SMP/SMT

Daniel Potts
danielp@cse.unsw.edu.au

University of New South Wales, Sydney, Australia
and
National ICT Australia
Overview

• Today’s multi-processors
  • Architecture
  • New challenges
• Experience and issues for multi-processing in L4
Multi-processor architectures

What we see today...

- uni processor
- multi-processor
- multi-core (aka multi-processor in same package)
- multi-threaded
  - Symmetric multi-threading (SMT)
  - Interleaved multi-threading (IMT)
- combination?
Processor Configurations
Processor Configurations

- SMP: Share external cache; separate TLB and units
Processor Configurations

- SMP: Share external cache; separate TLB and units
- SMT: Share all caches, TLB, ALUs..
**Processor Configurations**

- SMP: Share external cache; separate TLB and units
- SMT: Share all caches, TLB, ALUs..
- Multi-core: share level 2 cache, separate TLB...
Processor Configurations

- SMP: Share external cache; separate TLB and units
- SMT: Share all caches, TLB, ALUs..
- Multi-core: share level 2 cache, separate TLB ...
Processor Configurations

- **SMP:** Share external cache; separate TLB and units
- **SMT:** Share all caches, TLB, ALUs..
- **Multi-core:** share level 2 cache, separate TLB ...
Processor Configurations

- **SMP**: Share external cache; separate TLB and units
- **SMT**: Share all caches, TLB, ALUs...
- **Multi-core**: share level 2 cache, separate TLB ...
- **Memory (cache) latency** may vary between processing units
Processor Configurations

- SMP: Share external cache; separate TLB and units
- SMT: Share all caches, TLB, ALUs..
- Multi-core: share level 2 cache, separate TLB ...
- Memory (cache) latency may vary between processing units
Processor Configurations

- SMP: Share external cache; separate TLB and units
- SMT: Share all caches, TLB, ALUs...
- Multi-core: share level 2 cache, separate TLB ...
- Memory (cache) latency may vary between processing units
Multi-core SMP + SMT: OpenSparc T1

- 8 cores
- 4 threads per core
Processor Topology Considerations

What problems do these MPs create for us?
Generally, two ways of interacting:

- memory — shared data accesses
  - latency ?
  - coherency ?

- interrupts
  - latency ?

Local vs remote operations?
Memory: Types of multi-processors (MPs)

Uniform Memory Access (UMA)
Access to all memory occurs at the same speed for all processors.

Non-Uniform Memory Access (NUMA)
Access to some parts of memory is faster for some processors than other parts of memory. We focus on cache coherent NUMA.
NUMA

Topology of inter-connect often leads to different latency and bandwidth between sets of processors.


Cache Coherency

What happens if one CPU writes to address 0x1234, which gets stored in its cache, and another CPU reads from the same address?
What happens if one CPU writes to address 0x1234, which gets stored in its cache, and another CPU reads from the same address?

- Visibility of writes
SMP Coherency Protocol: MESI [1]

Each cache line is in one of four states (MESI):

- **Modified (M)**
  - The line is valid in the cache and in only this cache.
  - The line is modified with respect to system memory; that is, the modified data in the line has not been written back to memory.

- **Exclusive (E)**
  - The addressed line is in this cache only.
  - The data in this line is consistent with system memory.
SMP Coherency Protocol: MESI [2]

- **Shared (S)**
  - The addressed line is valid in the cache and in at least one other cache.
  - A shared line is always consistent with system memory. That is, the shared state is shared-unmodified; there is no shared-modified state.

- **Invalid (I)**
  - This state indicates that the addressed line is not resident in the cache and/or any data contained is considered not useful.
SMP Coherency Protocol: MESI [3]

RH = Read hit
RMS = Read miss, shared
RME = Read miss, exclusive
WH = Write hit
WM = Write miss
SHI = Snoop hit on inv
SHR = Snoop hit on read
LRU = replacement alg
Data structures we need to consider:

- scheduling queue[s]
- thread lists
- page tables (and TLB)

Atomic operations on thread state:

- IPC
- delete
- running (scheduled)?
Kernel Locking

• Several CPUs can be executing kernel code concurrently.
• Need mutual exclusion on shared kernel data.
• Issues:
  • Lock implementation
  • Granularity of locking
Mutual Exclusion Techniques

- Disabling interrupts: CLI — STI
- Spin locks
- Lock objects

Kernel may use a variety of these techniques.
Mutual Exclusion Techniques

- Disabling interrupts: CLI — STI
  - Not suitable for MP
- Spin locks
  - Busy-waiting wastes cycles?
- Lock objects
  - Flag (lock) indicates object is locked.
  - Manipulating lock requires mutual exclusion.

