

# Simulation Techniques for NUMA Architectures

Peter Strazdins,  
(with Alistair Rendell, Bill Clarke and Andrew Over)

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

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

21 August 2003

# 1 Talk Outline

- performance analysis of CC applications on NUMA architectures
  - why simulation is needed and fundamental challenges
- the Sparc-Sulima UltraSPARC SMP simulator
  - history & approach (as a complete machine simulator)
  - object-oriented structure
  - OS-emulation modes
  - implementation: instruction decoding & evaluation, memory system
  - performance & portability
  - control and debugging infrastructure (via a scripting interface)
- future challenges:
  - greater accuracy: 'nearly cycle-accurate' CPU mode & modelling advanced memory systems; threaded implementation
  - booting a full operating system
- conclusions

## 2 Performance Analysis of CC on NUMA

- 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 `libcpc` to instrument kernels; obtain useful statistics such as
- 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
    - counts of such events do not necessarily translate to cycles lost
  - 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 . . .

### 3 Fundamental Challenges for cc-NUMA Simulators

- Complete Machine: must simulate the memory system in detail:
  - multiple VM address spaces, translations via MMU, cache coherency and emulate/simulate (less-well documented!) devices:
    - PROM, DMA, disk, network, . . .yet must be reasonably fast as well as faithful to boot a full OS
- ideally to simulate arbitrary (preferably) un-modified executables (+ OS boot images)
  - eg. dynamically loaded libraries, self-modifying code, . . .
  - debugging a simulator running these is very hard!
- for SMP, need realistic memory interleavings; approaches:
  1. a single process simulates  $s$  cycles of each CPU ('round-robin')
    - $s$  must be small (realism); switching overhead problematic
  2. have a separate simulation thread per CPU
    - faster on SMP host! but must reduce synchronizations. . .

