18-344 Recitation 9:

Parallelism and Concurrency

Logistics

- Lab 3 due Tuesday Nov 7
- Lab4 released
  - Due ~~Nov 23~~ Nov 28
- Homework 7 due Thursday Nov 9
  - Grades should be published

Homework Review

HW7A

(2pts) The function `foo()` does 64 floating point operations and has no control flow logic. What is the operational intensity of this program in flops/byte? (Ignore address calculation, branch evaluation, and loop upkeep in your operation count).

64 flops /256 bytes = ¼ flops per byte

(1pts) The bandwidth of your off chip DRAM is 1 GiB/sec, and an average compute speed of 3 MFLOPs/sec. Are you Compute Bound or Memory Bound?

3 MLOPs/sec = 3 * (2^20) / (2^6) = 3 * (2^14) Executions of Foo per second
1 GiB/sec = (2^30 / 2^8) = 2^22 Executions of Foo per second

Compute Bound

```
// sizeof(data_struct_t) = 256
data_struct_t array[ARRAY_SIZE];

for (int idx = 0; idx < ARRAY_SIZE; idx++) {
      if(unlikely(array[idx].valid))
            foo(array[idx]);
}
```

HW7A

(2pts) Only 1 out of 256 elements in `array` have data. What is the effective operational intensity of this program in flops/byte? (Ignore address calculation, branch evaluation, and loop upkeep in your operation count).

(64 flops / 256)/256 bytes = 2^-10 flops per byte

(1pts) The bandwidth of your off chip DRAM is 1 GiB/sec, and an average compute speed of 3 MFLOPs/sec. What is the throughput of your system using the operational intensity calculated in the previous question? Are you Compute Bound or Memory Bound?

3 MFLOPs/sec = (3 * (2^20) * (2^8)/ (2^6) = 3 * (2^22) Executions of Foo per second
1 GiB/sec = (2^30 / 2^8) = 2^22 Executions of Foo per second

Memory Bound

```c
// sizeof(data_struct_t) = 256
data_struct_t array[ARRAY_SIZE];

for (int idx = 0; idx < ARRAY_SIZE; idx++) {
        if(unlikely(array[idx].valid))
                foo(array[idx]);
}
```

HW7B

Idea: Traverse the CSR to find the next destination which uses the source node

	D0	**D1**	**D2**	**D3**
S0	1	0	0	1
S1	1	0	1	0
S2	0	0	1	0
S3	0	1	0	1

HW7B

Idea: Traverse the CSR to find the next destination which uses the source node

```
int32_t replacement_score(int32_t src, int32_t dest) {
     int32_t start = OA_csr[src];
     int32_t end = src < NUM_NODES - 1 ? OA_csr[src+1] : NUM_EDGES;
     int32_t val = INT_MAX;
     for (int32_t j = start; j < end; j++) {
          if (NA_csr[j] > dest) {
               val = NA_csr[j];
               break;
          }
     }
     return val;
}
```

Memory Consistency

Memory Consistency

From Wikipedia:

In computer science, a **consistency model** specifies a contract between the programmer and a system, wherein the system guarantees that if the programmer follows the rules for operations on memory, memory will be consistent and the results of reading, writing, or updating memory will be predictable.

Sequential Consistency

- A thread perceives its own memory ops in program order
- Memory ops from threads in program order can be interleaved arbitrarily; different interleaving allowed on different runs
- For each run, all threads must not disagree on any orderings observed

Sequential Consistency Cont.

Assume *ptr_a = 0, *ptr_b = 0 initially.

Thread 1

```
{
…
Y = *ptr_b;
X = *ptr_a;
…
}
```

Thread 2

```
{
…

*ptr_a = 1;
*ptr_b = 1;
…
}
```

There is no interleaving of memory operations such that Y = 1 and X = 0

Memory Models across the System Stack

Language	**Compiler**	**Architecture**
Java/C++: SC for data-race-free programs	Conservative with reordering when d-r-f can’t be proved	Usually very weak for max optimization (lots of reordering) Note: fences from “above” ensure SC

Cache Coherence

Cache Coherence

“Coherence seeks to make the caches of a shared-memory system as **functionally invisible** as the caches in a single-core system. Correct coherence ensures that **a programmer cannot determine whether and where a system has caches** by analyzing the results of loads and stores.”

- Mark Hill

Cache Coherence

Requirements

1. Memory operations issued by any one processor occur in the order issued by the processor
2. The value returned by a read is the value written by the last write to the location… as given by the serial order

https://gfxcourses.stanford.edu/cs149/fall22content/media/consistency/11_consistency.pdf

Cache Coherence

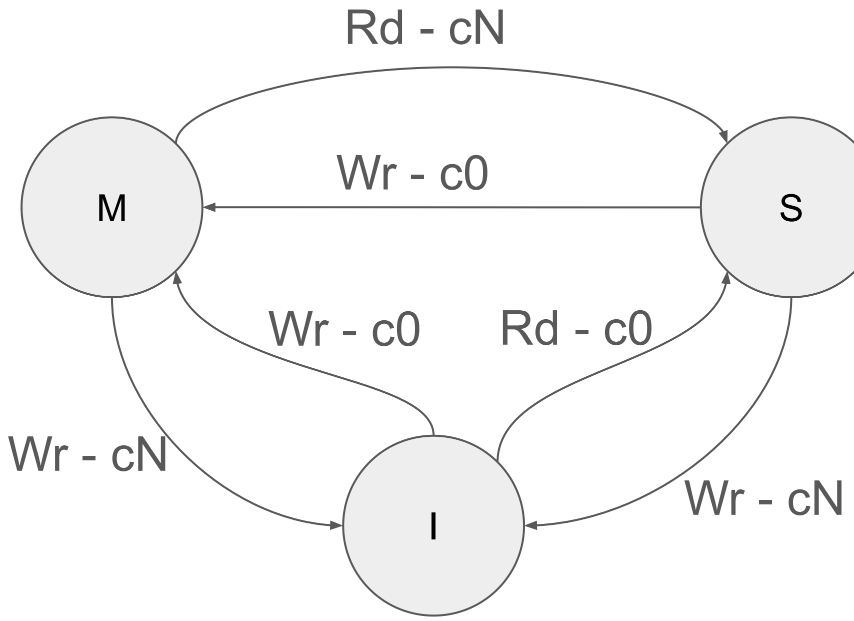
Implementation:

For any given time period (or Epoch)

1. Single Writer, Multiple Readers (SWMR) Invariant
    a. Read-write epoch: there exists only a single processor that may write to x (and can also read it)
    b. Read-Only epoch: some number of processors that may only read x
2. Data Value Invariant (Write Serialization)
    a. The value of the memory address at the start of an epoch is the same as the value of the memory location at the end of its last read-write epoch

The MSI protocol

Extend the validity state management in the cache



Not pictured: transitions to self.

Rd - cN	Read from another core
Rd - c0	Read from this core
Wr - cN	Write from another core
Wr - c0	Write from this core