classmates

What Type of Locality?

- The following function exhibits which type of locality? Consider *only* array accesses.

```
void who(int *arr, int size) {
  for (int i = 0; i < size-1; ++i)
    arr[i] = arr[i+1];
}
```

A.	Spatial
B.	Temporal
C.	Both A and B
D.	Neither A nor B

What Type of Locality?

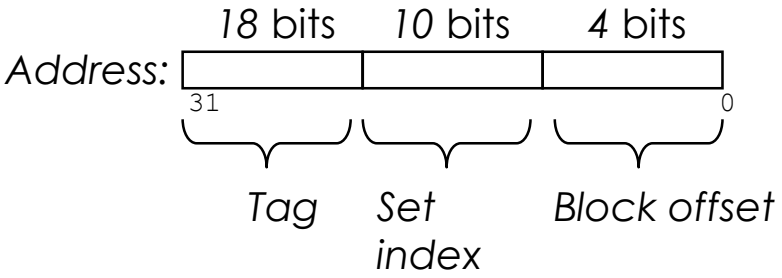
- The following function exhibits which type of locality? Consider *only* array accesses.

```
void coo(int *arr, int size) {
  for (int i = size-2; i >= 0; --i)
    arr[i] = arr[i+1];
}
```

A.	Spatial
B.	Temporal
C.	Both A and B
D.	Neither A nor B

Calculating Cache Parameters

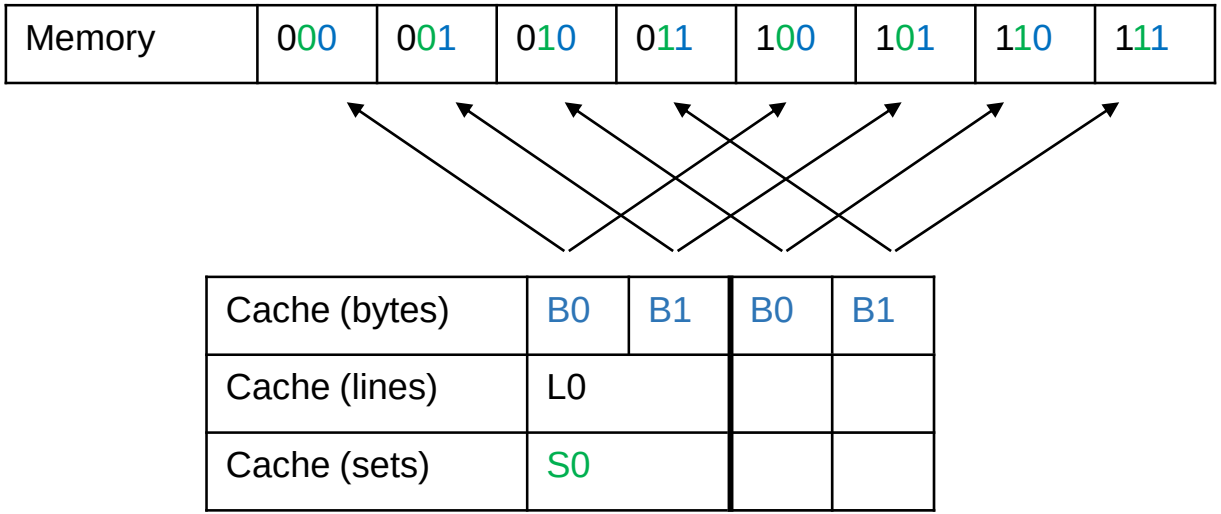
- Given the following address partition, how many `int` values will fit in a single data block?



	# of int in block
A.	0
B.	1
C.	2
D.	4
E.	Unknown: We need more info

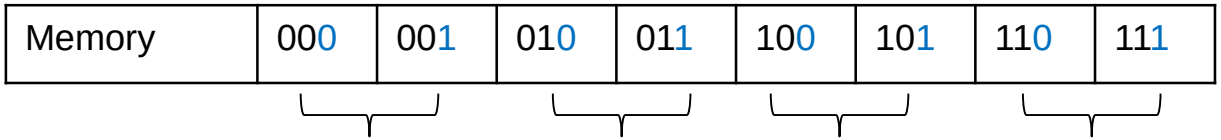
Interlude: terminology

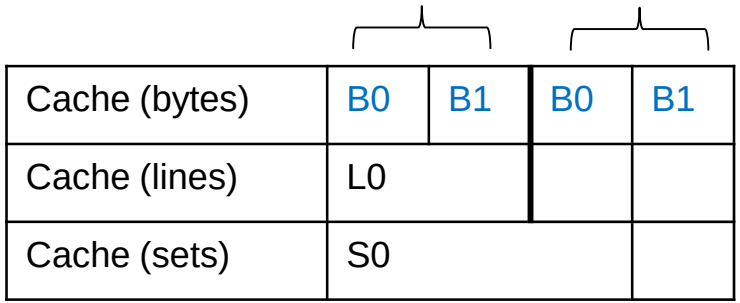
- A **direct-mapped** cache only contains one line per set. This means E = 2^e = 1.



