What is Virtualized System Development?

Dan Ekblom, PhD
Senior Application Engineer
Virtutech Simics

- **Full system simulation**
  - Complete machines, networks, backplanes
  - System-level from the beginning

- **Runs complete software stack**
  - Firmware, device drivers, OS, hypervisor, etc...

- **Very high performance**
  - Typically 100s of MIPS
  - Multiple GIPS top benchmark
Traditional System Development

- Software development methodology creates production binary
- Production binary runs on the real hardware
Same binary runs inside virtualized system development environment

The software can’t tell the difference

Virtual HW

Application stack
Operating system
Hardware-sensitive software
What is modeled in VSD?

- Identical build tools chain
- Runs binaries from real target

User program

<table>
<thead>
<tr>
<th>Server</th>
<th>DB</th>
<th>Middleware</th>
</tr>
</thead>
</table>

Operating system

<table>
<thead>
<tr>
<th>Drivers</th>
<th>Firmware</th>
</tr>
</thead>
</table>

Hardware

- CPU
- RAM
- ROM
- Bus
- PCI
- I2C
- PCI
- Disk
- Disk Ctrl
- FLASH
- Network

Simulated hardware

Complete production software

Copyright ©2009. Virtutech, Inc. All rights reserved.
Virtualized System Development: Enables Changes!

- Hardware Development
- Coding, Integration & Test
- Application Stack

Virtualized Software Development

Benefits:
- Lower Risk
- Faster Time-to-Market
- Better Quality

Copyright ©2009. Virtutech, Inc. All rights reserved.
“Simulation is the key to advanced microprocessor development, and Simics is by far the most advanced realization of this technology available. Our vision is to eventually simulate the entire code stack from firmware up, and Virtutech’s Simics will be the cornerstone of this development.”

Kevin Collins, Director, Global Firmware Development, IBM

“Debug with Simics is 4-8 times faster than with hardware”

Tracy Bashore, Manager SLIC storage management development, IBM

“Simics is really the only way to develop multi-core software”

Tomas Evensen, CTO, WindRiver

“The processing potential of multi-core devices remains untapped because multicore systems are only as effective as software’s ability to handle parallelism”

Chekib Akrout, VP & GM Networking System Division, Freescale

“Simics allows us to test our software and validate it while the underlying hardware design is being”

Gerry Vossler, VP, Advanced Marketing & Technology
In the virtual world, anything is possible

Synchronous stop for entire system

Determinism and repeatability

Reverse execution

Unlimited and powerful breakpoints

Trace anything

Insight into all devices

break -x 0x0000->0x1F00
break-io uart0
break-exception int13
The entire system can be stopped, inspected and debugged at any time
Traditional debug: A Single Component may stop …

Connection to the World

Identical platforms

Network hubs & switches

Chassis and Racks

Interface:
- Backplane
- RapidIO
- PCI-express
- shared memory

Mixed Architectures

Dedicated Subsystems

Multicore Boards

… but the rest of the system continues to run
VSD debugging: Synchronized System Stop

Identical platforms

Interfaces:
- Backplane
- RapidIO
- PCI-express
- Shared memory

Chassis and Racks

Network hubs & switches

Mixed Architectures

Multicore Boards

Dedicated Subsystems

Connect to the World

... the whole system freezes in an operational state

Copyright ©2009. Virtutech, Inc. All rights reserved.
A virtual system can be frozen, captured, and restored at any time, location or computer ... without replication errors
Taking a Check Point

Identical platforms

Interfaces:
- Backplane
- RapidIO
- PCI-express
- shared memory

Chassis and Racks

Network hubs & switches

Mixed Architectures

Dedicated Subsystems

Multicore Boards

Connect to the World
Sending the Check Point
Restore and Run

Restore the checkpoint and resume
• At any time
• In any location
• On any computer
The “path” taken through code execution is repeated on every run (determinism) … until stimuli are specifically modified.
• Physical systems are not wholly predictable or controllable
• The system will usually follow a slightly different path from start to finish
• Some runs will hit bugs, others will not.
• Physical systems are not wholly predictable or controllable.
• The system will usually follow a slightly different path from start to finish.
• Some runs will hit bugs, others will not.
Physical systems are not wholly predictable or controllable
The system will usually follow a different path from start to finish
Some runs will hit bugs, others will not.
Run 1

