New Architectural Technologies for Shared-Memory Systems

Josep Torrellas

University of Illinois
http://iacoma.cs.uiuc.edu

High-Speed Computing Conference, Salishan, April 2004
What to Expect 10 Years from Now

- Major emphasis on:
  - Programmability & ease of use
  - Cost-effective fault tolerance
Emerging Architectural Technologies

- Speculative threading
- Transparent checkpointing
- Intelligence in the memory subsystem

Warning: I will not mention the word MPI once
Emerging Architectural Technologies

- Speculative threading
- Transparent checkpointing
- Intelligence in the memory subsystem
Multithreading State of the Art: Helper Threads

- Thread 0
  - Thread 1
  - Thread 2

Multithreaded Core

- Helper Thread 1
- Helper Thread 2

- Frequently Missing Load
- Hard to Predict Branch
From Prefetching to Speculative Multithreading
Idea in Speculative Multithreading (SM)

- Current processors: speculate within the pipeline
- SM: speculate on code long enough that state overflows into cache hierarchy

Entering the speculative section:
- Hardware checkpoints the register state

Executing the speculative section:
- Buffer all memory updates in the cache -- cannot update mem
- Mark cache lines read and written
- Monitor for errors or violations

If error or violation occurs:
- Hardware invalidates updated cache lines & restores regs

Else: Successful end of speculation:
- Reset marks & allow eviction of updated cache lines
SM to Ease Parallel Programming

Speculative Synchronization (Atomic Sections)

Thread 0
Enter Atomic
Exit Atomic

Thread 1
Enter Atomic
Exit Atomic

Thread 0
Thread 1
Thread 2

Barrier
Barrier
Barrier

OK to write coarse atomic sections or put additional barriers
SM to Help Debugging

On the fly undo/redo

Can be used to debug data races in multithreaded codes
SM to Help Debugging

Watch memory location and trigger monitoring function

Watch(address, monitor_fn1)

* \*p = ...

Monitor_fn1 (Addr) {
    return(addr != 0)
}

SM to Help Debugging

Automatically fix a program

Assert \((\text{ptr} < \text{maxptr} \&\& \text{ptr} > \text{minptr})\)

fails
Implications for Algorithms/Applications

- Easier to write parallel programs
  - coarse synchronization OK

- Easier to debug programs… in production runs
  - fine grain memory protection (Watch)
  - checker thread performs distributed consistency checks on data structures
  - support deterministic replay of code sections
Emerging Architectural Technologies

- Speculative threading
- Transparent checkpointing
- Intelligence in the memory subsystem
Checkpointing and Rollback Recovery

- Faults (especially transient) will remain a challenge in future

- Currently:
  - Apps often manage their checkpointing
  - Apps often stop for a long time to write their checkpoint to disk

- Goal:
  - Checkpointing transparent to app
  - Low cost:
    - No HW changes to processors/caches/memories/disks
    - No changes to OS
  - Effective: High availability with low overhead
Hardware-Aided Checkpointing *

- Global interrupt every 100 ms creates checkpoint
  - CPUs write back registers and dirty cache lines to memory
  - Main memory is the checkpoint state

* Experiments performed in a simulated 16-processor machine
Hardware-Aided Checkpointing

Between checkpoints:
- When machine is about to modify line in memory for 1st time: mem controller saves old value of line in memory log

To ensure main memory “is safe”
- Mem controllers protect main memory by keeping distributed parity like RAID-5
- Can tolerate loss of a node
Recover the Loss of a Node Under 1 Second

Execute

Detection

Repair Log

Repair Data

P0

P1

P2

P3

Checkpoint

Bzzzt!

100ms

80ms

50ms

~100ms

~490ms

~20s

Unavailable (~840ms)

Degraded (~20s)

Self-Check

Reroute

Rollback

Time

Detection

Repair Log

Repair Data
“Recovering” I/O too

- Add Pseudo Device Driver (PDD) between kernel and device drivers
- I/O output requests redirected to PDD, which buffers it
- After next checkpoint, the I/O requests passed to DD and committed in background
Implications for Algo/Apps

- No need to add checkpointing code to your app

- Very fast, transparent recovery from many faults:
  - transient faults
  - permanent faults: up to the loss of a node
Emerging Architectural Technologies

- Speculative threading
- Transparent checkpointing
- Intelligence in the memory subsystem
Intelligence in the Memory System

- Simple, narrow-issue engines associated with:
  - memory controllers
  - L3 caches

- Execute “intelligent memory operations”:
  - software threads running “in memory”
  - hardware operations
Intelligent Memory Operations

**Software Threads**
- Data preparation, page table
- Pretouch
- Reduction
- Synchronization
- Scatter/gather
- Bit operations
- Execute memory-intensive code sections

**Hardware Operations**
- Prefetching
- Logging
- Checkpointing
- Cache coherence management
- Memory RAIDing
Memory Side Prefetching

1. Memory Controller
2. Mem Proc
3. L1
4. L2
5. Interconnect

Implications for Algo/Apps

* Use “in memory” software threads
  – How to manage heterogeneous threads (proc and memory)
  – Map what parts of the program where?
  – How to synchronize processor and memory threads?
  – How to maintain data coherence?
Final Thoughts

- Ease of programming, in the presence of:
  - software bugs
  - transient faults

- Use transistor surplus for debugging support

- Continuous optimization in the background (intelligence in the mem system)
New Architectural Technologies for Shared-Memory Systems

Josep Torrellas

University of Illinois
http://iacoma.cs.uiuc.edu

High-Speed Computing Conference, Salishan, April 2004