Software Performance Analysis Using High Speed Virtual Platforms

Puneet Maheshwari
Staff CAE, CoWare Inc.

Charu Khosla
Staff CAE, CoWare Inc.

Andreas Wieferink
Staff Software Engineer, CoWare Inc.
Overview

• Goal
• Vision
• Use model
• Use cases
• Solution components
• Driver application
• Results
Goal

Prevent delays in embedded software projects

- **Detect** functional bugs AND performance issues *early*
  - Early and **continuous** software performance validation
    - Before hardware is available
    - Available to all **software developers**
      - On every developers desk
  
- **Fix** functional bugs AND performance issues *faster*
  - Better **observability** and **controllability** than hardware
    - Start and stop complete platform without artifacts
    - Non-intrusive debugging capabilities
    - Quick and in depth investigation of issues
    - Validate deadlines or measure timing between probe points
    - **Detailed analysis views** not available on hardware prototype
      - Quickly navigate from symptom to cause
Modeling Requirements for SystemC TLM

- **Programmers View (PV)**
- **Programmers View With Timing (PVT)**
- **Software Functionality**
- **Software Performance Validation**
- **Architecture View (AV)**
- **Hardware Architecture Validation**
- **Cycle Accurate TLM (CA)**
- **Hardware Functionality**

Log SPEED

- 100Mcps
- 10Mcps
- 1Mcps
- 100Kcps
- 10Kcps
- 1Kcps

Log ACCURACY

Programmers View

100Kcps

1Mcps

10Kcps

1Kcps

Programmers View With Timing

Cycle Accurate TLM

Software Functionality

Software Performance Validation

Architecture View

Hardware Architecture Validation

Hardware Functionality

ESL Design Tasks
Platform Modeling – ESL Opportunity

- Single Platform Model
  - Modeling effort
  - Verification effort

- Log SPEED
  - 100Mcps
  - 10Mcps
  - 1Mcps
  - 100Kcps
  - 10Kcps
  - 1Kcps

- Log ACCURACY

- ESL Design Tasks
  - Hardware Architecture Validation
  - Hardware Functionality
  - Software Performance Validation

- Programmers View
- Programmers View With Timing
- Software Functionality
- Architecture View (AV)
- Programmers View (PV)
- Software Functionality
- Hardware Functionality
- Cycle Accurate TLM (CA)

- ESL Design Tasks

- Platform Modeling – ESL Opportunity

- © CoWare, Inc. 1997-2007

NASCUG DAC 2007
Software Performance Analysis: Use Model

Virtual platform

Boot Operating System

Validate Software Performance

Billions of instructions

Millions of instructions

PV (Fast)

PVT (Accurate)
Software Performance Analysis: Key Solution Components

• Accuracy on demand capabilities, through support of multiple simulation modes for SystemC TLM
  – Processors
  – Memory subsystem
  – Frequently accessed peripherals

• Software performance measurement capabilities
Accuracy on Demand Simulation Modes

- **Software Development mode**
  - Ultra fast mode for software development and debugging
  - Instruction Accurate processor
  - Register accurate peripherals
  - No timing accuracy requirement

- **States In-Synch mode**
  - Fast simulation mode for reaching a software use case faster
  - Peripherals and Processor in same state as they would have been in a cycle accurate simulation
  - Instruction Accurate processor
  - Timing accurate platform – Timing modeled at PV+T level using timing annotation methodology

- **Timing Accurate mode**
  - Slow but accurate mode for software performance measurement
  - Cycle Accurate processor
  - Timing accurate platform – Timing modeled at PV+T level using timing annotation methodology
  - Capturing analysis
Software Performance Analysis: Use Model

Virtual platform

Boot Operating System

Validate Software Performance

States In-Sync Mode

Timing Accurate Mode

PV (Fast)

PVT (Accurate)
Scenario 1: Engineering level testing

Virtual platform

Boot Operating System

Validate Software Performance

Billions of instructions

Thousands of instructions

Accuracy > 90%/Speed KIPS
- Driver development
- Interrupt routines
- Small software functions
- Hardware interaction
Scenario 2: User level & algorithm testing

- Virtual platform
- Boot Operating System
- Validate Software Performance

Billions of instructions

Millions of instructions

Accuray > 80%/Speed MIPS
- User response time
- Open application
- Open window
- Deadlines
- Samples, Frames/second
Simulation: Software Development Mode

- Ultra fast mode for software development and debugging (~30 MIPS)
- Register accurate peripherals
- No timing accuracy
- L1 or L2 cache not simulated
- Direct memory accesses from the processor
Simulation: States-in-Synch Mode

- Timing inside peripherals modelled at PV+T level
- Timing affecting behaviour of peripherals simulated to maintain the state of the platform
- Runs ~1/10 the speed of ultra fast software development mode (~3 MIPS)
- L1 cache in synch
- L2 cache in synch
- Interconnect not exercised
- Direct memory accesses from the processor
Simulation: Timing Accurate Mode

- Runs at Cycle Accurate speed (~30 Kips)
- Very accurate processor with pipelining and multiple accesses in the same cycle

Timing inside peripherals modelled at PV+T level
- Lump Latencies returned by peripherals are then serialized by the transactors
- All I and D accesses through cache
- Capability to switch off specific interconnects to increase speed
Software Performance Analysis – Key Solution Components

• Accuracy on demand capabilities, through support of multiple simulation modes for SystemC TLM
  – Processors
  – Memory subsystem
  – Frequently accessed peripherals

• Software performance measurement capabilities
Software performance measurement

• Multiple Analysis views
  – Software timing profile (Hot spots in code, MIPS achieved)
  – Memory Profiling, Function Profiling, Interrupts Profiling etc.