Interlude: terminology

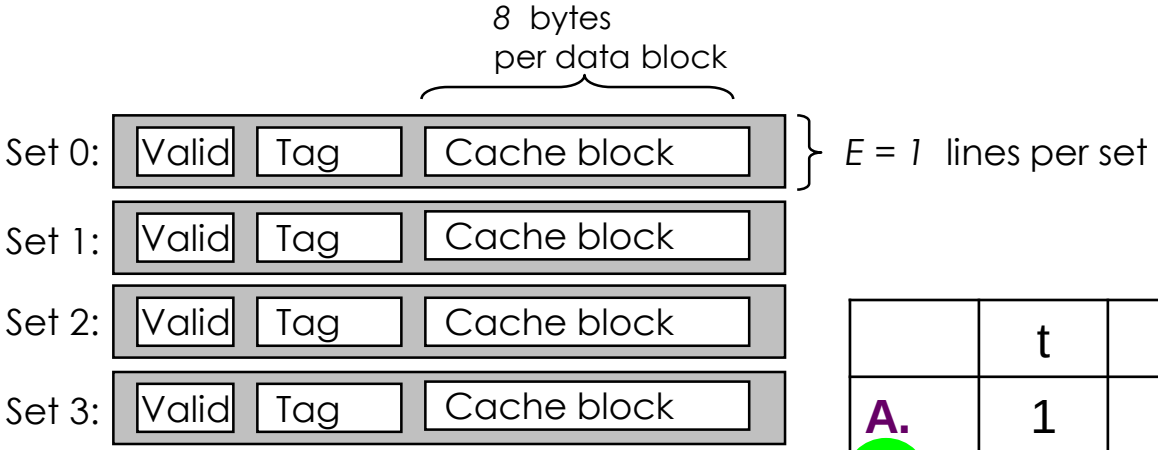
- A **fully associative** cache has 1 set, and many lines for that one set. This means S = 2^s = 1.





Direct-Mapped Cache Example

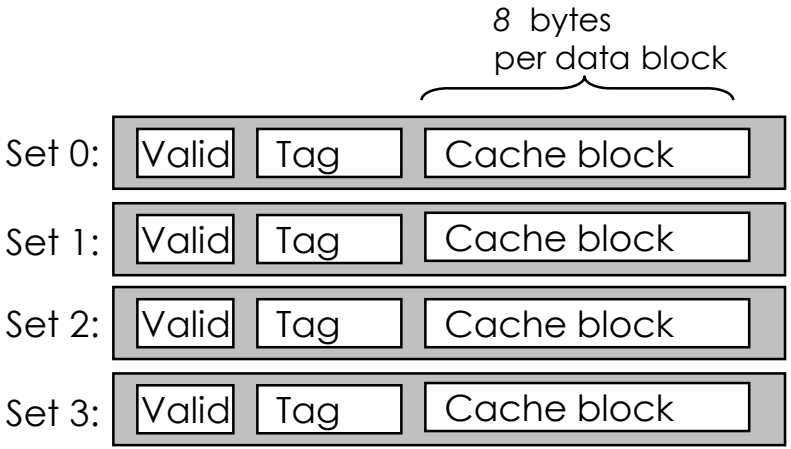
- Assuming a 32-bit address (i.e. m=32), how many bits are used for tag (t), set index (s), and block offset (b).

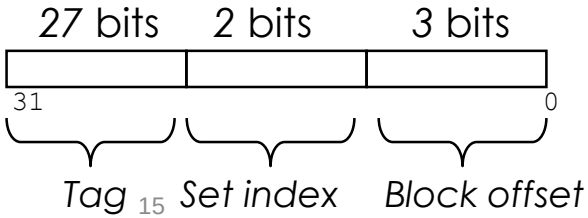


	t	s	b
A.	1	2	3
B.	27	2	3
C.	25	4	3
D.	1	4	8
E.	20	4	8

Which Set Is it?

- Which set may the address **0xFA1C** be located in?

