Benefits of System Simulation for Multicore Software Development

Ross Dickson
Principal Technology Specialist, Virtutech, Inc.
Agenda

• Overview of Simics and Virtual Platforms

• Developing Software for Multicore can be hard!

• How Virtual Platforms can help with multicore debugging
  • Debugging examples
  • Real-world examples
  • For the skeptic...

• Summary

• Q&A
What is Simics?

Simics® - an adaptive simulation platform that enable customers to define, develop and deploy electronics systems more efficiently.
Simics Key Characteristics

- **High Performance** – Fast and accurate enough for developers to run *real* system software loads
- **High Fidelity** – Functionally accurate & software transparent
- **Full System Visibility** – Everything can be seen and modified
- **Run-to-Run Repeatable and Controllable** – Hardware state, IO streams & Time are replayable.
- **Customizable** – Any system can be virtualized
What is a Virtual Platform?

- A piece of software
- Running on a regular PC, server, or workstation
- Functionally identical to the target hardware
- Runs the same software as the physical hardware system
What Types of Systems Can be Simulated?

Examples

- Freescale PPC cores such as e300, e500-mc, e600
- Freescale QorIQ P4080, MPC8572E, MPC8548E, MPC8548EDS, MPC8572DS
- Telecom rack, avionics
- Satellite, telecom network, backbone network
Multicore Computing and Parallel Software

Click to edit Master subtitle style
Future Embedded Systems Template

One shared memory space

Network with local memory in each node
Software Development for Multicore

- Parallelism required to gain performance
  - Parallel hardware is “easy” to design
  - Parallel software is (very) hard to write

- Fundamentally hard to grasp true concurrency
  - Especially in complex software environments

- Existing software assumes single-processor
  - Might break in new and interesting ways
  - Multitasking no guarantee to run on multiprocessor

- These are difficult issues for software developers
  - Requires additional tool support
  - Physical hardware is often not the best development platform
(Embedded) Software Reality Today

• Programmers used to single-threaded programs

• Legacy code in C, C++, Java, Ada, assembler
  - Essentially sequential languages

• Fine-grained parallelism added to sequential code
  - OpenMP, pthreads, OS threads, MPI, special C variants, Java threads, Ada concurrency, ...

• Debuggers designed for single processors
  - Or multiple instances of single processors
Multiprocessors & Debug

• **Limited visibility into hardware**
  - Single debug port, multiple processors
  - High speed, concurrent execution

• **Timing-sensitive chaotic behavior**
  - Small changes in timing alters system behavior radically
  - Hardware variations impact software behavior

• **Lack of determinism**
  - Rerunning a program gives different results
  - Hard to reproduce bugs

• **Heisenbugs**
  - Inserting probes to trace behavior alters behavior
  - Bugs hide when they are being debugged

• **System keeps running even if one core stopped**
Code is not just about CPUs

On a modern SoC, the processor cores are just one part of the system.

Much application functionality is implemented by using special accelerators... and you need to debug their interaction with the processors & software.
Virtual Platforms Makes Debugging Multicore Applications Easier

Click to edit Master subtitle style
Three Steps of Debugging

1. Provoking errors
   - Forcing the system to a state where things break

1. Reproducing errors
   - Recreating a provoked error reliably

1. Locating and fixing errors
   - Investigating the program flow and data
   - Depends on success in reproduction

Virtual Platforms helps with all three steps
Provoking Errors

- Virtual Platforms provides complete control over system configuration and execution
- Replicate failing tests from production units
- Vary system hardware configuration
  - Like testing on a variety of real-world machines
- Vary system software configuration
  - Easy to test different software loads on different machines
- Systematically provoke corner cases
  - Use oddball processor counts
  - Make a processor slower or stop it entirely
  - Increase communication latencies
  - Slow down individual processors to increase perceived load
Typical “untested configuration because the hardware did not exist” error
Reproducing Errors

• Virtual hardware state can be checkpointed

• Virtual hardware execution is deterministic
  - Simulation engine imposes a well-defined sequential semantic to the parallel execution of the target machine
  - All machine-internal events have deterministic time
  - Input from real world recorded & replayed

