PERFORMANCE ANALYSIS FOR MODERN SERVER CPUS

KANISHKA LAHIRI (KANISHKA.LAHIRI@AMD.COM)
PRINCIPAL MEMBER OF TECHNICAL STAFF, SERVER PERFORMANCE
AGENDA

- Performance analysis and computer architecture
- Performance modeling methodology
- Server workload and CPU challenges
PERFORMANCE ANALYSIS TECHNIQUES

- **RTL**
  - 1-10 Hz

- **Cycle-Accurate Models**
  - 1-10 KHz

- **Mixed Abstraction Models**
  - 10-100 Mhz

- **Trace Analyzers**
  - 10-100 Mhz

- **Fast Functional Models**

- **Hardware performance counters & directed tests**
  - M/C Ghz
ROLE OF PERFORMANCE MODEL / ANALYSIS

- **Model reflects a configurable arch theme**
- **100s of small (<1%) u-architectural tradeoffs**
- **Competitive analysis, Simulated/measured data**
- **Compare Perf model/RTL Find/fix perf bugs in both Validate Perf model**
- **Perf Correlation / Verif**
- **Perf Projections**
- **Feature Evaluation**
- **Architectural Development**
- **Compare Perf model/HW Diagnose Perf problems Evaluate fixes Eval Perf/Power tradeoffs**
- **Silicon Performance Validation and Debug**

**Project Timeline**

**“Accuracy” or Detail Level or Cost**
CYCLE-ACCURATE PERFORMANCE MODELS

- C++ model with higher-level of abstraction than RTL
  - 100K lines of uarch specific code
  - 400K lines of shared infrastructure and library code
  - Highly parameterized at both the macro and micro level
  - Many, many configuration switches for structures, queues, algorithms, policies
  - Output: Hundreds of counters and statistics covering the uarch
  - Don’t model everything in the simulator (exceptions, power states, many rare conditions).

- Accuracy goal: match RTL (realistically 1-2%)
- Speed goal: as fast as possible (realistically ~10 Khz)
- Workhorse simulator for microarchitecture exploration/dev/correlation
  - Limited mT capabilities
SERVER WORKLOADS TO DRIVE PERFORMANCE MODELING

- **SPEC CPU2006/2017**
  - Workhorse CPU throughput/speed benchmark
  - Gcc / optimizing compilers

- **Enterprise (Classic)**
  - SPECJbb15 (Java)
  - Traditional Data bases (TPC-C/TPC-E)

- **Cloud**
  - Spark / Hadoop
  - NoSQL databases
  - Machine learning

- **High-performance computing**
  - DGEMM (matrix multiply aka HPL), FFT
  - LS-Dyna3D, Ansys, ...

- **Virtualization**
  - SPECVirtSC 2013
  - VMmark (from VMware)

- **Microbenchmarks**
  - Latency, bandwidth

---

A great model is useless without proper workloads to drive it. **Representativeness is key**
REAL WORKLOAD SIMULATION & SAMPLING

- Cycle accurate simulators run at ~10 Khz
- Simulating SPEC CPU 2006 in entirety (44T dynamic insts) would take 100+ yrs

Real instruction stream (unmodified app on real OS)

~100M inst/sample

PM PM PM PM

Statsfiles

Average

Final IPC + 100s of stats
Leverage fast functional simulators
- Virtual platform of a PC/server
- Boot unmodified OS, run applications
- Examples: QEMU, AMD SimNow

Hardware based tracing tools
- Needs proprietary custom hardware

Dynamic binary instrumentation tools
- Popular for program analysis
- Injects profiling code into running binary
- User space only
- Examples: DynamicRIO, Valgrind
CHOOSING SAMPLE POINTS

- Exploit phase behavior in programs
  - Profile program stats over fixed intervals (e.g. BBVs)
  - Run clustering
  - Choose one trace / input per cluster

- **Dramatically** reduces the no. of inst simulated (<0.5% for SPECINT)

- Widely used in academia and industry (with variants)

- Sample points:
  - Fixed length traces
  - State snapshots

Source: Sherwood et al, ASPLOS02
Non-determinism in server workloads

- Need environment where SW adapts to performance changes
- IPC doesn’t make sense any more. Need to estimate Wall Clock Time

