Computer Science 703

Advance Computer Architecture

2008 Semester 1

Lecture Notes 3

11Mar08

Multiprocessing Issues

James Goodman

Department of Computer Science

Readings

Today: Mark Hill, “Multiprocessors Should Support Simple Memory-Consistency Models”

Tomorrow: Sweazey & Smith, “A class of compatible cache consistency protocols and their support by the IEEE Futurebus”

Multiprocessing Issues

- “Missing Update Problem”
- Memory Consistency
- Cache Coherence

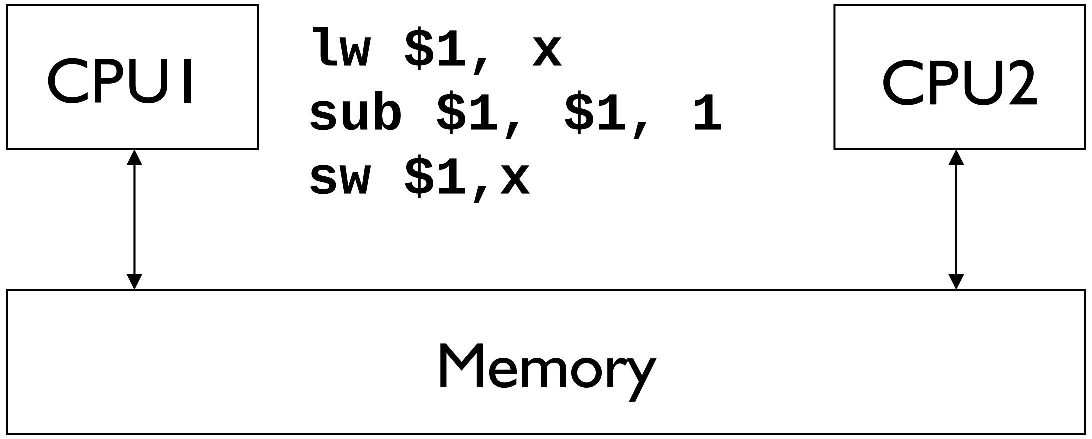
2008 YEAR
PRESENTATION
The University of Auckland | New Zealand

Multiprocessing Issues

- **“Missing Update Problem”**
- Memory Consistency
- Cache Coherence

The “Missing Update” Problem

- Assume there is a shared **`int x = 3;`**
- CPU1 executes a program fragment **`x = X - 1;`**
- CPU2 executes a program fragment **`x = X - 2;`**
- What is the final value of the shared variable **`x`**?



```
lw $1, x
sub $1, $1, 1
sw $1,x
```

```
lw $1, x
sub $1, $1, 2
sw $2,x
```

2008
YEAR
PRESENTATION
The University of Auckland | New Zealand

Possible Answer: 0

CPU 1

```
lw $1, x
sub $1, $1, 1
sw $1,x
```

CPU 2

```
lw $1, x
sub $1, $1, 2
sw $1,x
```

Result: x = 0

Possible Answer: 0

CPU 1

```
lw $1, x
sub $1, $1, 1
sw $1,x
```

CPU 2

```
lw $1, x
sub $1, $1, 2
sw $1,x
```

Result: x = 0

Possible Answer: 1

CPU 1

```
lw $1, x
sub $1, $1, 1
sw $1,x
```

CPU 2

```
lw $1, x
sub $1, $1, 2
sw $1,x
```

Result: x = 1

Is this acceptable?

2008
YEAR
PRESENTATION
The University of Auckland | New Zealand

Possible Answer: 2

CPU 1

```
lw $1, x
sub $1, $1, 1
sw $1,x
```

CPU 2

```
lw $1, x
sub $1, $1, 2
sw $1,x
```

Result: x = 2

Is this acceptable?

Expectation of Atomicity & Isolation

- In the example, we expect that the code

```
    x = x - 1
```

  will be executed *atomically* and in *isolation*

Isolation: the appearance that a sequence of operations occur at a single instant in time.

Atomicity: the requirement that the sequence of operations either occurs in its entirety or not at all.

Parallel “Correctness”

- Our programs must execute “correctly” no matter how the two sequences of instructions are interleaved
- But correctness must be defined. The example introduces a *data race*
- If only a result of zero is acceptable, the code must explicitly eliminate data races
- Data races can be eliminated by the use of *locks* (*semaphores*) and *critical sections*

Observation: This problem has nothing to do with cache consistency or coherence!

