

Simulation Techniques for NUMA Architectures: Performance Evaluation via Simulation and Hardware Counters

Peter Strazdins,
(with Alistair Rendell, Bill Clarke and Andrew Over;
also Rui Yang and Joseph Anthony)

The CC-NUMA Project,
Department of Computer Science,
Australian National University,

<http://cs.anu.edu.au/~Peter.Strazdins/seminars/SimTechNUMA>

29 September 2004

1 Talk Outline

- CC-NUMA project overview
- performance analysis of CC applications on NUMA architectures
- the Sparc-Sulima UltraSPARC SMP simulator
 - approach
 - object-oriented structure
 - OS-emulation modes
 - LWP system call emulation
 - implementing the pipelined backplane
 - the UIII CPU cycle-counter module
 - status and performance
- future work in performance evaluation

2 The CC-NUMA Project Overview

- project details:
 - Australian Research Council Linkage Grant, funded from 04/2003 for 3+ years
 - industry partners:
 - Sun Labs (Menlo Park); CI Dr Ilya Sharapov
 - Gaussian Inc (New Haven); CIs Dr Michael Frisch & Dr Gary Trucks)
- goal: isolate linear scaling kernels in Gaussian
 - primarily user-level, and (SMP) memory effects of most interest
 - parallelize with special emphasis on data placement
 - thread affinity issues also important
 - use `libccpc` to instrument kernels; obtain useful statistics
 - detailed simulation results for further information
 - UltraSPARC III architectures and beyond of chief interest (primarily in the memory system)
- other memory-intensive scientific applications also of interest

3 The Need for Simulator Development for CC-NUMA

- limitations of using instrumented benchmarks:
 - some useful information may not be available
 - eg. E-cache misses by type (conflict, invalidation), bus contentions, success of prefetches
 - results on #'s of CPU limited to available hardware
 - larger systems may have qualitatively different NUMA effects! (eg. extra level of interconnect, contention)
 - results are also limited to actual cache / interconnect configurations
 - interesting to explore variants, even completely different models

In principle, a simulator can supply all such information ...

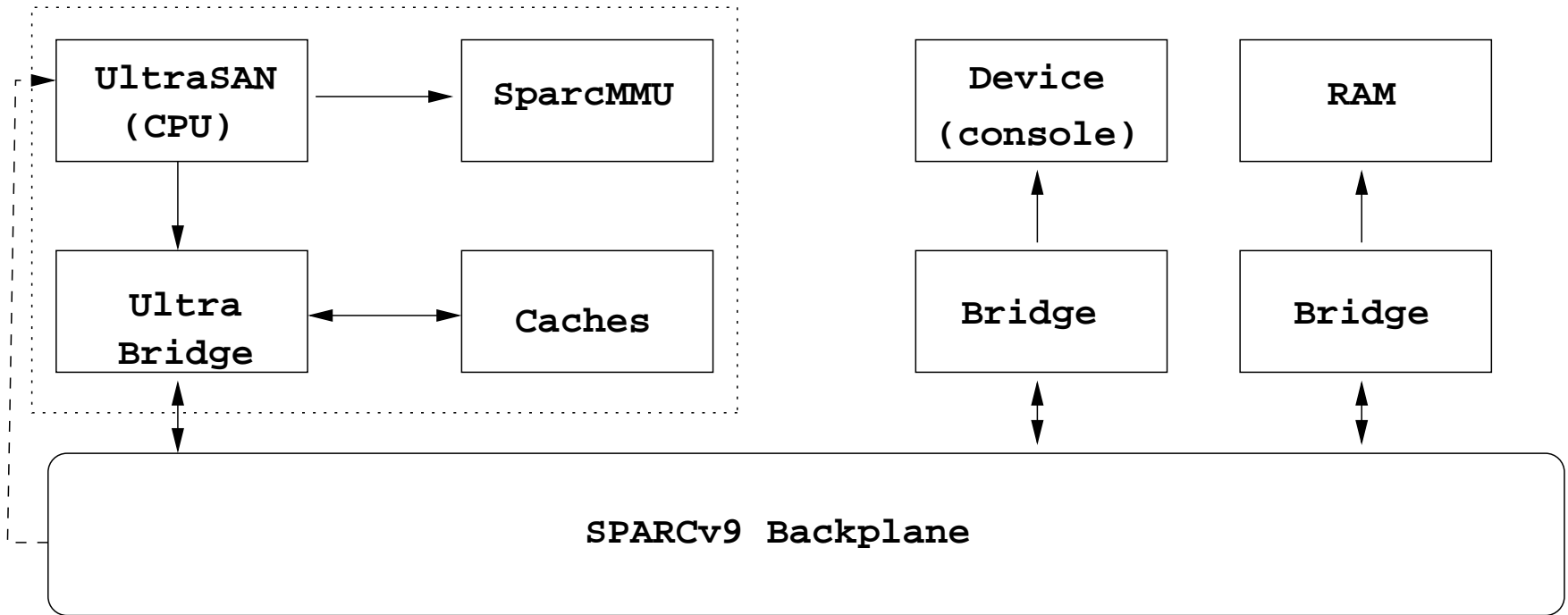
- the Sparc-Sulima Project
 - began in 2000 under the ANU-Fujitsu CAP Program, now part of the CC-NUMA project

aims to meet these needs

4 The Sparc-Sulima Approach

- UltraSPARC is an an implementation of the 64-bit SPARC V9 architecture
- model all aspects of (US-III) architecture explicitly using a heavily OO approach
 - eg. D-cache data as well as tags explicitly modelled;
E-cache module only accessed upon (simulated) miss
- CPU modelled by a fetch-execute-decode core (UltraSAN)
 - generally deemed suitable for analysing memory performance
 - nominally 1 cycle per instr'n, with added latency from memory system
- the CPU is connected to the memory system (caches and backplane) via a 'bridge'
 - can have a plain (fixed-latency) or fully pipelined Fireplane-style backplane
 - memory system is (will be) fully modular
 - plug into other simulators and/or trace generators
 - can be replaced by a 'vanilla' (cacheless, CC RAM) memory