Kernel may use a variety of these techniques.
Hardware Provided Locking Primitives

- test_and_set
- compare_and_swap
- atomic_increment
- load_linked + store_conditional
L4 Design Issues

• One scheduling queue per processing unit?
• IPC rendezvous and optimisations
• How do we share shared data structures?

Does one approach suit all?
Scheduler per processing unit:
- Keeps data access local
- Migration performed explicitly
- No load balancing (except at user-level or...)

Global scheduler (shared queue):
- Shared queue between processing units
- Mutual exclusion required
- Load balancing natural

Hybrids: Scheduling domains
- Use both approaches to suit processor topology
Scheduler: SMP and ccNUMA
Scheduler: SMP and ccNUMA
Scheduler: SMP and ccNUMA
Scheduler: SMP and ccNUMA

- Inter-processor interrupt (IPI) in 4000 cycles
- Memory latency >100 cycles

Preference to keep data local.

Solution:
Use RPC via IPI to perform operation.
Scheduler: SMT

- IPI <100 cycles
- memory latency <50 cycles

Shared data structures may be viable

Solution:
Use locking or lock-free data structures
**L4 IPC Path**

**UP**

- **IPC with send**
  - Dest == waiting
    - Yes: Transfer message
    - No: Block polling
  - Dest == waiting
    - Yes: Transfer message
    - No: Block polling

- **IPC without send**
  - Receive?
    - No: return running
    - Yes: 
      - Src == Polling
        - No: Block waiting
        - Yes: Activate src
          - return running
**L4 IPC Path**

- **SMT**
  - Needs to be atomic
  - Before continuing, Dest = locked

- **IPC with send**
  - Dest == waiting
    - Yes: Transfer message
    - No: Block polling

- **IPC without send**
  - Receive?
    - Yes: src == Polling
      - Yes: Activate src
      - No: Block waiting
    - No: return running

- return running
L4 IPC Path

SMT

Needs to be atomic
Before continuing, Dest = locked

IPC with send

Dest == waiting

No

Block polling

Yes

Transfer message

Dest = Running

IPC without send

Receive?

No

return running

Yes

Dest = Running

Src == Polling

No

Block waiting

Yes

Activate src

return running
L4 IPC Path

SMT Optimisation

IPCs with send

1. Dest == waiting
   - Yes: Transfer message
   - No: Block polling

2. Dest == Running
   - Return running

IPCs without send

1. Receive?
   - Yes: Src == Polling
   - No: Block waiting

2. Activate src
   - Return running
**L4 IPC Path**

UP

- **IPC with send**
  - Dest == waiting
    - Yes: Transfer message
    - No: Block polling
  - Dest = Running

- **IPC without send**
  - Receive?
    - No: return running
    - Yes
      - **Src == Polling**
        - No: Block waiting
        - Yes: Activate src
          - return running
**L4 IPC Path**

- **SMP**
  - Is local?
  - RPC(xcpu_send)
  - RPC(xcpu_send_done)

- **IPC with send**
  - Dest == waiting
  - Transfer message
  - Dest = Running

- **IPC without send**
  - Receive?
  - Src == Polling
  - Activate src

- **Block polling**
  - Return running

- **Block waiting**
  - Return running
**IPC Summary**

- SMP/UP + SMT work together
- Some overlap of SMT run-time possible
- Atomic operations on SMT path required
What do priorities mean on an SMP system?

- No longer guarantee lower priorities won’t run
- Priority inversion problems
Other SMP/SMT considerations

- Interrupt routing
- Cost of atomic instructions and code sequences
- Power management
Conclusion / Observation

- No perfect locking mechanism for all situations
- No single approach to data structures

Interesting research and development topic!