Source: SynchroTrace: ISPASS 2015
TRACE DRIVEN V EXECUTION DRIVEN MODELS

TDM
- Inst Traces → Cycle accurate model → IPC, Uarch stats

EDM
- M/C State Snapshot → SimNow (functional model)
  - Dynamic i-stream
  - Performance feedback
  - Cycle-accurate model → Ex. Time, scores
## TDM VS EDM TRADE OFFS

<table>
<thead>
<tr>
<th>Trace Driven</th>
<th>Execution Driven</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
<td><strong>Pros:</strong></td>
</tr>
<tr>
<td>• <em>Faster</em></td>
<td>• <em>Wrong path</em></td>
</tr>
<tr>
<td>• <em>Deterministic i-stream</em></td>
<td>• <em>More realistic i-stream</em></td>
</tr>
<tr>
<td>• <em>Easier to sample</em></td>
<td></td>
</tr>
<tr>
<td>• <em>Easier to validate</em></td>
<td></td>
</tr>
<tr>
<td><strong>Cons:</strong></td>
<td><strong>Cons:</strong></td>
</tr>
<tr>
<td>• <em>No wrong path instructions</em></td>
<td>• <em>IPC is not a good metric, need WCT</em></td>
</tr>
<tr>
<td>• <em>Deterministic i-stream (doesn’t model mT effects)</em></td>
<td>• <em>Sampling less understood</em></td>
</tr>
<tr>
<td></td>
<td>• <em>Complexity</em></td>
</tr>
<tr>
<td></td>
<td>• <em>Speed</em></td>
</tr>
<tr>
<td></td>
<td>• <em>Validation</em></td>
</tr>
</tbody>
</table>
Dynamic sampling techniques can help (Falcon et al, ISPASS 2007)

**Step 1:**
- Run functional simulation, use low overhead profiling to detect phase changes (E.g. BBV profiling, Online clustering)

**Step 2:**
- Reuse perf from previously seen phases

**Step 3:**
- Take a sample on a new phase (engage timing model, run EDM simulation)

**Drawback:** serialized simulation
- Run time, Health
POTENTIAL SOLUTION – 2: SYNCHRO TRACE (ISPASS 2015)

- Encodes dependency and synchronization information into the trace
- Sacrifices details (exact instructions)
- Replies on trapping on calls to specific sync primitive (e.g. pthreads)
- Limited to user space synchronization (limitation of tracing infra)
MODERN SERVER CPU EXAMPLE
AMD EPYC™ 7601

- Each CCX is a 4C/8T complex
  - Each core has private L2
  - Each CCX has a shared 8MB L3

- Each die is a 8C/16T SCM
  - 16MB of L3
  - 2 DDR channels

- Each socket is 4 dies:
  - 32C/64T
  - 64MB L3
  - 8 DDR channels

- Cycle accurate simulation simply does not scale
SIMULATING COMPLETE SERVER SOCS

- Need to model loaded latency

- Running 64 threads in the perf model is not practical
SIMULATING COMPLETE SERVER SOCS

- Use lower thread counts
- Mix abstraction models
- Bus trace driven models
SIMULATING LARGE CACHES: SOLUTIONS

Why not simply warm up caches for longer?
- Instruction trace lengths are limited
- Storage is a challenge
- The problem of legacy traces

Research directions:
- Check point cache state (very accurate, but uarch dependent) [Lauterbach et al, SUN, 1993]
- Stitch (take previous sample’s state) [Kessler, IEEE Trans Computers, Iss 43]
- Combine stitch with fraction of new sample [Conte 1996]
- Use reuse distance distros to estimate hit/miss rates [WarmSim/CacheSim, ISPASS 2014,16]
- Use an estimated miss rate to statistically warm up the cache
SUMMARY & FUTURE AREAS OF RESEARCH

.performance analysis is critical
- Variety of tools
- C++ based modeling is key
- 10% inspiration, 90% perspiration

cycle accurate CPU models are critical
- Model all the details and more of a uarch
- Stimulate with proper workloads
- Aggressively downsample

server CPU challenges:
- Modeling non-determinism
- High thread counts
- Warming up large caches

Near term challenges for performance modeling for server
- Ever more aggressive sampling
- Warm up techniques
- Execution driven challenges
- Synchronized traces (ISPASS 2016)

Longer term challenges
- Model/predict cluster level performance
- Low utilization workloads
- Exploit multi core architectures
DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2018 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.