• An error is provoked in simulation can be reproduced
  - Reset back to initial state (or restore a checkpoint)
  - Rerun the test case that ended in error
  - Same error state results
  - ...any number of times
  - ...on any machine running the simulator
  - ...from a checkpoint distributed to multiple developers
Locating and Fixing Errors

- Determinism and control key features
- No probe effect from instrumentation
  - Tracing and observation from the outside, not by code mod
- No timing disturbance from debugging
  - Breakpoints behave like hardware breakpoints
- Global system stop
  - For practical reasons, this has a small skid of 1-100kcycle
- Heisenbugs cannot occur
  - No intrusion in timing behavior, no varying behavior
Repeatability and Reverse Debugging

• **Repeat any run trivially**
  - No need to rerun and hope for bug to reoccur

• **Stop & go back in time**
  - Instead of rerunning program from start
  - Breakpoints & watchpoints backwards in time
  - Investigate exactly what happened this time

• **This control and reliable repeatability is very powerful for parallel code!**

On hardware, only some runs reproduce an error

On virtual hardware, debugging is much easier
Some (Real!) Multicore Bug Stories

Click to edit Master subtitle style
Divide-by-zero in OS Kernel

• Operating-system kernel crash in virtual model
  - Divide-by-zero right in the kernel
  - Algorithm to determine and compensate for clock skew
  - Division by difference in time between two processors

• Virtual model had zero clock skew = provoked error
  - Could have happened on a real system
  - Just not very likely
  - Typical rare problem in the field
  - Essentially testing a rare corner case in system state
Race Condition in Serial Driver

• The problem:
  - Dual-core MPC8641D machine
  - Changed clock frequency from 800 to 833 Mhz
  - OS froze on startup – quite unexpectedly

• Investigation:
  - Only happened at 832.9 to 833.3 MHz
  - Determinism: 100% reproduction of error trivial
  - Time control: single-step code feasible
  - Insight: look at complete system state, log interrupts, check the call stack at the point of the freeze, check lock state

• What we found:
  - An interrupt service routine attempted to take a lock, before re-enabling interrupts. In the case that froze, the lock was already taken when the service routine was entered, and with no interrupts enabled there was no way for it to be released.
The Disk Corruption Example Bug

- **Distributed fault-tolerant file system got corrupted**
  - Rack-based system with many (single-processor) boards
  - Intermittent error
  - Error seen as a composite state across multiple disks: they got inconsistent, for some reason
  - Months spent chasing it on physical hardware

- **Simics solution:**
  - Reproduce corruption in Simics model of target
  - Pin-point time when it happens, by interval halving
  - Around the critical time, take periodic snapshots of disks
  - Check consistency of disk states in offline scripts

- **Result:**
  - Found the *precise instruction* causing the problem
  - Capture the network traffic pattern causing the issue
  - Communicated the complete setup and reproduction instructions to development, greatly facilitating fixing the bug
Validity of Virtual Debug

Still a skeptic?
“But it is just a simulation”

• Sometimes people do not believe in debugging using a simulator... it is just a simulation after all, not the real thing

• All experience contradicts this
  - Any bug found in a virtual environment is a real bug!
  - Our customers are very happy with virtual debugging

• Experimental evidence backs up usefulness
  - As discussed later…
Simplified Timing and Multicore

- **Simplified simulation is necessary** to run workloads
  - Simplify target timing to get performance
  - Billions and billions of instructions to execute

- **You still expose common concurrency bugs and coarse-grain performance effects**
  - Race conditions
  - Missing locks
  - Lock contention
  - Locking overhead

- **The way to find bugs is not precise simulation of the target but deliberate variation**
  - Do not rely on physical randomness
  - Introduce controlled patterns, known variations
  - Rely on repeatability of the simulator to find root errors

- **Low-level code sometimes break because of timing-related details such as cache coherency issues**
  - Use **hybrid simulation** to investigate failure scenarios with full timing and pipeline and memory system details
Locking Test Program: Find Race