• Capturing latency at multiple probe points in the platform
  – Peripheral, L2 Cache, Processor Bus Interface Unit, Processor pipeline shift stage

• Measuring cache performance
  – hit/miss ratio
  – Replacement policy

• Tracing state transitions in the peripherals to optimize transaction sequences
Analysis Views - Examples

• Function Call Graph
  - Graphical representation of function call and return sequence
  - Helps identifying algorithmic hotspots and accelerate code organization optimizations
Analysis Views – Examples continued.

- Memory Access Profile

<table>
<thead>
<tr>
<th>Phys. addr</th>
<th>count</th>
<th>cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>801bc658:</td>
<td>1</td>
<td>47</td>
</tr>
<tr>
<td>80ffd9e8:</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>80ffd9ec:</td>
<td>1</td>
<td>38</td>
</tr>
<tr>
<td>a019b64c:</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a019b650:</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>a019b654:</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a019b658:</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a01d3f08:</td>
<td>1</td>
<td>63</td>
</tr>
<tr>
<td>a01d3f0c:</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a01d3f10:</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>a01d3f14:</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a01d3f18:</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a01d3f1c:</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

- Tabular representation of memory access delays for a specific software use case
- Helps identifying memory bottlenecks and provides insight into the performance of the memory sub-system hierarchy
Analysis Views – Examples continued.

- Function Profiling

Instructions executed: 16
Cycles executed: 454
Execution time (msec): 0.001135

<table>
<thead>
<tr>
<th>%Instr</th>
<th>Time(msec)</th>
<th>#Occur</th>
<th>Instr</th>
<th>Cycles</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.250000</td>
<td>0.000347</td>
<td>1</td>
<td>5</td>
<td>139</td>
<td>os_init</td>
</tr>
<tr>
<td>31.250000</td>
<td>0.000490</td>
<td>1</td>
<td>5</td>
<td>196</td>
<td>pu_main_set_run_level</td>
</tr>
<tr>
<td>12.500000</td>
<td>0.000115</td>
<td>0</td>
<td>2</td>
<td>46</td>
<td>pu_main</td>
</tr>
<tr>
<td>25.000000</td>
<td>0.000183</td>
<td>0</td>
<td>4</td>
<td>73</td>
<td>suIntEnableInt</td>
</tr>
</tbody>
</table>

- Tabular representation of function profiling
- Helps identifying algorithmic hotspots and accelerate peep-hole optimizations
Analysis Views – Examples

• Function X Memory Profile

<table>
<thead>
<tr>
<th>#Accesses</th>
<th>#Cycles</th>
<th>MinCycles</th>
<th>MaxCycles</th>
<th>Address</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>80ff9e98</td>
<td>os_init</td>
</tr>
<tr>
<td>1</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>80ff9ec</td>
<td>os_init</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>a019d64c</td>
<td>os_init</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>a019d650</td>
<td>os_init</td>
</tr>
<tr>
<td>1</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>a01d3f08</td>
<td>os_init</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>a01d3f0c</td>
<td>os_init</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>a01d3f10</td>
<td>os_init</td>
</tr>
<tr>
<td>1</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td>a04a0160</td>
<td>os_init</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>a04a0168</td>
<td>os_init</td>
</tr>
<tr>
<td>1</td>
<td>47</td>
<td>47</td>
<td>47</td>
<td>801bc658</td>
<td>pu_main_set_run_level</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>a019b654</td>
<td>pu_main_set_run_level</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>a01d3f14</td>
<td>pu_main_set_run_level</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>a01d3f18</td>
<td>pu_main_set_run_level</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>a01d3f1c</td>
<td>pu_main_set_run_level</td>
</tr>
<tr>
<td>1</td>
<td>62</td>
<td>62</td>
<td>62</td>
<td>a01d3f20</td>
<td>pu_main_set_run_level</td>
</tr>
<tr>
<td>1</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>a01d3fa4</td>
<td>pu_main_set_run_level</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>a019b658</td>
<td>suIntEnableInt</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>a04a0170</td>
<td>suIntEnableInt</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>a04a0178</td>
<td>suIntEnableInt</td>
</tr>
<tr>
<td>1</td>
<td>62</td>
<td>62</td>
<td>62</td>
<td>a0440180</td>
<td>suIntEnableInt</td>
</tr>
</tbody>
</table>

• Tabular representation of memory access delays per function for a specific software use-case
• Helps identifying algorithmic bottlenecks caused by memory delays
Results

• User level test cases
  – 18 test cases, size varying from 78 ms to 830 ms of real time

  **Accuracy between 99% to 91% in all cases except for two where the accuracy was 85% and 79%**

• Engineering level small test cases
  – 9 test cases, size varying from few to few hundred instructions

  **Accuracy always on par with specification**

3G Mobile Phone Virtual Platform

• 55 unique models (95 instances)
  → 4 models within memory sub-system enabled with PV+T

• Runs the actual, unmodified software for the phone
  → PV+T enables SW performance analysis during OS boot, network registration, …
Thank You!

- Q&A