Profiling Virtual Prototypes: Simulation Performance Analysis & Optimization

Sandeep Jain, NXP, Noida

Agenda

- Motivation
- Requirements
- Modeling Library
- Wall-clock time measurement
- Profiler Design
- Results
- Conclusion

Motivation

- Virtual Prototyping is a proven methodology for early HW architectural exploration, performance analysis and software development
- Lots of guidelines available for modeling techniques to improve simulation performance of Virtual Prototypes
  - Minimize Context switching (minimize usage of SC_THREAD)
  - Make as fewer copies as possible (use pointers instead of data copies)
  - Use higher level of abstraction (model only what is required)
  - Use JIT-ISS instead of interpretive-ISS
  - Prefer DMI over blocking transport
- There is lack of profiling tools geared towards VP

Motivation

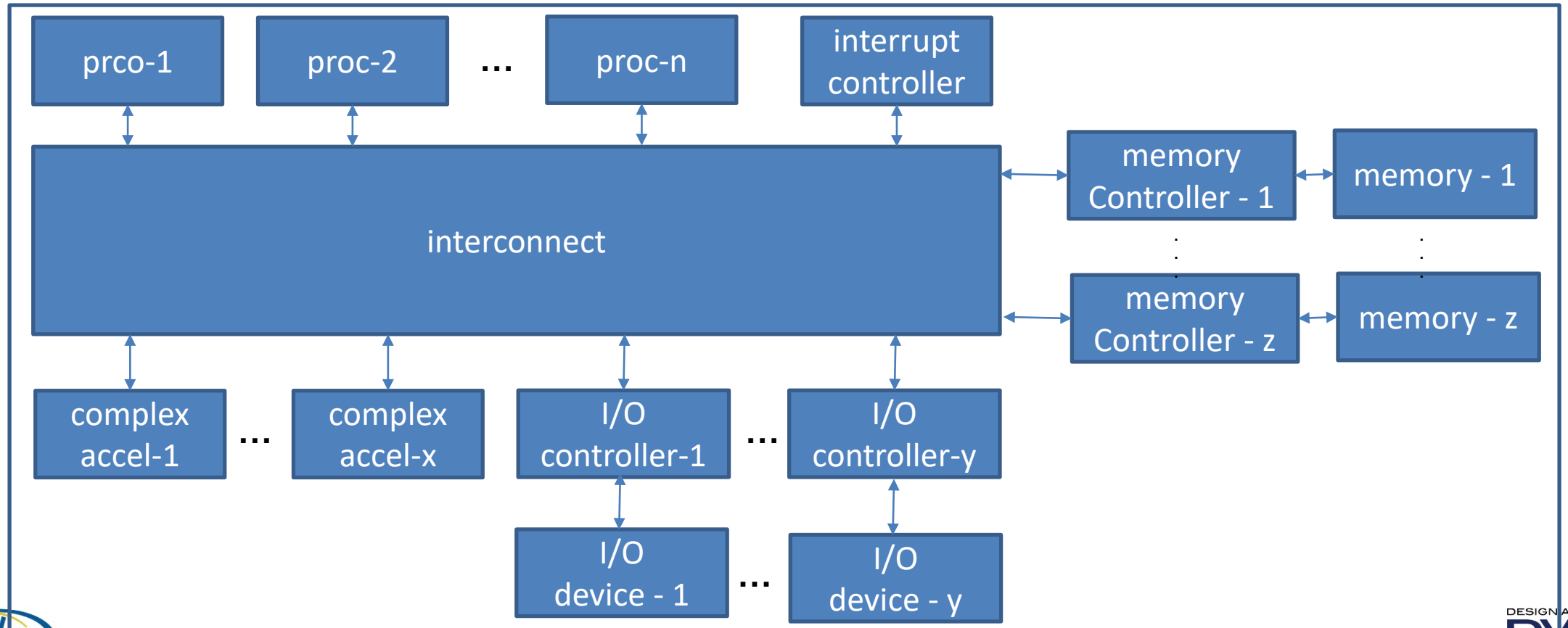
- Most off-the-shelf profilers like gprof, oprofile, perf, etc. are primarily designed to profile code for analysis of software function calls
  - These profilers lack visibility into the design hierarchy of the Virtual Prototype
  - For any method used to implement a common function across different component models, these profilers can't distinguish how much time it takes in context of these different models
- For profiling VP, a desirable format is one based on design hierarchy of system being modeled

Requirements

- Simulation execution time consumed by various component models as per the design hierarchy modeled by VP
- Negligible overhead
  - Enabling the profiler should have minimal impact of overall simulation performance
- Non intrusive
  - No change required to the component models to enable profiling
- No re-compilation to enable profiler

