COTSon: Infrastructure for system-level simulation

Ayose Falcón, Paolo Faraboschi, Daniel Ortega
HP Labs — Exascale Computing Lab

http://sites.google.com/site/hplabscotson

MICRO-41 tutorial
November 9, 2008
Lake Como, Italy
Core Concepts

- Functional Simulator (SimNow)
  - Sequences the behavioral simulation of CPUs and devices
- Timers
  - Using functional events, it computes the target metrics (time, power)
- Sampler
  - Decide when to turn on or off the Timers and for how long
- Interleaver
  - Decides how to buffer and reorder functional events (SMP)
- Time Predictor
  - Based on timer metrics evolution over time, decides how to feed the information back to the functional simulator
Decoupling Simulation

- **Functional Simulation (fast)**
  - Emulates the behavior of all the components of our system
    - Disks, video, network cards, etc.
  - Necessary to verify correctness, run software

- **Timing Simulation (slow)**
  - Models the timing of all the components
  - Used to measure performance (or power)

- **COTSOn approach:**
  "Functional Directed with sampling and time feedback"
Timers (a.k.a. CPU/device models)

- Accept instructions, process them and update metrics
- All timers share the memory hierarchy
- Some “must have” metrics: cycles and instructions
- Pluggable architecture
- Not only CPU models, but also:
  - Profiling
  - Trace generation
  - “Simpoint”-like analysis

Current models
- Timer0: simple “linear” model + cache hierarchy
- Timer1: Timer0 + in-order pipeline
- Bandwidth: Only limited by memory bandwidth
- PTLSim (open source): linked to COTSon, full x86 → OoO superscalar
Samplers

- Decide when and how much to simulate and when to move from one simulation state to another
  - **Functional**: fast forward to the next state as quickly as possible
  - **Warming (simple/detailed)**: get data in *stateful* structures (e.g., caches), but do not account for time
  - **Simulation**: account for time

- Pluggable architecture

- Many implementations
  - Smarts\(^1\), SimPoint\(^2\), Dynamic Sampling\(^3\), Random, Interval-based, ...

  \(^1\) Wunderlich et al. SMARTS: Accelerating Microarchitecture Simulation Via Rigorous Statistical Sampling, ISCA’03
  \(^2\) B. Calder. Simpoint (www.cse.ucsd.edu/~calder/simpoint)
  \(^3\) A. Falcón et al. Combining Simulation and Virtualization through Dynamic Sampling, ISPASS’07

- Samplers are what provide the **major acceleration component**
  - Even for very accurate (hence slow) timing models, a good sampler only needs to invoke the timing model < 1% of the time.
Single CPU simulation

• Fast and accurate single node simulation using Dynamic Sampling
  – Detect dynamically program phase changes
  – The challenge is to avoid disturbing the VM execution in the code cache during fast functional emulation
  – Phase changes are correlated with VM statistics (exceptions, I/O events, code cache invalidations, ...) which are easy to get and don’t impact performance
Dynamic Sampling

A. Falcón, P. Faraboschi, and D. Ortega, “Combining Simulation and Virtualization through Dynamic Sampling”, in Proceedings of ISPASS’07

- Allows users to favor accuracy or speed, depending on their requirements
  - **High accuracy**: 0.4% accuracy error with 8.5x speedup
  - **High speed**: 309x speedup with 1.9% error

- Fully dynamic
  - Does not require any a priori analysis
  - Automatically detect code phases

- Allows for providing timing feedback to the functional simulator
Multi-core simulation

- SimNow performs functional simulation of multi-cores
  - It simulates MP as “sequential interleaved” at coarse granularity
  - This misses fine grain memory interactions
- COTSon buffers events and delivers them interleaved to the CPU timing models
- **Problem**: Hard to scale up → OS? BIOS?
Interleaving Fundamentals

MP functional simulation runs “sequentially interleaved” at coarse granularity.

This may miss fine-grain memory interactions

We buffer events at every MP quantum and deliver them interleaved to the timers

Interleaved based on the CPUs IPC
Timing Feedback

- **Problem**: feed back timing information to the functional emulator
  - Give the simulated application an illusion of approximate time (functional time corresponding to simulated time)
- **Define the IPC of a quantum based on previous history**
  - "Classic" time-series prediction problem, with unknown model
- **Current model**: simple predictor
  - The IPC is fed back to the functional simulator
  - The application being simulated acts as if execution is faster or slower

![Diagram showing the process of timing feedback](image-url)
Many-core simulation


- Translate SW thread-level into simulated core-level parallelism
  - Identify and separate the instruction streams of the different threads at the OS level (context switches)
  - Dynamically map each instruction flow to the corresponding core of the target multicore architecture, taking into account application-level thread synchronization
Multi-node simulation

- Simulate a computer cluster as a cluster of full-system simulators
  - Each node of the cluster is simulated with a full-system simulator
  - Network simulator used to simulate network topology

- Problems:
  - Time skew between nodes needs to be controlled with quanta
  - Quantum size must be chosen carefully
    - Small quanta \( \rightarrow \) Bad simulation speed
    - Large quanta \( \rightarrow \) Bad simulation accuracy
Adaptive Synchronization

A. Falcón, P. Faraboschi, and D. Ortega, “An Adaptive Synchronization Technique for Parallel Simulation of Networked Clusters”, in Procs. of ISPASS’08