- Test on single core and dualcore setups
  - Range of frequencies
  - Test program run 20 times on each setup
  - Count percentage of runs triggering race

- Results:
  - Race always triggers in dual-CPU mode
  - Triggers around 10% in single-CPU mode
  - Higher clock = less chance to trigger

- Simulator: simple timing, quantum 1000

- Simplified timing does not hide the race!
Locking Test Program: Contention

**Observations:**
- Locking overhead visible
- Lock contention visible
- Only proper locking varies in execution time

**On real hardware:**
- no << fake << proper
- Same relation seen in simulation, even if magnitude varies

**Test program details**
- 2 threads
- 1000000 iterations
- MPC8641D virtual target
- All locking disciplines
- Time quantum 10-10000
Tools for Multicore

• PolyCore Software™ provides tools and target-resident software for the configuration and deployment of fast, flexible, configurable, and hardware transparent communications solution for multicore and multiprocessor systems.

• Combined Virtutech/PolyCore Benefits:
  - Improved Early-Stage System Architecture Definition
    § PolyCore solutions can be deployed across any number of Simics virtual platforms or system models.
  - System Debugging
    § PolyCore support for inter-processor communications complements Simics ability to accurately model multicore systems.
  - Flexibility and Scalability
    § The PolyCore software is flexible and configurable allowing diverse systems to be modeled on Simics.
Tools for Multicore

- CriticalBlue® Prism™ allows engineers to take their existing sequential code and, without changing it, explore and analyze opportunities for concurrency, implement parallel structures, and verify efficient and safe operation across multicore processors.

- Combined Virtutech/CriticalBlue Benefits include:
  - Gather and Analyze Detailed Hardware Traces:
    § Detailed hardware trace information is available from any model or CPU that Simics supports.
    § Trace information is used by Prism to highlight where and how the software can be optimized for parallelism and concurrency.
  - Validate software changes on multicore system architectures:
    § Simics flexible virtual model complements Prism’s ability to run what-if analysis that may be used to rapidly explore the impact of different threading strategies, numbers of cores, dependency removal, and scheduling policies.
  - System Debugging:
    § Simics provides a full system stop which enables bugs to be identified and resolved without the rest of the system or CPU cores moving past a point of detection and correction.
## Simics Models for Freescale CPU/SoC Devices

<table>
<thead>
<tr>
<th>Target CPU / SoC</th>
<th>Reference Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>e300</td>
<td>Freescale QorIQ P4080</td>
</tr>
<tr>
<td>e500</td>
<td>Freescale MPC8572E</td>
</tr>
<tr>
<td>e500-mc</td>
<td>Freescale HP-Net MPC8641D</td>
</tr>
<tr>
<td>e600</td>
<td>Freescale MPC8323</td>
</tr>
<tr>
<td>MPC603e</td>
<td>Freescale MPC8360</td>
</tr>
<tr>
<td>MPC750, MPC755 (&quot;G3&quot;)</td>
<td>Freescale MPC8540ADS</td>
</tr>
<tr>
<td>MPC74xx (&quot;G4&quot;)</td>
<td>Freescale MPC8548CDS</td>
</tr>
<tr>
<td>QorIQ™ P4080</td>
<td>Freescale MPC8555CDS</td>
</tr>
<tr>
<td>PowerQUICC II (MPC82xx)</td>
<td>Wind River SBC7447/57</td>
</tr>
<tr>
<td>PowerQUICC II Pro (MPC83xx)</td>
<td>Curtiss-Wright SVME-183</td>
</tr>
<tr>
<td>PowerQUICC III (MPC85xx)</td>
<td></td>
</tr>
<tr>
<td>MPC8641/D</td>
<td></td>
</tr>
</tbody>
</table>

Summary

- Virtual hardware provides an additional tool for embedded software development

- Frees software from hardware dependence

- Especially useful for tough show-stopper bugs
  - Parallel software
  - Hardware-software interaction
  - "Heisenbugs"
Thank you!

Click to edit Master subtitle style