## 4 The Sparc-Sulima Project

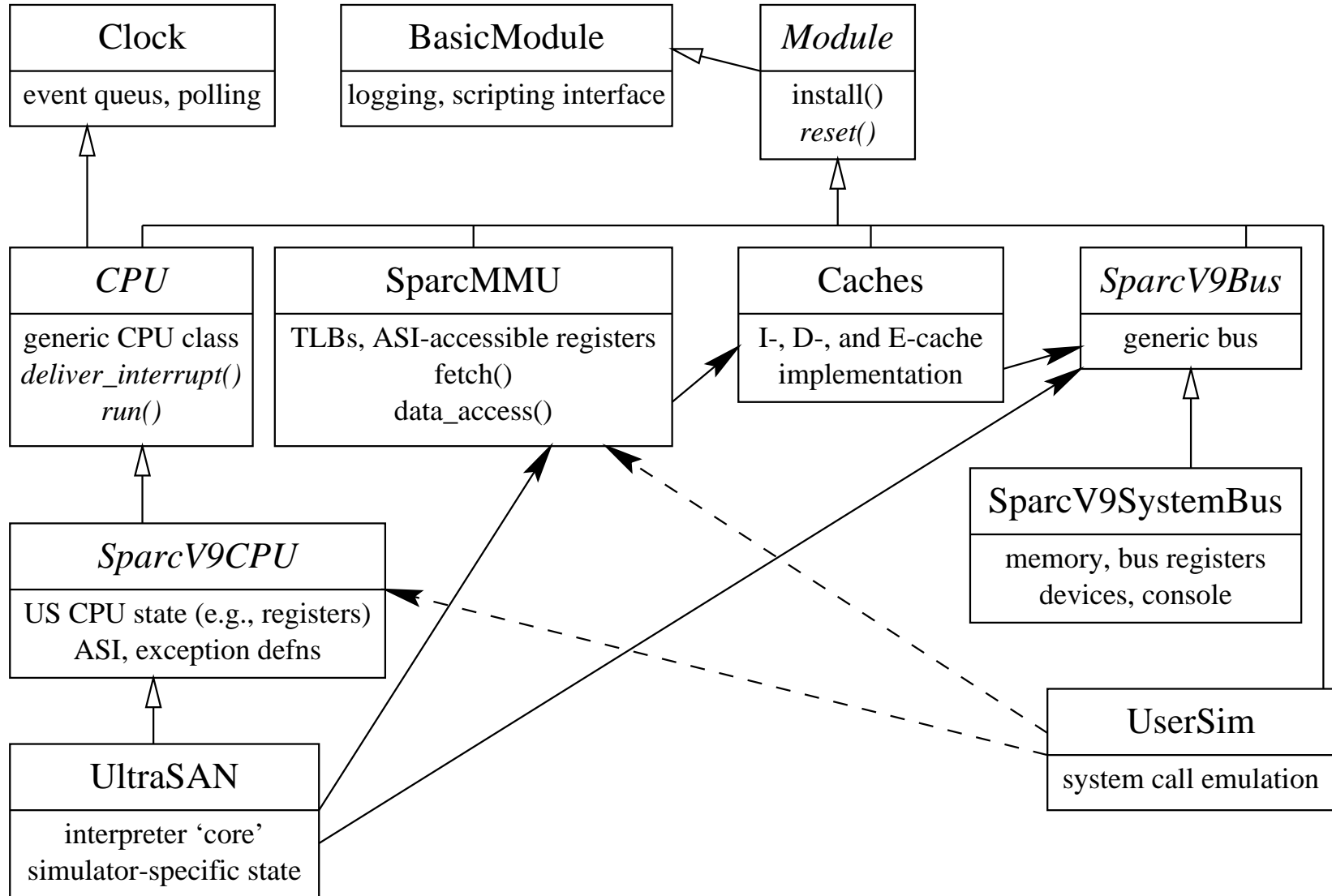
- the Sulima project from the DiSy Group at UNSW (1998-2000)
  - (prototype) 64-bit MIPS simulator, grew out of a frustrated SimOS 'port'
  - heavily OO approach designed for portability and modularity
- a brief history of the Sparc-Sulima project:
  - 10/99: ANU-Fujitsu CAP Program (Phase III) began; required a tool to investigate kernel-level memory behaviour on SPARC V9 SMPs
  - 01/00: initial investigation: an execution-driven CMS needed
  - 08/00: SimOS and other alts. rejected; design of Sparc-Sulima began
  - 10/01: Ultra I/II functionality; boot (limited) executables; fully optimized; achieved goal of  $200\times$  slowdown
  - 12/01: first code release (under GPL / BSD)
  - 10/02: 2nd release (SWIG/Python scripting interface, annotations & tracing infrastructure, SMP)
  - 01/03: development continued under CC-NUMA Project
  - 08/03: Solemn emulation mode (dynamically linked executables)

## 5 The Sparc-Sulima Approach

- UltraSPARC is an an implementation of the 64-bit SPARC V9 architecture
  - RISC, but not so *Reduced* any more!
    - eg.  $\approx$  320 instruction types; complex operations; sliding GPR windows, ‘alternate’ global register sets SPARC V8 (32-bit) backward compatability, 5 trap levels, . . .

This has an impact on a (robust and fast) simulator design!
- model all aspects of (US-I/II) 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
  - however, bus coherency transactions not explicitly modelled
- CPU modelled by a fetch-execute-decode core (UltraSAN)
  - generally deemed suitable for analysing memory performance (speed-accuracy tradeoff)
  - nominally 1 cycle per instruction, with (possible) added latency from memory system

# 6 Structure of Sparc Sulima



## 7 Emulation Modes in Sparc-Sulima

- but isn't it a complete machine simulator ??? Has emulation modes:
  - 'boot from `main( )`': executable uses `stdin/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;  $PA = f(VA)$
    - can simulate most 32/64-bit executables unmodified; (will be) sufficient for CC-NUMA
- there is a continuum between the user level and CMS approaches
  - to extend Solemn for `_lwp` traps: expand nucleus to simulate as much as possible: captures memory overheads (of a minimal OS)