2008 YEAR
PRESENTATION
The University of Auckland | New Zealand

Programming a Multiprocessor

- Multiprocessors may simply execute independent tasks that require more computing power that is available on a single processor.
  - Particularly useful if one or more jobs is computationally intensive
  - Often a maximum of two processors can handle all the jobs
- Major challenge: divide up a single job into pieces that can be computed concurrently.
- Two general models of parallel computation
  - The *epoch* model
  - The *work queue* model

2008
YEAR
PRESENTATION
The University of Auckland | New Zealand

The Epoch Model

- The program involves similar operations on large amounts of data (large, regular data structures)
- The data is partitioned into non-overlapping parts and assigned to various threads
- A fixed amount of computation is performed independently, then coalesced through synchronization
  - All nodes run the same code, over a different range of data
  - This is an *epoch*
- This process is repeated
  - A *barrier* assures that none of the threads proceed beyond the synchronization point until all have arrived at it
- Within an epoch, *usually* no races are allowed, i.e., no variable can be written by some node and read by another.

2008
YEAR
PRESENTATION
The University of Auckland | New Zealand

Synchronization Mechanism for Epochs

The barrier: wait for all nodes to arrive here before continuing:

Initially, **`Count`** = # of threads

```
barrier() {
  Count -= 1;
  while (Count > 0)
   ;
}
```

Note: decrementing **`Count`** *on multiple nodes introduces a race condition!*

2008
YEAR
PRESENTATION
The University of Auckland | New Zealand

The Work Queue Model

- The system is initialized by identifying a set of tasks to be performed. These are placed on a queue with information identifying the task and its parameters.
- Processors remove an assignment from the queue and perform the task. In the process, they may identify new tasks to be performed and place them on the queue.
- This process continues until all the tasks have been completed.
- The challenge is to divide tasks up fine enough so that all the threads can be kept busy, but course enough so that the threads don’t spend all their time dealing with the work queue

Multiprocessing Issues

- “Missing Update Problem”
- **Memory Consistency**
- Cache Coherence

The University of Auckland | New Zealand

2008
YEAR
PRESENTATION
The University of Auckland | New Zealand

Memory Ordering

Initial state:

```
A = 0
B = 0
```

CPU 1

```
Write A = 1
Write B = 1
```

CPU 2

```
Read B = 1
Read A = 0
```

Is this possible?

Is it *acceptable*?

2008
YEAR
PRESENTATION
The University of Auckland | New Zealand

Sequential Consistency

Definition: "…the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program."

2008
YEAR
PRESENTATION
The University of Auckland | New Zealand

Memory Ordering Requirement (1)

Initial state: `A = 0`
`B = 0`

CPU 1

```
Write A = 1
Write B = 1
```

CPU 2

```
Read B = 1
Read A = 0
```

Is this permitted?

SC: No
Intel: No
Alpha: Yes

Alpha Memory Ordering Requirements

- Reads and writes may appear out of order
- A *memory barrier* assures that all previous operations have been made globally visible before any subsequent operations are made visible

Memory Ordering Requirement (1)

Initial state:

```
A = 0
B = 0
```

CPU 1

```
Write A = 1
MemBar
Write B = 1
```

CPU 2

```
Read B = 1
MemBar
Read A = 0
```

Is this permitted?

SC: No
Intel: No
Alpha: *No*

Memory Ordering Requirement (2)

Initial state: `A = 0`
`B = 0`

CPU 1

```
Write A = 1
Read B = 0
```

CPU 2

```
Write B = 1
Read A = 0
```

SC: No
Intel: *Yes*
Alpha: Yes

Is this permitted?

YEAR 2008
PRESENTATION
The University of Auckland | New Zealand

Yet Another Memory Model

- Release Consistency
  - Assumes that locks are used to protect shared data
  - No reads may be performed before acquiring the lock
  - All writes must be completed before releasing the lock

YEAR 2008
PRESENTATION
The University of Auckland | New Zealand

Summary of Memory Ordering

- Identical code sequences may result in different *acceptable* answers on multiprocessors with different memory models
- Compilers must account for memory model
  - Recognize potential data races
  - Insert barriers if necessary to assure correctness