- Simics virtual platforms are predictable and controllable
- The system will follow exactly the same path from start to finish
  - Every developer will precisely duplicate every execution step
• Simics virtual platforms are predictable and controllable
• The system will follow exactly the same path from start to finish
  • Every developer will precisely duplicate every execution step
Repeatability – Virtual Hardware

Run 3

- Simics virtual platforms are predictable and controllable
- The system will follow exactly the same path from start to finish
  - Every developer will precisely duplicate every execution step
Run 4 (new stimuli)

- New stimuli can be injected to ensure different paths
- Random paths can be generated
Physical systems can only run forward

... requires traditional iterative debug approaches
Iteratively Converging on the Problem

Traditional Hardware Debug

1. Guess where to set a break point
2. Inspect stack
3. Move break point
4. Restart or reboot
5. Repeat
Iteratively Converging on the Problem

Traditional Hardware Debug

1. Guess where to set a break point
2. Inspect stack
3. Move break point
4. Restart or reboot
5. Repeat
Iteratively Converging on the Problem

Traditional Hardware Debug

1. Guess where to set a break point
2. Inspect stack
3. Move break point
4. Restart or reboot
5. Repeat
Iteratively Converging on the Problem
Traditional Hardware Debug

1. Guess where to set a break point
2. Inspect stack
3. Move break point
4. Restart or reboot
5. Repeat
Iteratively Converging on the Problem
Traditional Hardware Debug

1. Guess where to set a break point
2. Inspect stack
3. Move break point
4. Restart or reboot
5. Repeat
Iteratively Converging on the Problem

Traditional Hardware Debug

1. Guess where to set a break point
2. Inspect stack
3. Move break point
4. Restart or reboot
5. Repeat
Iteratively Converging on the Problem

Traditional Hardware Debug

1. Guess where to set a break point
2. Inspect stack
3. Move break point
4. Restart or reboot
5. Repeat
A system runs backwards, through every operation and breakpoint along the way.

A new paradigm to debug and investigate problems.
Linearly Converging on the Problem
VSD Debug

Stack
G G G G G B B B B B

Stack is now at the last known bad point

- Begin after the problem occurs
- Set breakpoint on OS kill signal
- Run in reverse up to breakpoint

Reverse & Stop
Stack

G G G G G G G B B B B

Stack is known bad at this point

While observing the stack, run in reverse, stopping at breakpoints along the way
Stack

Stack is known bad at this point

While observing the stack, run in reverse, stopping at breakpoints along the way.
While observing the stack, run in reverse, stopping at breakpoints along the way.

Stack
G G G G G G G G G B B

Stack is known bad at this point.
Now, set a watchpoint on the corrupt variable and resume reverse execution.

This is where the first bad stack frame appears
Stack

- Watchpoint triggers & execution stops
- Debugger points to offending line of source code.
Virtual Systems Development - Summary

- Reduces the risk in software projects, decouples hardware and software development.
- Very efficient platform for full system multisystem/multicore debug.
Virtual Systems Development - Summary

- Reduces the risk in software projects, decouples hardware and software development.
- Very efficient platform for full system multisystem/multicore debug.
- Also possible to use Simics for architectural exploration.
Virtual Systems Development - Summary

- Reduces the risk in software projects, decouples hardware and software development
- Very efficient platform for full system multisystem/multicore debug

- Also possible to use Simics for architectural exploration:
  - Adding more cores
  - Comparing different architectures
  - Adding cache models
  - The limit is the sky…
Cache Modeling in Simics

Basic model
1 instr = 1 cycle
No cache, perfect memory
100+ MIPS speed

Cache model
Compute instr = 1 cycle
Memory instr = cache time
Cache statistics & traces
1+ MIPS speed
What is Virtualized System Development?

Dan Ekblom, PhD
Senior Application Engineer