Software Quality is Directly Proportional to Simulation Speed

CDNLive!
11 March 2014

Larry Lapides
Software Quality is Directly Proportional to Test Speed

- Intuitively obvious (so my presentation is done!)
- How to achieve more speed?
- How to find more bugs?
- How to know that quality has improved?

Greater Quality

More Bugs Found

More Tests Run

Faster Tests
Cadence-Imperas Integration
Supports Simulation, Debug and Software Development & Test Tools

Verification, Analysis & Profiling (VAP) Tools

- CPU and OS awareness
- Tracing, profiling, coverage, memory analysis, ...
  - Over 25 different tools
- User extendable

Cadence Virtual System Platform

SystemC/TLM-2.0

Cadence and GDB debuggers supported
Agenda

- Simulation speed
- Simulation based tools for finding bugs
- Quality metrics
- Case study: OS porting, bring up and verification on Altera Cyclone V SoC FPGA
- Summary
Agenda

- Simulation speed
  1) Start with a faster simulation engine
  2) Use the multiple cores available in the host PC
- Simulation based tools for finding bugs
- Quality metrics
- Case study: OS porting, bring up and verification on Altera Cyclone V SoC FPGA
- Summary
Latest Many-Core Platforms Require Scalable Simulation

Server SoC software test suites can consist of 10s or 100s of tests, each executing 10s or 100s of billions of instructions.

Currently: Single threaded simulation does not scale with multicore platforms.

Simulation market leader’s solution would take 1 week to simulate that test suite.

Challenges: Get needed simulation speed and fix platform simulation scaling problem.
1) Start with a Faster Simulation

---

**Imperas Simulator Benchmarks**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Altera Nios II</th>
<th>ARM32</th>
<th>Imagination MIPS32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simulated Instructions</td>
<td>Run time</td>
<td>Simulated MIPS</td>
</tr>
<tr>
<td>Linpack</td>
<td>3,075,857,171</td>
<td>2.52s</td>
<td>1225</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>1,810,082,387</td>
<td>1.18s</td>
<td>1547</td>
</tr>
<tr>
<td>Whetstone</td>
<td>6,850,867,389</td>
<td>3.26s</td>
<td>1789</td>
</tr>
<tr>
<td>peakSpeed2</td>
<td>22,000,013,458</td>
<td>3.11s</td>
<td>7097</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Xilinx MicroBlaze</th>
<th>ARM AArch64</th>
<th>Imagination MIPS64</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simulated Instructions</td>
<td>Run time</td>
<td>Simulated MIPS</td>
</tr>
<tr>
<td>Linpack</td>
<td>6,386,275,169</td>
<td>3.77s</td>
<td>1699</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>3,770,115,740</td>
<td>2.61s</td>
<td>1450</td>
</tr>
<tr>
<td>Whetstone</td>
<td>27,108,532,655</td>
<td>13.23s</td>
<td>2054</td>
</tr>
<tr>
<td>peakSpeed2</td>
<td>22,000,023,433</td>
<td>6.76s</td>
<td>3826</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>PowerPC</th>
<th>Renesas v850</th>
<th>Synopsys ARC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simulated Instructions</td>
<td>Run time</td>
<td>Simulated MIPS</td>
</tr>
<tr>
<td>Linpack</td>
<td>3,163,956,113</td>
<td>2.95s</td>
<td>1076</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>2,206,082,239</td>
<td>1.75s</td>
<td>1260</td>
</tr>
<tr>
<td>Whetstone</td>
<td>6,424,865,755</td>
<td>3.97s</td>
<td>1622</td>
</tr>
<tr>
<td>peakSpeed2</td>
<td>22,400,002,937</td>
<td>5.6s</td>
<td>4007</td>
</tr>
</tbody>
</table>

All measurements on 3.4GHz Intel i7-3770, Linux, QVPath 2014.0127.0

* Hardware Floating Point Instructions
Imperas: Fastest Virtual Platform Solution Available

Just In Time (JIT) Code
Morphing Simulator

Virtual platform with ARM Cortex-A9, single thread simulation
2) Use the Multiple Cores Available in the Host Machine

- Multiple cores for parallel simulation should result in performance gains.
- Previous attempts at using multiple cores have been unsuccessful due to high overhead from synchronization of multiple simulation threads.
QuantumLeap: 15x Faster Than Next Fastest Solution

- Advanced parallel synchronization algorithm for SMP, AMP and hardware accelerators
- Transparent operation to user: No model, tool, software changes
- Total performance on benchmarks recorded up to 16K MIPS
- Accelerates execution 2-3x over current simulation performance (already 6x faster), 15x over nearest alternative solution
SMP Acceleration Results

Simulated Applications Running Under Simulated Linux

- QuantumLeap speeds up Imperas SMP models by 2.25x on average for quad core SMP and host
- Works for Imperas OVP Fast Processor Models of SMP cores even when used in a SystemC platform

All benchmarks run on ARM Cortex-A9MPx4 models
Agenda

- Simulation speed
- Simulation based tools for finding bugs
- Quality metrics
- Case study: OS porting, bring up and verification on Altera Cyclone V SoC FPGA
- Summary
Imperas technical conclusion at founding: typical software simulation flow is not adequate for software development, debug and test.