5 Structure of Sparc Sulima



- the active components are the CPUs and the Backplane

6 Emulation Modes in Sparc-Sulima

- but isn't it a complete machine simulator ??? Has emulation modes:
 - 'boot from `main()`': executable uses `stdim/out` only; can optionally install a (tiny) nucleus:
 - enable MMUs, trap handlers; $PA = VA$
 - runs in privileged mode! can run SPMD-style SMP programs
 - User-sim: emulate at system call level, as for RSIM (but no SMP)
 - boot from pseudo-PROM; nucleus is non-trivial; $PA = VA + \text{offset}$
 - limited `v8plusa` executable must be specially (statically) linked
 - CMS issue: memory-accessing calls must be restartable! (TLB miss)
 - Solemn: emulate Solaris at system trap level
 - implements `mmap`, dynamic linking, a subset of `_lwp` traps; $PA = f(VA)$
 - can simulate 32/64-bit executables unmodified; sufficient for CC-NUMA

7 LWP Implementation under Solemn

- Solemn emulation mode runs threaded programs as follows
 - program starts running on main CPU
 - other simulated CPUs are in an 'idle' state, until woken up by an `_lwp_create()` call
 - 'idle' state also used to simulate `_lwp_wait()`
- data structures are used to maintain thread and CPU states / attributes
- current limitation: 1 CPU per (active) thread
- most calls (except mutex lock/unlock) are emulated, rather than simulated
- again, only a subset of calls are emulated (only those used by threaded/OMP scientific applications)
 - difficult to find the exact semantics of some calls

8 Implementation Issues for the Pipelined Backplane

- uses conservative parallel discrete event simulation
 - backplane and CPUs are each allowed to simulate for 'window' (30 CPU cycles for the current FirePlane)
 - no new event can affect another component until end of window
 - could in principle be parallelized
- currently models address repeaters (2 levels). contention on memory banks and cache snooping & pipelined transactions (TODO: bus contention)
- CPU simulation must be modified to support this as follows:
 - has a **priority queue** to handle externally initiated E-cache events (snoop requests, completion of invalidations and writebacks)
 - internally generated E-cache events (misses):
 1. returns a minimum possible latency
 2. generates an internal exception (retry instr'n later)
 3. the above is repeated until the operation completes
- inside caches, completion of misses via call-back by the bridge
- ideally, would like to plug this into to other simulators (SimICS?)

9 The CycleCounter Module for Improved CPU Accuracy

- desire to balance accuracy of the pipelined Backplane with amore accurate CPU *without* major performance impact
- an optionally installable CycleCounter module can be called upon every instruction cycle (models the UIII Cu pipelines)
 - takes a decoded instruction structure as input
 - returns latency that that instruction contributed
 - non-zero only for the last instruction of the group, or if there was a delay to a register dependency
 - the CPU clock value then corresponds to the Execute Stage (this value is seen by the memory system)
- preliminary version adds about 50% to simulator time
 - most expensive is maintaining register interlocks
- very accurate on some benchmarks, such as matrix-vector multiply
- could be possibly plugged in to other simulators, e.g. SimICS
 - however, would need Sparc-Sulima decoder (and its 'cache')
- difficult to use a modular approach for out-of-order execution . . .

10 Status and Performance

- existing optimizations: use **caching** of expensive calculations
- currently slowdowns of between $250\text{--}500\times$ are observed
 - highly memory-intensive applications (but with good cache performance) tend to be slower
- multiple CPU simulations are slower (e.g. 4 CPUs about $500\text{--}1000\times$ slowdowns)
 - prototyping a threaded simulator (1 thread per CPU) (Mark Thorne, Honours Student)
- runs pthread / OMP versions of various scientific style applications, including:
 - NAS Parallel Benchmarks (have tried S class only so far!)
 - SCF Gaussian-like kernels (C++/OMP)
 - currently, 'slow' algorithms from McMurchie Davidson PRISM used
 - revealed yet another obscure bug, suspected to be in the caches

...

11 Future Work

- simulator development:
 - complete threaded simulator; ||ize the pipelined backplane
 - implement prefetch! (maybe UIII MMU)
 - infrastructure: completion of work on annotations and checkpointing
 - develop event gathering infrastructure
- performance evaluation for CC-NUMA:
 - analyse benchmarks of interest with `libcpc`
 - compare with simulator results for calibration
 - analyse NUMA-aware Gaussian kernels (to be developed)
 - simulate alternate (NUMA) memory system designs
- performance evaluation of broader interest:
 - other memory-intensive scientific benchmarks
 - investigate queueing models & comparing with simulation results
 - evaluation of futuristic memory models
 - . . .