

Solemn: Solaris Emulation Mode for Sparc Sulima

Bill Clarke,
(with Peter Strazdins, Andrew Over, Adam Czezowski)

Department of Computer Science,
Australian National University,
and
The CC-NUMA Project (ANU/Sun/Gaussian)

<http://cs.anu.edu.au/~Bill.Clarke/seminars/Solemn>

19 April 2004



THE AUSTRALIAN NATIONAL UNIVERSITY

1 Talk Outline

- Glossary
- Some simulators: RSIM, Shade, SimICS, SimOS
- Sparc-Sulima: complete machine simulator for SPARC V9
- Why System Call Emulation?
- How traps and system calls work
- So what is involved?
- Solemn: Solaris System Call Emulation
 - Structure of Solemn
 - Traps and system calls
 - System call emulation
 - Reentrant system call handlers
 - Memory management
- Current status and benchmarks, bugs and todo

2 Glossary

- Complete machine simulator: e.g., SimOS, SimICS
- User-level simulator: e.g., RSIM, Shade
- Execution-driven simulation (as opposed to trace-driven)
- SPARC V8: 32-bit RISC architecture
- SPARC V9: 64-bit (backwards compatible to V8)
- UltraSPARC I, II, III, IV: Sun's implementation of SPARC V9
- Solaris: Sun's operating system, UNIX
 - Process: a running program
 - Address space: virtual memory unique to each process
 - System calls: operating system requests

3 Some execution-driven simulators

- RSIM: user-level SPARC V8 simulator
 - + detailed CPU and memory model
 - very restricted system call library
 - requires re-linking (and possibly source editing)
 - + open source
 - UserSim (on Sparc Sulima) uses RSIM's C library
- SimICS: commercial complete machine simulator, including SPARC V9
 - +/- “had” 32-bit Solaris emulation mode (no longer maintained)
 - closed source
- others: Shade, SimOS

4 Sparc Sulima

- SPARC-V9 ISA complete machine simulator
- currently targeting UltraSPARC I and II (plans for USIII)
- Written in C++, with some SPARC assembler for optimisations
- Python scripting interface (via SWIG)
- Efficient instruction decoding via a SLED specification of SPARC V9
- Instruction decode is cached (as part of simulated I\$)
- Other optimisations: GPR caching, SPARC-on-SPARC simulation
- Portable: tested on PowerPC and x86 (as well as SPARC)

5 Why System Call Emulation?

- Already have OS-Emulation mode (UserSim):
 - has its own C-library (from RSIM, based on glibc)
 - quite limited in what system calls are provided (no mmap!)
 - results not identical to host executables (e.g., floating point printf)
- Complete machine boot so far unsuccessful:
 - not enough experience in low level boot sequence
 - we are still working on it, new student is keen and experienced
- Gain experience in internals of Solaris: useful for future full machine boot
- Implementing our own memory manager is an interesting challenge
- Easier to change architecture (e.g., MMU, cache sizes) than with complete machine simulation (would likely require OS modifications)

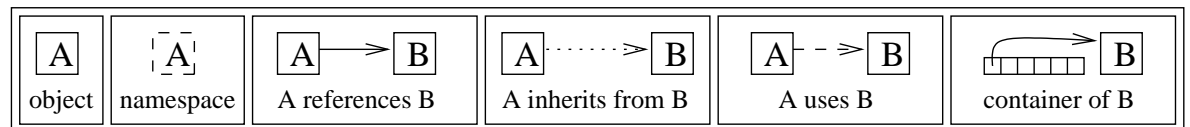
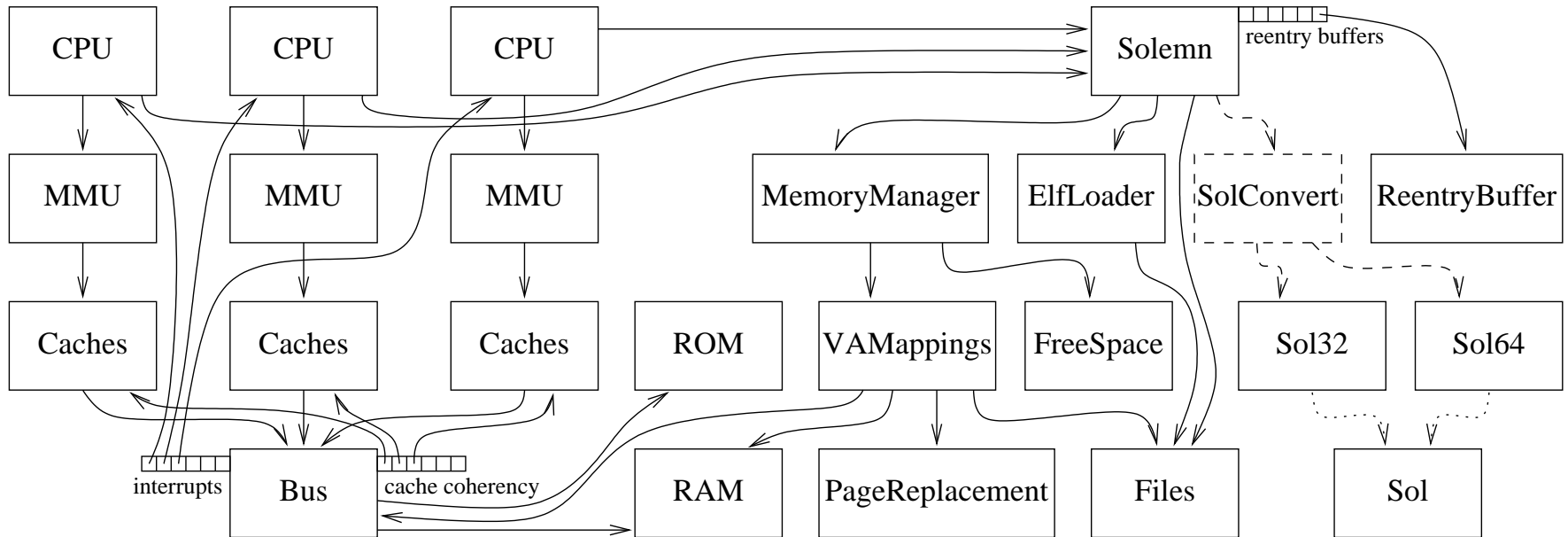
6 Traps and System calls