Background

- VP comprises of many TLM of processors, accelerators, interconnects, IO peripheral and memory controllers and associated device and memory models



Modeling Library

- Transaction Level models (System C, C, C++)
- System C kernel or custom simulation kernel in C/C++
- In-house Modeling library to expose simple C/C++ APIs to model developers
  - Hide most of the simulation kernel and interface semantics and associated complexities
  - Uniform coding style across all models
- ***Execution Control*** vs ***Non-Execution Control*** component models
  - Execution Control component models are provided simulation time by the kernel
  - Co-operative multi-threading
  - Non-Execution Control component models execute in context of Execution Control models in a blocking fashion

Wall-clock Time Measurement

- Requirements
  - Negligible overhead
    - Use Timestamp Counter (TSC) register of the underlying processor of the host machine
    - Avoid relying on system-call provided by host operating system
  - High resolution
    - Able to measure miniscule changes in time
- boost::posix_time or std::chrono found to be good candidates
- Profiling Framework is independent of the library chosen
  - ***get_timestamp***() is the API used by profiler which can be implemented using the library of choice

Profiler Design

- Execution Control components code snippet without profiler

```
void do_timeslice(..) { // declared as SC_METHOD
…
do_stuff(..); // implemented by individual component models
…
next_trigger(time_elapsed, sc_core::SC_NS); // dynamic sensitivity (optional)
}
```

- Execution Control components code snippet with profiler

```
void do_timeslice(..) {
…
if (profiling_enabled) now =  get_timestamp(); // library call to get current wall clock time stamp
do_stuff(..);
if (profiling_enabled) profile.bucket[idx] += (get_timestamp() – now);
…
next_trigger(time_elapsed, sc_core::SC_NS);
}
```

- Separate buckets in profiler for maintaining execution time consumed by each Execution Control
- Since Non-Execution Control component models work in context of Execution Control models, any overhead by these get accumulated into the corresponding Execution control component model

Profiler Design

- Component model interface code snippet without profiler

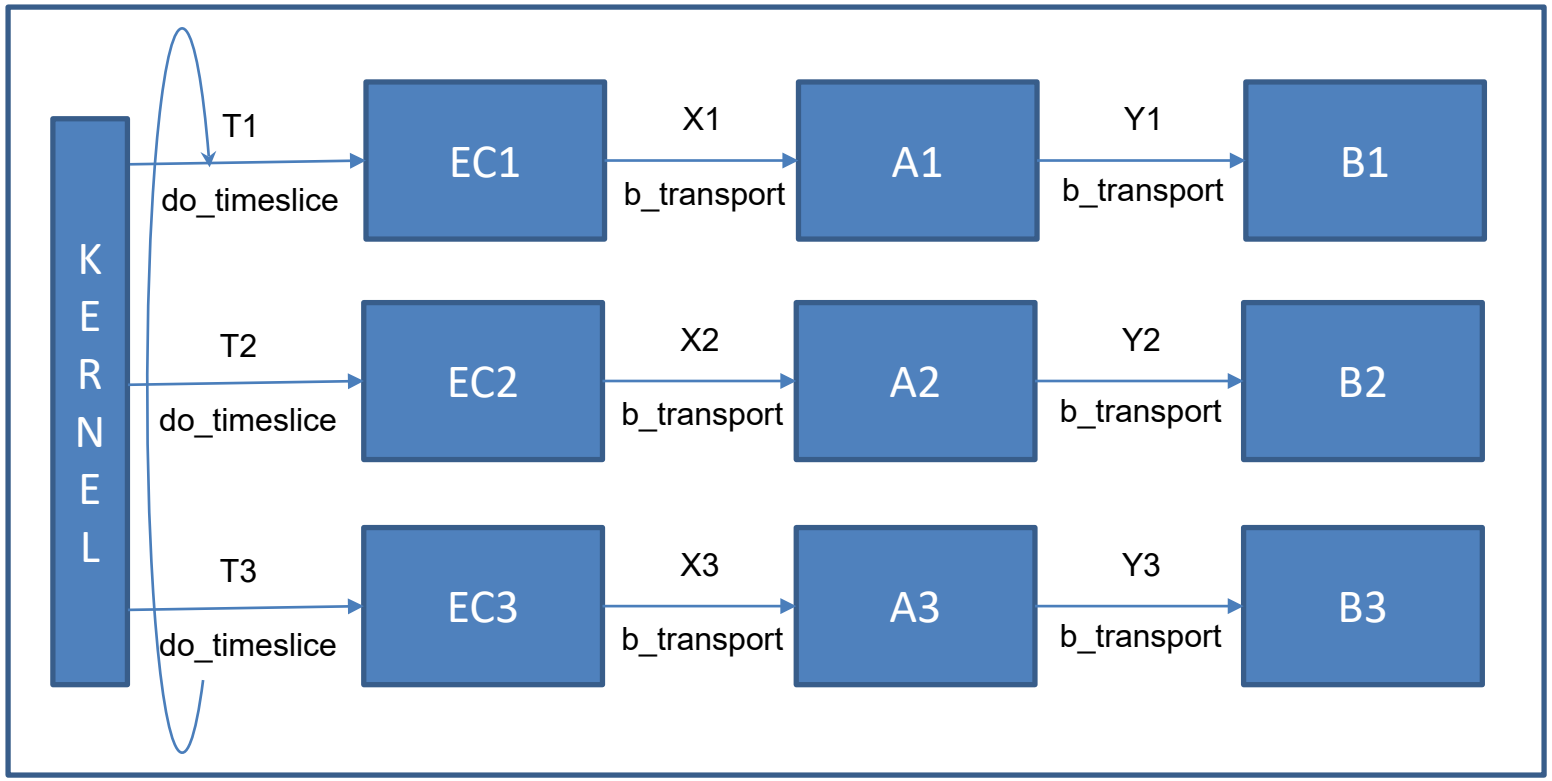
```
void do_interface_access(..) { // called from do_stuff
…
socket->b_transport(*trans, delay); // implemented by target component model
…
}
```