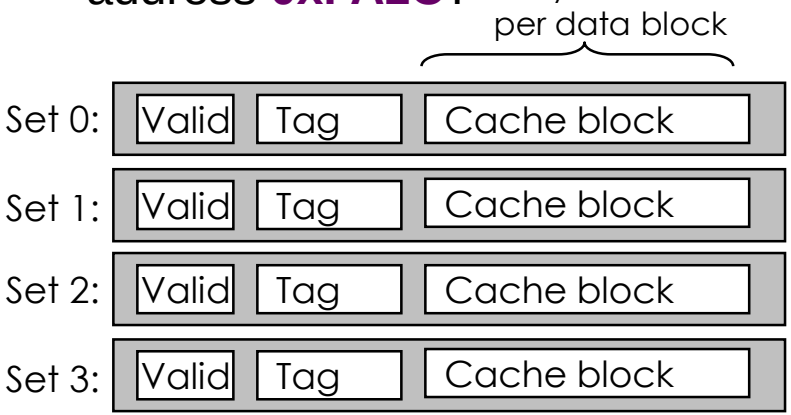


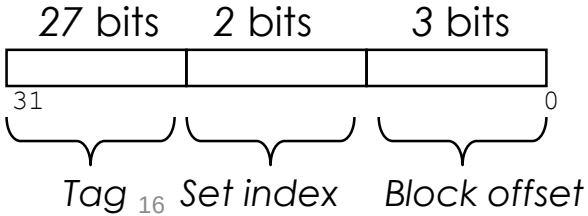


	Set # for 0xFA1C
A.	0
B.	1
C.	2
D.	3
E.	More than one of the above

Cache Block Range

- What range of addresses will be in the same block as address **0xFA1C**?





	Addr. Range
A.	0xFA1C
B.	0xFA1C – 0xFA23
C.	0xFA1C – 0xFA1F
D.	0xFA18 – 0xFA1F
E.	It depends on the access size (byte, word, etc)

Cache Misses

If N = 16, how many bytes does the loop access of A?

```
int foo(int* a, int N)
{
    int i, sum = 0;
    for(i = 0; i < N; i++)
        sum += a[i];
    return sum;
}
```

	Accessed Bytes
A	4
B	16
C	64
D	256

Cache Misses

If there is a 48B cache with 8 bytes per block and 3 cache lines per set, how many misses if foo is called twice?
N still equals 16

```
int foo(int* a, int N)
{
    int i, sum = 0;
    for(i = 0; i < N; i++)
        sum += a[i];
    return sum;
}
```

	Misses
A	0
B	8
C	12
D	14
E	16

Cache-Friendly Code

- Keep memory accesses bunched together
  - In both time and space (address)
- The working set at any time should be smaller than the cache
- Avoid access patterns that cause conflict misses
- Align accesses to use fewer cache sets (often means dividing data structures into pieces whose sizes are powers of 2)

Blocking Example

- We have a 2D array `int[4][4] A;`
- Cache is fully associative and can hold two lines
- Each line can hold two `int` values

- Discuss the following questions with your neighbor:

- What is the best miss rate for traversing `A` once?
- What order does of traversal did you use?

- What other traversal orders can achieve this miss rate?

Class Discussion

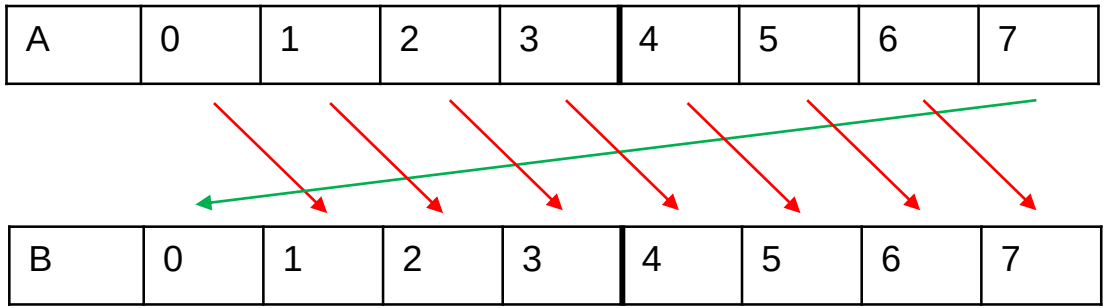
- What did the optimal transversal orders have in common?
- How does the pattern generalize to `int[8][8] A` and a cache that holds 4 lines each of 4 `int`'s?