## 8 Instruction Decoding

- need to efficiently decode 32-bit instruction → opcode & operands
- usual method is via jump tables or functions
  - × prone to error, hard to debug, hard for a complex ISA
- we used SLED (the Specification Language for Encoding and Decoding) to specify the syntax of the UltraSPARC ISA
  - ✓ njmctk (New Jersey Machine-Code Toolkit) can automatically generate decoders and encoders from a SLED specification
    - can also verify the specification with aid of an external assembler
- the Sparc-Sulima main instruction decoder distinguishes 252 instructions
  - is  $\approx$  6000 lines of C code
- an 8-word instruction buffer is also used to reduce overhead of simulated fetches (as on real machine)

## 9 Efficient and Robust Instruction Evaluation

- decoder produces an opcode & operands structure
- 326 instruction evaluation functions in a function table (indexed by opcode)
- the decoding is cached in the simulated I-Cache to save repeated decoding (like the UltraSPARC's pre-decode fields)
- general purpose register file index calculations also need to be cached!
- 43 integer instructions:  $rs1 \odot rs2\text{-or-imm} \rightarrow rd$  (with side-effects)
  - some of these are quite difficult to (reliably) specify in C++ (esp. condition-code setting)
- also do floating-point and (complex!) graphics instructions
- potential optimisation for simulating SPARC V9 on SPARC V9: use the exact same instruction!
- have implemented in g++ inline assembler and Forte CC external inline templates (implementation is rather esoteric)

## 10 Memory System Modelling

- is quite complex: contributes to 1/4 of SPARC-specific code
- interface is basically load from and store to an address
- ASIs (address space identifiers) change the behaviour of the load/store
  - for UltraSPARC, many ASIs are privileged and are used for manipulation of the memory system
- normal loads and stores need to be translated from the virtual address to the physical address via a lookup in the TLB (translation lookaside buffer)
  - we implement this with both a translation cache and a naive direct implementation as fallback
  - translation cache lookup must fail if an exception is possible
- caches (data, instruction and external) are implemented directly, each with their own copies of the data C++ templates express common parts)
  - MOESI copyback-invalidate coherency protocol implemented without modelling individual bus transactions
  - round-robin CPU scheduling is used
- the bus is currently fairly basic but has RAM and slots for some simple devices (like a PROM or a console)

## 11 Performance

- performance: main technique was to ‘cache’ expensive calculations repeatedly carried out by simulator:
  - dc: recent instruction decodings
  - gprs: register window-related calculations
  - mmu: virtual → physical addresses, incl. TLB lookup
- on a US-I host:

bzip2test

compiler	exe	CC, 64-bit						CC 32-bit	g++ 2.95.2	g++ 3.0.2
optimisation	-	none	gprs	dc	mmu	all	all+ipo	all	all	all
time (s)	2.12	1934	1671	1269	894	626	590	744	895	911
slowdown	1	912	788	599	422	296	278	351	422	430

- slowdowns of  $\approx 200$  achieved for the Linpack benchmark
- code bloat problems:
  - causes I-cache misses on the host
  - above figures from 10/01; may be a little slower now ...

## 12 Control & Debugging Infrastructure

- objectives:
  - to conveniently access (selected) Sparc-Sulima methods and members from a scripting interface
    - eg. `reset()` & `run()`, register files, and sim. control variables
  - to be able to do so in a workload-specific way
  - to determine progress of simulation (& locate where things go wrong. . .)
- solutions:
  - for debugging, use tracing of exceptions, calls and instructions  
(the latter two are highly customizable and reasonably fast loadable modules)
  - also recently added `gdb` support
  - add annotation classes, with **hooks** from UltraSAN run loop, MMU etc
    - main effect: manipulate simulator control variables
      - controlling start/stop, tracing, event collection
    - use external state of corresponding module, plus own internal state
    - members must also be accessible from the scripting interface
      - can change these dynamically, but not annotation function code

## 13 Scripting Interface

- chose Python (has 64-bit/object/array support) for scripting
  - the original Sulima Tcl interface not satisfactory
  - however, need a convenient way of expressing the C++/Python interface to a large number of (large) classes
- solution: SWIG (Simplified Wrapper and Interface Generator)  
a tool to connect C/C++ programs to Perl, Tcl, Python, etc
  - create `.swg` files shadowing the `.h` files
    - semi-automated (eg. must remove any function bodies)
  - for Python version, top-level `Makefile` creates `SparcSulimac-module.so`  
and `SparcSulima.py` (contains all Python-C++ linkages)