- Basic idea: dynamically adjust the quantum for maximum speed at a controlled accuracy loss
  - Quantum increases/decreases depending on packet traffic
  - Slow Acceleration, fast deceleration ("driving over speed bumps")
Speed vs. Accuracy Tradeoffs

• We can play the speed vs. accuracy game at several control points
  – Within a node: dynamic sampling sensitivity
  – At cluster level: adaptive quantum range
• By choosing the appropriate values we can reach
  – Single node accuracy in the order of 11%–15% error (simple CPU model)
  – Networking accuracy (microbenchmark) up to 15 Gb/s
  – All of the above with self-relative slowdown (vs. native) of ~15x-30x
• Improvement Areas
  – SMP and cluster validation on larger applications
  – Better CPU models (if needed), especially in the SMP coherency area
  – Distributed simulation sometimes “unstable” for large clusters (> 50 nodes)
  – “Canned recipes” for non-expert users for accuracy/speed requirements
Success stories

- Fault isolation for commodity architectures study
  - “Configurable isolation: building high-availability systems with commodity multi-core processors” (ISCA’07)
  - “Isolation in Commodity Multicore Processors” (IEEE MICRO’07)

- Nanophotonics architecture investigation
  - “Corona: System implications of emerging nanophotonic technology” (ISCA’08)

- Last level cache technologies study (CACTI-D)
  - “A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies” (ISCA’08)

- Web 2.0 workload analysis
  - “Microblades and megaservers: system architectures for emerging Web 2.0 / internet workloads” (ISCA’08)

- …and some other internal projects at HP Labs
Putting it all together

Acc. IPC over time of 800 nodes running NAMD

IPC

Network traffic
COTSon Labs – Experiments

1. Functional simulation
2. Simple timers
   - dump_to
   - in_order
3. Memory tracer
4. Timing feedback
5. Samplers
   - Random sampling
   - Dynamic sampling
6. Selective tracing
7. Network simulation
8. Disk simulation
Functional simulation (I)

cotson-node

Lua file

Lua command

Lua file

cotson-node

Lua file

Lua command

Lua file
Functional simulation (II)

• How to start a (deterministic) simulation
  – Send keystrokes to SimNow
  – xtools \rightarrow using SimNow hacks
  – Network access
  – Pre-started application
Simple timer: "dump_to"

- Use COTSon SDK to create your own timing or sampling module
- Experiment:
  - Instructions from SimNow are disassembled and dumped to a file
  - No time feedback
  - Output fields (disasm)

<table>
<thead>
<tr>
<th>pid</th>
<th>tid</th>
<th>cr3</th>
<th>PC</th>
<th>(length)</th>
<th>Opcodes</th>
<th>disasm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[load</td>
<td>store]</td>
<td>virtual @</td>
</tr>
</tbody>
</table>
Simple timer: “in-order”

- 3-stage in-order pipeline + cache stalls
- Memory hierarchy in Lua
Memory tracer

- Transparent memory
  - Dump to file/display
Timing feedback

With timing feedback

CPU 1

CPU 2

0.4

0.8

1.2

1.6

2.0

0 500 1000 1 500 2000

0 0.2 0.4 0.6 0.8 1.0 IPC

26 7 November 2008
Timing feedback

Without timing feedback

CPU 1

CPU 2

0 200 400 600 800 1000 1 200 1 400 1 600 2000

0 0.2 0.4 0.6 0.8 1

IPC

0 2 4 6 8 10 12 14 16 18 20

time

27 7 November 2008
Random sampling

- Sampling states
  - **Functional**: pre-program IPC
  - **Simple Warming**: warm caches and branch predictor
  - **Detailed Warming**: simple warming + warm reorder buffer
  - **Simulation**: sample, full timing
Dynamic sampling (I)
Dynamic sampling (II)
Selective Tracing

- Lets user determine which application(s) or part(s) of an application running inside SimNow is simulated with timing
- Combined with CR3 tracing, allows the user to skip instructions from OS or other applications
  - Change in CR3 register = context switch
- Uses SimNow tagging of instructions to communicate data between guest OS and COTSOn
  - Via a reserved CPUID instruction

Ex: application instrumentation

```c
#include "cotson-tracer.h"
int main(void)
{
    COTSON_BEGIN_TRACE (1)
    [benchmark code]
    COTSON_END_TRACE (1)
}
```

Ex: OS instrumentation

```
$> cotson_tracer.sh begin 1
$> benchmark1
$> $> cotson_tracer.sh end 1
$> $> ...
$> $> cotson_tracer.sh begin 2
$> benchmark2
$> $> cotson_tracer.sh end 2
```
Network simulation

- 4-node cluster, 1 CPU per node
  - NAS benchmarks with mpich2 MPI library
  - Node discovery, MPI boot and five NAS benchmarks (cg, ep, is, lu, mg) with 8 threads
- Simple crossbar switch, 2Gb/s bandwidth
- 1 Gb/s NICs
- Adaptive quantum synchronization 10:1000
Disk simulation

- Disksim integrated into COTSOn
  [http://www.pdl.cmu.edu/DiskSim](http://www.pdl.cmu.edu/DiskSim)
- Experiment
  - No CPU timing $\Rightarrow$ IPC=1
  - Disk model
    - Seagate Cheetah 4LP $\Rightarrow$ 4.5 GB 10,033 rpm