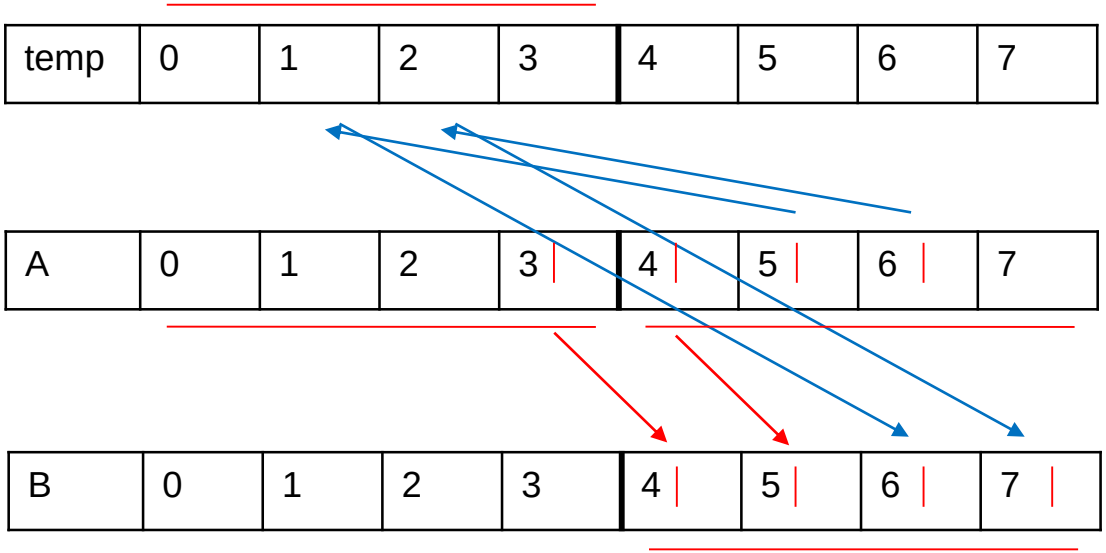
Cache-alignment

- Suppose you have arrays
- `int[8] A, B, temp;`
-
- A[0], B[0] and temp[0] all correspond to byte 0 of set 0 on the cache. We say that all three arrays are cache-aligned.
    - For example, suppose we use a direct-mapped cache. If we request first A[0] then B[0], the cache will evict the line containing A[0].

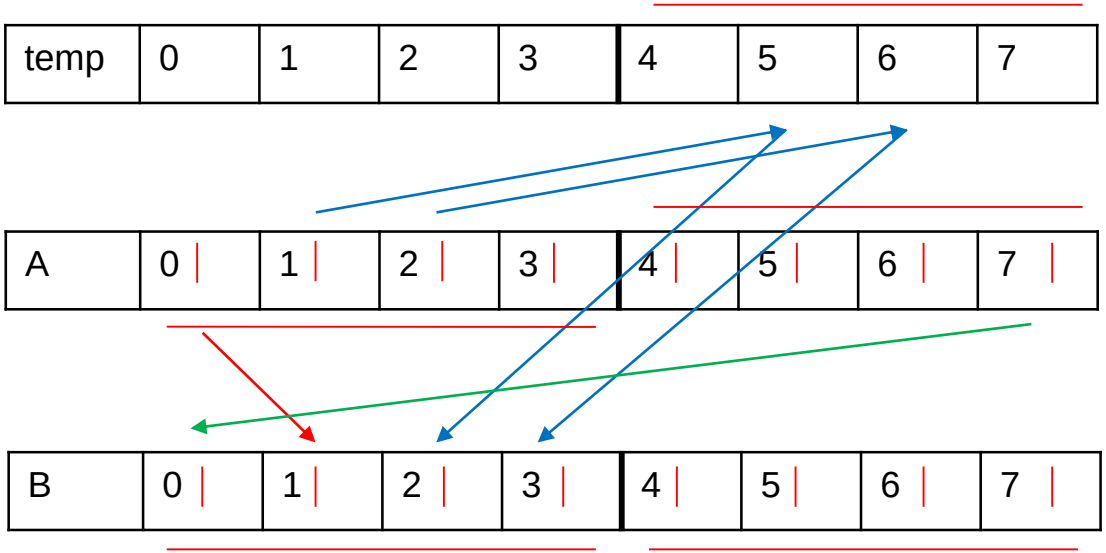
Very Hard Cache Problem

- We will use a direct-mapped cache with 2 sets, which each can hold up to 4 `int`'s.
- How can we copy A into B, shifted over by 1 position?
  - The most efficient way? (Use `temp`!)





Number of misses:



Number of misses:

← Could've been 16 misses otherwise!
We would save even more if the block size were larger, or if `temp` were already cached

If You Get Stuck

Please read the writeup

Read it again after doing ~25% of the lab

- CS:APP Chapter 6
- View lecture notes and course FAQ at http://www.cs.cmu.edu/~213
- Office hours Sunday through Thursday 5:00-9:00pm in WeH 5207
- Post a **private** question on Piazza
- `man malloc`, `man gdb`, gdb's `help` command
- http://csapp.cs.cmu.edu/public/waside/waside-blocking.pdf

Appendix: C Programming Style

- Properly document your code
  - Header comments, overall operation of large blocks, any tricky bits
- Write robust code – check error and failure conditions
- Write modular code
  - Use interfaces for data structures, e.g. create/insert/remove/free functions for a linked list
  - No magic numbers – use #define
- Formatting
  - 80 characters per line
  - Consistent braces and whitespace
- No memory or file descriptor leaks