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

Virtualized System Development

- Same binary runs inside virtualized system development environment

The software can’t tell the difference
What is modeled in VSD?

Virtualized System Development: Enables Changes!

Customer Experience

“In 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 Ahroud, 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
SYSTEM STOP

The entire system can be stopped, inspected and debugged at any time.

Traditional debug: A Single Component may stop …

VSD debugging: Synchronized System Stop

… the whole system freezes in an operational state

SYSTEM CHECKPOINTS

A virtual system can be frozen, captured, and restored at any time, location or computer

… without replication errors
Taking a Check Point

Identical platforms

Chassis and Racks

Network hubs & switches

Mixed Architectures

Dedicated Subsystems

Multicore Boards

Connecting to the World

Restore the checkpoint and resume

- At any time
- In any location
- On any computer

Sending the Check Point

Multicore Boards

Dedicated Subsystems

Mixed Architectures

Network hubs & switches

Chassis and Racks

Identical platforms

Backplane

RapidIO

PCI-express

shared memory

Connection to the World

Restore and Run

Mixed Architectures

Dedicated Subsystems

Multicore Boards

Chassis and Racks

Identical platforms

Backplane

RapidIO

PCI-express

shared memory

Connection to the World
Restore and Run

Identical platforms

Chassis and Racks

Network hubs & switches

Mixed Architectures

Dedicated Subsystems

Multicore Boards

CONNECTED TO THE WORLD

RUN TO RUN REPEATABILITY

The “path” taken through code execution is repeated on every run (determinism)

… until stimuli are specifically modified

Repeatability - Traditional Hardware

Run 1

Run 2

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.

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

REVERSE EXECUTION

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
GGGGGGBBB
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

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

Stack
GGGGGGGBBB
Stack is known bad at this point
Linearly Converging on the Problem
VSD Debug

Stack
GGGGGGGGBB

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

Stack is known bad at this point

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

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

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

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:
What is Virtualized System Development?

Dan Ekblom, PhD
Senior Application Engineer