How to get full observability, controllability – *the promise of simulation* – from virtual platforms?
- Need minimal overhead to maximize performance
- Need to maintain order of instruction execution
- Cannot introduce new “bugs” through the act of observation

How to get near real time simulation performance?

**Solution:** innovation in both simulator engine and processor model
- Architect the simulation environment, from the beginning, for performance and tools; software tools should not be an afterthought

- OVP Fast Processor Models contain special information for tools
- SlipStreamer libraries for tools
  - Non-intrusive: no modification of source code
  - Executes as native host code for minimal overhead
- ToolMorphing engine tightly integrates models and tools
**ToolMorphing Technology**

**Enables Tool Definition**

| VAP Tool (from Imperas) or User-Defined Tool: |
| Definition of the tool, written in C, included in simulation environment |

| Tool Helper: |
| API enabling definition of software analysis tools |

| CPU and OS Helpers: |
| CPU and OS specific information |

| OVP Processor Model: |
| CPU functionality, predefined views, events, actions |

| Simulation Engine: |
| Just In Time (JIT) code morphing (binary translation) |

---

**Application Software & Operating System**

- binaries

- Virtual Platform
  - simulation engine
  - OVP CPU Model
  - CPU, OS Helpers
  - Tool Helper
  - instrumentation

**results**
Verification, Analysis & Profiling (VAP) Tool Suite for HDS Development

Operating System

Bare Metal Apps & Middleware

Platform (e.g. Drivers)

- Multi Processor Debug
- Address space introspection
- Virtual2physical mapping
- Print CP registers
- TCL callbacks

- Break on line
- Break on function call
- Elf introspection
- Unlimited HW breakpoints
- Memory region watchpoints
- Trace source line
- Trace context
- Trace functions
- Line Coverage
- Instruction profiling
- Function profiling
- Heap checks
- Stack checks
- Malloc checks
- Semaphore checks

Processor

- Trace coprocessor registers
- Trace TLB trace exceptions
- Trace modes
- Trace service calls
- Trace hypervisor calls
- Trace secure monitor calls
- Trace MT/MP extensions
- Trace system calls
- Trace timer
- Trace cache instructions
- Trace SIMD extensions
- Trace instruction
- Trace register change

Simulator

- Break on messages
- TCL callbacks
- Full GDB command set

- Drivers
- Firmware
- Assembly libraries
- OS porting and bring up
- Hypervisors

- Multiprocessor, multicore, multithread, multi-everything
- Non-intrusive
- Low overhead
- User extendable
Agenda

- Simulation speed
- Simulation based tools for finding bugs
- Quality metrics
- Case study: OS porting, bring up and verification on Altera Cyclone V SoC FPGA
- Summary
Code Coverage

- Non-intrusive statement and branch coverage analysis using intercepts
- Full multiprocessor, multicore, and peripheral code coverage

Runtime analysis

Standard .lcov file format
TN:cpuA.lcov
SP:cpuA
SF:/home/graham/mpeg2decode/src/getbits.c
DA:44,3
DA:45,3
DA:46,3
DA:47,3
DA:58,3
DA:59,3
DA:60,3
DA:63,1151
DA:66,1151
DA:67,1151
...


Agenda

- Simulation speed
- Simulation based tools for finding bugs
- Quality metrics
- Case study: OS porting, bring up and verification on Altera Cyclone V SoC FPGA
- Summary
OS Porting, Bring Up and Verification on Altera Cyclone V SoC FPGA

1) Linux boot on single core ARM Cortex-A9
2) SMP Linux boot on dual core ARM Cortex-A9
3) RTOS boot on single core ARM Cortex-A9
4) AMP boot on dual core ARM Cortex-A9
Cyclone V SoC FPGA Virtual Platform

- Top level virtual platform built using Open Virtual Platforms (OVP, www.OVPworld.org) ICM API
- ARM Cortex-A9MPx2 and Altera Nios II processor core models from the OVP Library
- Peripheral models
  - Some models available in the OVP Library
  - Remaining models of peripheral components developed using OVP APIs
- OVP APIs written for C language
- Simulation engine: Imperas M*SDK

- All OVP processor and peripheral models include both native OVP and native SystemC/TLM2 interfaces, so all the following results could have been achieved using the OSCI SystemC simulator plus Imperas M*SDK product
  - Peripheral models could have been written in SystemC
  - M*SDK tools require OVP processor core models for ToolMorphing capability
1a) Linux Boot on Single Core ARM Cortex-A9