- Component model interface code snippet with profiler

```
void do_interface_access(..) {
…
if (profiling_enabled) now = get_timestamp();
socket->b_transport(*trans, delay);
if (profiling_enabled) profile.bucket_ext[jdx] += (get_timestamp() – now);
…
}
```

- Separate buckets in profiler for maintaining execution time consumed outside of each component model
- Difference between the ***profile.bucket*** and ***profile.bucket_ext*** provides the wall clock time spent during execution of the component itself

Profiler Design

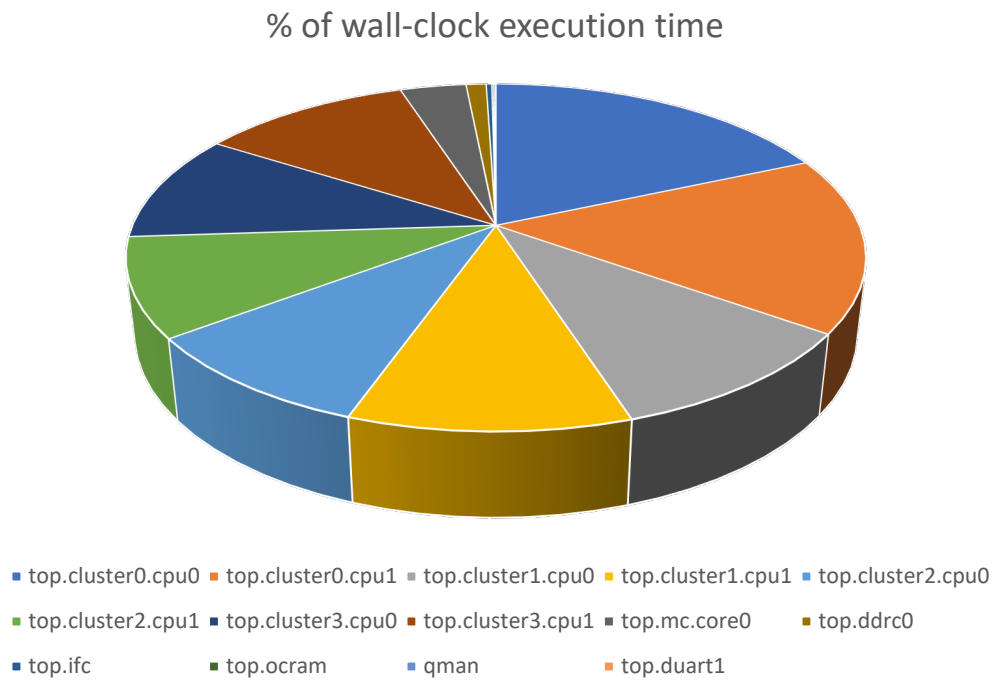


EC1 execution time: T1 - X1
EC2 execution time: T2 - X2
EC3 execution time: T3 – X3
A1 execution time: X1 - Y1
A2 execution time: X2 - Y2
A3 execution time: X3 – Y3
B1 execution time: Y1
B2 execution time: Y2
B3 execution time: Y3
…