- **eg.** `runsim-swig.sh MatFact.sim -w 64 -C 100`
- `runsim-swig.sh` **contains:**

```
...
import SparcSulima
sim = SparcSulima
bus = sim.SPARCV9SystemBus("bus")
cpu = sim.UltraSAN("cpu", 200L)
cpu.bus = bus

user = sim.UserSim("user")           # load user-sim module
user.exe_filename = "$1"
user.set_args(['printf "%s",', "$@";printf "\n"])
prom = sim.ROM("prom")              # load corresponding prom

cpu.symtab = sim.SymbolTablePtr.create(user.exe_filename)
a = sim.UltraAnnoteState();         # use annotations to break at dgemv
a.pc_watch = symtab.ELF_StringLookup("dgemv");

itracer_impl = sim.BinaryITracer.create()
cpu.itracer = sim.ITracer(itracer_impl)

...
sim.reset()                         # reset all modules
sim.run()                            # go!
```

## 14 Portability Issues

- useful to be able to run on non Sparc/Solaris hosts
  - issues include variations in host word-size & endianness; also 32- or 64-bit binaries
  - also non-standard location of tools; differing makefile conventions
  - particular challenges for Solemn: will need to add a 'change root'
- solutions:
  - have (slower, less precise) software implementations of most instructions
  - use `autconf` (with `m4`) to create a `configure` script
- currently can run on Power PC; next stage on i386

## 15 Challenges: Improving NUMA Simulation Accuracy/Speed

- nearly cycle-accurate CPU mode:
  - more accurately model latency of instructions taking into account groupings, stalls due to dependencies
  - thus need not model pipeline explicitly; only modest overhead
  - but need also to model memory system more accurately!
- simulating NUMA interconnects presents many challenges:
  - Sun Fireplane is a complex & highly configurable
  - at a first approximation, latencies can be determined without fully implementing its state machine
  - contention effects problematic to model with round-robin simulation: bus/interconnect needs to remember events at same simulated time
- threaded implementation: improving performance on SMP hosts
  - must minimize synchronizations; (alt. approach use optimistic transactions with rollback)
  - especially challenging for accurate simulation modes
  - still useful to get to 'point of interest' quickly

## 16 Challenges: Full OS Boot

- initial attempts to boot Sparc Linux:
  - emulated calls to PROM (via `p1275cmd( )`), using image of an US-I PROM device tree
  - was tested and debugged on a real US-I!
  - however, OS boot on simulator failed early in boot sequence (PROM trap handlers were needed to translate address of unknown purpose)
- while not essential for CC-NUMA, has the following advantages:
  - accurate capturing of thread overheads, esp. many threads per CPU
  - can take into account of actual OS' scheduling, page coloring and memory placement policies
  - highly useful for other (commercial) applications
- current status (with aim to also boot Solaris):
  - a bit further down the boot sequence (Linux unhappy with given device tree)
  - have relied on 'reverse-engineering'; need proper documentation!
  - still must model (minimally) console and disk devices

## 17 Conclusions and Future Work

- Sulima's OO approach is appropriate, but is no silver bullet!
  - an inherently complex task; **the devil is in the details!**
  - performance acceptable but at expense of greater complexity
- debugging can be very hard; use traces but too low-level and voluminous
  - large number and range of (C / assembler) test programs
  - some defensive programming methods used (eg. self-checking trans. cache)
  - an open problem to develop better techniques
- future work:
  - UltraSPARC III version (major changes to memory system)
  - adding  $_1wp$  to Solemn, possibly to extent of many threads per CPU
  - checkpointing and event collection infrastrucutre
  - previously mentioned challeges
    - need vendor support for a full OS boot!
    - undoubtedly will turn out challenging that we realize ...
  - wish to find out more about the experiences of others in the field!

## 18 Acknowledgements

- Fujitsu Labs Pty Ltd (sponsor 2000–2002)
- Australian Research Council / Sun Microsystems / Gaussian Inc (sponsors for CC-NUMA Project)
- Markus Watts (1st port of Sparc-Sulima)
- Adam Czezowski and other contributors to Sparc-SulimaSparc-Sulima