Previously, we introduced multi-cores.

Today we’ll look at issues related to multi-core memory systems:

1. Threads
2. Cache coherence
Remember we introduced the MIPS memory image?
Process View

- Process = thread + code, data, and kernel context

**Thread**
- Program context:
  - Data registers
  - Condition codes
  - Stack pointer (SP)
  - Program counter (PC)

**Code, data, and kernel context**
- Shared libraries
- Run-time heap
- Read/write data
- Read-only code/data

**Kernel context:**
- VM structures
- brk pointer
- Descriptor table
Process with Two Threads

Thread 1

Program context:
- Data registers
- Condition codes
- Stack pointer (SP)
- Program counter (PC)

Stack pointer (SP)

stack

Thread 2

Program context:
- Data registers
- Condition codes
- Stack pointer (SP)
- Program counter (PC)

Stack pointer (SP)

stack

Code, data, and kernel context

- shared libraries
- run-time heap
- read/write data
- read-only code/data

Kernel context:
- VM structures
- brk pointer
- Descriptor table

©2012 Craig Zilles
Thread Execution

- **Single Core Processor**
  - Simulate concurrency by time slicing

- **Multi-Core Processor**
  - Can have true concurrency

Run 3 threads on 2 cores

4/25/18
©2012 Craig Zilles
How do threads from the same process communicate?

A) Their registers are kept in sync; when one thread writes a register, the written value is copied to the corresponding register in all cores that are running threads from the same process.

B) Threads have distinct register files, but can read and write each other register files directly.

C) Threads have distinct register files and must communicate through memory by one thread writing a memory location and later another thread reading that same memory location.

D) Threads have distinct registers and memory and must communicate through network protocols like TCP/IP.
(Hardware) Shared Memory

- The programming model for multi-core CPUs

What is it?

- Memory system with a single global physical address space
  - So processors can communicate by reading and writing same memory

- Design Goal 1: Minimize memory latency
  - Use co-location & caches

- Design Goal 2: Maximize memory bandwidth
  - Use parallelism & caches
Some example memory-system designs

(a) Shared cache

(b) Bus-based shared memory

(c) Dancehall

(d) Distributed-memory
The Cache Coherence Problem

1. LD u
2. LD u
3. ST u
4. LD u
5. LD u

- Multiple copies of a block can easily get inconsistent
  - Processor writes; I/O writes
- Processors could see different values for u after event 3
Cache Coherence

- According to Webster’s dictionary ...
  - **Cache**: a secure place of storage
  - **Coherent**: logically consistent

- **Cache Coherence**: *keep storage logically consistent*
  - Coherence requires enforcement of 2 properties

1. **Write propagation**
   - All writes eventually become visible to other processors

2. **Write serialization**
   - All processors see writes to same block in same order
Cache Coherence Invariant

- Each block of memory is in exactly one of these 3 states:

  1. Uncached: Memory has the only copy
  2. Writable: Exactly 1 cache has the block and only that processor can write to it.
  3. Read-only: Any number of caches can hold the block, and their processors can read it.

**Invariant**

noun Mathematics
a function, quantity, or property that remains unchanged when a specified transformation is applied.
Snoopy Cache Coherence Schemes

- Bus is a broadcast medium & caches know what they have

- Cache controller “snoops” all transactions on the shared bus
  - Relevant transaction if for a block it contains
  - Take action to ensure coherence
    - Invalidate or supply value

Depends on state of the block and the protocol
Maintain the invariant by tracking “state”

- Every cache block has an associated state
  - This will supplant the valid and dirty bits

- A cache controller updates the state of blocks in response to processor and snoop events and generates bus transactions

- Snoopy protocol
  - set of states
  - state-transition diagram
  - actions
MSI protocol

This is the simplest possible protocol, corresponding directly to the 3 options in our invariant

- **Invalid State**: the data in the cache is not valid. \( \text{val}_i \cap \not\emptyset = \emptyset \)

- **Shared State**: multiple caches potentially have copies of this data; they will all have it in shared state. Memory has a copy that is consistent with the cached copy. \( \text{read-only} \quad \text{val}_i \cap \not\emptyset = 1 \quad \text{dir} = \emptyset \)

- **Dirty or Modified**: only 1 cache has a copy. Memory has a copy that is (potentially) inconsistent with the cached copy. Memory needs to be updated when the data is displaced from the cache or another processor wants to read the same data. \( \text{dir} \cap \not\emptyset = 1 \quad \text{val}_i \cap \not\emptyset = 1 \)
Actions

Processor Actions:
- Load ✓
- Store ✓
- **Eviction**: processor wants to replace cache block

Bus Actions:
- **GETS**: request to get data in shared state
- **GETX**: request for data in modified state (*i.e.*, eXclusive access)
- **UPGRADE**: request for exclusive access to data owned in shared state

Cache Controller Actions:
- **Source Data**: this cache provides the data to the requesting cache
- **Writeback**: this cache updates the block in memory
The Cache Coherence Problem Solved

1. LD u

P₁

$ u : 7 S$

2. LD u

P₂

$ S$

3. ST u

P₃

$ u : 7 M$

4. LD u

GETS

5. LD u

GETS

I/O devices

a) I
b) S
c) M

a) GETS
b) GETX
c) UPGRADE

Source Data

GETS

UPGRADE

Memory

a) Source Data
b) Writeback

©2012 Craig Zilles