- Use Linux from Altera: Altera-3.4
- Use default configurations
- Use default device trees
  - Comment out a few peripherals not yet modeled
- Bug found in Linux kernel preemptive scheduling
  - Running multiple applications under Linux part of standard Imperas bring up testing
  - Linux boots and runs, but does not switch tasks properly
  - Not observed in previous virtual platform (different virtual platform vendor) using much slower model of ARM Cortex-A9MPx2
    - Could not run multiple applications for long enough simulation to observe the bug

- Approximately 2 man weeks effort to build virtual platform able to boot Linux
- Virtual platform boots Linux in under 5 sec on standard PC, Windows or Linux
1b) OS-Aware Tools Used to Find the Bug

- Use OS tracing [task, execve, schedule, context, …] to trace at the OS level, not instruction level
  - Higher level of abstraction makes debug easier: ~700,000,000 to boot Linux, however, only ~700 tasks
- OS-aware tools debug in hours, once the bug was observed
- Simulation overhead due to OS-aware tools < 10%
2) SMP Linux Boot on Dual Core ARM Cortex-A9

- Use Linux from Altera: Altera-3.6
- Use default configurations
- Use default device trees
  - Comment out a few peripherals not yet modeled
- No problems in SMP Linux bring up on virtual platform
3a) Micrium µCOS-II Boot on Single Core ARM Cortex-A9

- Use Altera µCOS-II release
- Bugs found and fixed in GIC register accesses using OS-aware tools
  - Access ICDICER1 to 8 when only 0 to 7 exist
  - Access ICDIPTR08 to 63 when only 00 to 55 exist
- Typically < 1 week effort to add support for new RTOS
- RTOS OS-aware tools include event scheduler viewing as waveform
3b) OS Porting and Bring Up

- Non-intrusive (no modification of OS source) trace of:
  - process creation
  - context switch
  - process deletion
- Captures communications between processes
- Supported OS include Linux, FreeRTOS, Nucleus, µC/OS
  - < 1 week to support new RTOS
  - View in waveform viewer
4a) AMP boot on Dual Core ARM Cortex-A9

- Linux booting on first core, μC/OS-II on second core
- Bug found in Linux accesses of GIC registers
- Virtual platform debug took 2 days versus 2 weeks on hardware platform (5x improvement)
- Also need to ensure that different operating systems do not access forbidden memory segments
  - Bugs found using custom memory access monitor
4b) Custom Memory Access Monitor Accelerates AMP Platform Debug

- Memory access monitor is just C code, less than 350 lines, loaded into simulation environment
- When simulation is run, monitor produces warning if memory access rules are violated

```c
// Define watch areas for memory and peripherals defined in the platform
memWatchT amcWatch[] = {
    // watchLow    watchHigh   allowedCPUs
    { "Linux memory", 0, 0x2fffffff, LINUX_CPU },
    { "uCOS memory", 0x30000000, 0x31ffffff, UCOSII_CPU },
    { "gmac0", 0xff700000, 0xff700fff, LINUX_CPU },
    { "emac0_dma", 0xff701000, 0xff701fff, LINUX_CPU },
    { "gmacl", 0xff702000, 0xff702fff, LINUX_CPU },
    { "emac1_dma", 0xff703000, 0xff703fff, LINUX_CPU },
    { "uart0", 0xffc002000, 0xffc002fff, LINUX_CPU },
    { "uart1", 0xffc03000, 0xffc03fff, UCOSII_CPU },
    { "CLKMGR", 0xffd04000, 0xffd04fff, LINUX_CPU },
    { "RSTMGR", 0xffd05000, 0xffd05fff, LINUX_CPU },
    { "SYSMGR", 0xffd08000, 0xffd08fff, LINUX_CPU },
    { "GIC", 0xfffe0c000, 0xfffe0dff, LINUX_CPU },
    { "L2", 0xfffe0f000, 0xfffe0fff, LINUX_CPU },
    { 0 } /* Marks end of list */
};
```

Warning (AMPCHK_MWV) cpu_CPU0: AMP write access violation in uart1 area. PA: 0xffc03008 VA: 0xffc03008
Warning (AMPCHK_MWV) cpu_CPU0: AMP write access violation in uart1 area. PA: 0xffc0300c VA: 0xffc0300c
Warning (AMPCHK_MWV) cpu_CPU0: AMP write access violation in uart1 area. PA: 0xffc03010 VA: 0xffc03010
Warning (AMPCHK_MRV) cpu_CPU1: AMP read access violation in Linux memory area. PA: 0x00000020 VA: 0x00000020
Summary

- More processor cores, more complex systems ⇒ more tests are needed
- Simulation speed is critical for running more tests
- Also need tools and metrics, architected into the simulation environment from the start
- Results were shown for SMP and AMP systems on Altera Cyclone V SoC FPGA

Software quality is proportional to simulation speed!