- Solaris user programs communicate with the operating system call via system calls; these are implemented using the `TCC` instruction
- A `TCC` instruction contains a trap number (0 to 127)
- Solaris defines operating system actions given different trap numbers:
 - SunOS 4.x, Solaris 32-bit and Solaris 64-bit system calls
 - `%g1` determines the actual system call
 - `%o0 ... %o5` are the parameters
 - `%o0` (+ possibly `%o1`) are the return values
 - return value is an error number if the condition code register is set
 - flush/clean register windows, 32-bit get/set CCR, fast system calls (e.g., `gethrtime`, `gettimeofday`)
- Solaris defines 231 system calls (32 and 64-bit)

7 So what is involved?

- Boot / initialisation
- System call emulation:
 - intercept system call (`TCC` instruction)
 - emulate the effect of the system call (inc. target \leftrightarrow host translation)
 - return to simulator
- Memory management:
 - virtual memory
 - virtual to physical translation
 - page misses, faults, swap
 - memory maps
- File management:
 - target \leftrightarrow host file descriptor translation
 - target \leftrightarrow host file *name* translation (`chroot`)

8 Structure of Sparc-Sulima with Solemn



10 Reentrant system call handlers

- Solemn communicates with the simulation's memory via the MMU
- This can cause exceptions: data page miss or protection fault
- This exception must be handled before the system call can continue:
 - generate the exception and return control to the simulator
 - exception handled by our Solemn-specific trap table
 - the TTE is entered into the TLB and the faulting instruction is retried
- In general, the system call should restart where it failed
- Host buffer and state stored in a ReentryBuffer until system call complete
- Threading: multiple ReentryBuffers, indexed by CPU id

11 Memory management

- **MemoryManager**: responsible for all handling memory-related
 - traps: page misses and protection faults; and
 - system calls: `brk`, `mmap`, `munmap`, ...
- **Contains FreeSpace and VAMappings managers**:
 - **FreeSpace**:
 - unused virtual address intervals
 - free space search (for unfixed `mmap`)
 - **VAMappings**:
 - current virtual address mappings and host pointers
 - physical address assignments: if (virtual) page is on RAM
 - reverse lookup: given RAM page (PA), get virtual address
 - page replacement: virtual memory (least recently used)
- Currently pages are fixed at 8KB

12 Benchmarks

- Forte 8, and g++ 3.2.3, 32 and 64-bit builds, high optimisation
- Sun Blade 1000, 2×750 MHz US-III, 2 GB RAM, 8 MB E\$, Solaris 9
- empty.c: (i.e., `int main() { return 0; }`)

| mode | build | #instructions |
|---------|----------------|---------------|
| UserSim | 32-bit static | 258 |
| Solemn | 32-bit static | 649 |
| Solemn | 32-bit dynamic | 224913 |
| Solemn | 64-bit dynamic | 220064 |

- Matrix factorisation (g++64 Solemn)

| options | 500×500, bf=1 | | | | 1000×1000, bf=1 | | | | 1000×1000, bf=64 | |
|-------------|---------------|-------|---------|-------|-----------------|-------|---------|-------|------------------|-------|
| | static | | dynamic | | static | | dynamic | | dynamic | |
| exe build | | | | | | | | | | |
| sim build | native | g++64 | native | g++64 | native | g++64 | native | g++64 | native | g++64 |
| time (secs) | 0.74 | 182 | 0.80 | 185 | 5.25 | 1364 | 5.37 | 1382 | 1.73 | 733 |
| slowdown | 1 | 246 | 1 | 231 | 1 | 260 | 1 | 257 | 1 | 424 |
| MFLOPs | 125 | 0.5 | 119 | 0.5 | 137 | 0.5 | 136 | 0.5 | 534 | 1.1 |

13 Current status, problems, and todo

- Current status:
 - simulate 32-bit and 64-bit executables
 - statically or dynamically linked
 - single or multi-threaded (simulator is not threaded)
 - more than 50 system calls implemented
- Problems:
 - debugging: Sparc Sulima is large: over 50000 lines of code, of which 9000 are Solemn
- Todo:
 - more system calls
 - performance evaluation: Gaussian kernels
 - UltraSPARC III memory model (Fireplane)