Results

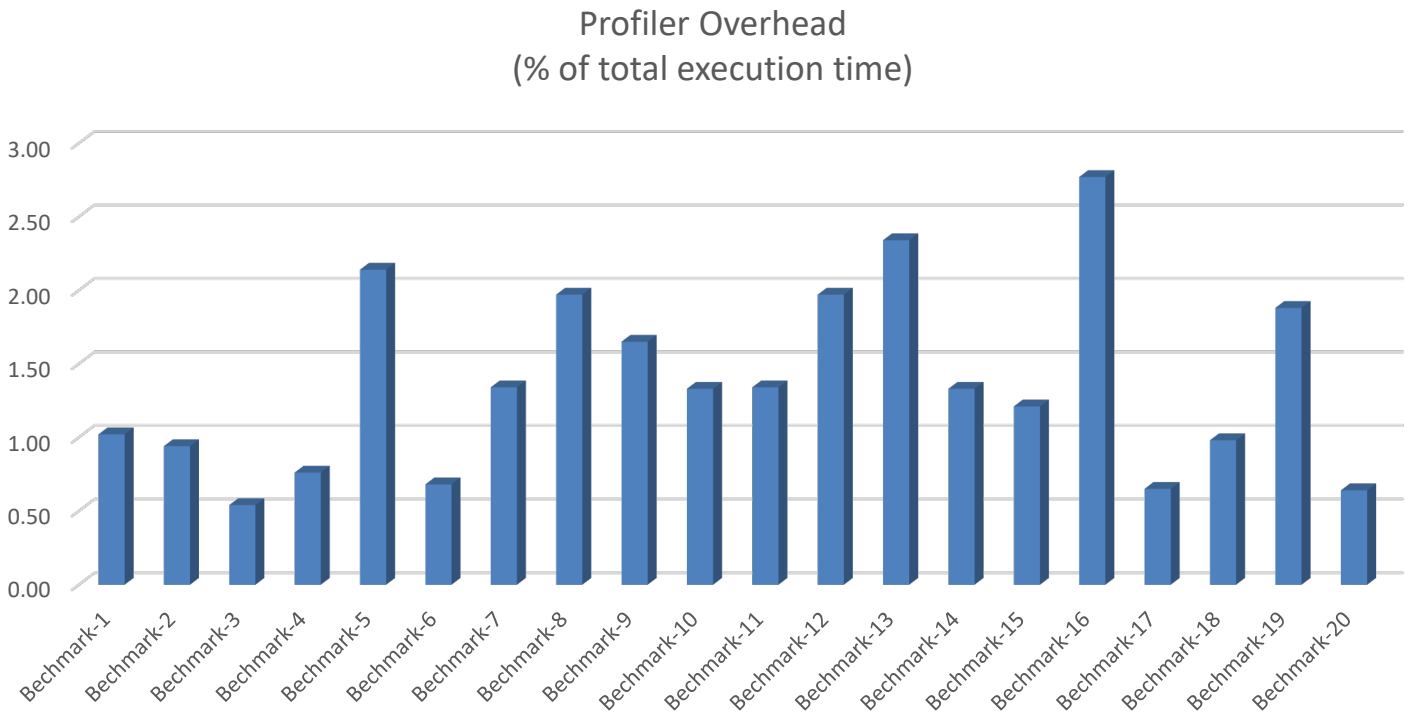
- Following is a sample report from the profiler for a workload with 8-cores

Component	% of wall-clock execution time
top.cluster0.cpu0	18.33
top.cluster0.cpu1	16.32
top.cluster1.cpu0	10.13
top.cluster1.cpu1	10.36
top.cluster2.cpu0	9.24
top.cluster2.cpu1	9.17
top.cluster3.cpu0	9.99
top.cluster3.cpu1	10.95
top.mc.core0	3.43
top.ddrc0	1.04
top.ifc	0.29
top.ocram	0.10
qman	0.10
top.duart1	0.02



Results

- Following chart shows the profiler overhead over a range of 20 benchmarks comprising of SMP Linux boot, IP-Fwd, IP-Sec, EEMBC, SPEC
- Average overhead was less than 1.5% with approx. 2.8% for worst case



Conclusion

- VP profiler was applied to several VP for various SOCs for early software bring-up
- VP profiler provided simulation execution time consumed by various component models as per the design hierarchy modeled by VP with negligible overhead on simulation performance
- VP profiler didn’t require any changes in individual component models
  - It was fully implemented within the modeling library abstraction layer
- VP profiler can be enabled at run-time to measure impact of different software applications and different VP configuration options on simulation performance of different component models
- VP profiler helped uncover several performance issues in simulation models as well as optimal configuration options to be achieve best simulation performance

Questions