THE SNIPER MULTI-CORE SIMULATOR

13:30  INTRODUCTION
14:00  INTERVAL SIMULATION
14:45  SIMULATOR INTERNALS
15:15  – COFFEE BREAK –
15:45  VALIDATION RESULTS
16:00  RUNNING SIMULATIONS AND PROCESSING RESULTS
16:30  DEMO
17:00  – END –

HTTP://WWW.SNIPERSIM.ORG
SATURDAY, JUNE 9TH, 2012
ISCA 2012, PORTLAND, OR
• Collaboration between Intel, imec and 5 Flemish universities
• Study Space Weather as an HPC workload
THE SNIPER MULTI-CORE SIMULATOR

INTRODUCTION

Wim Heirman, Trevor E. Carlson,
Ibrahim Hur and Lieven Eeckhout

HTTP://WWW.SNIPERSIM.ORG
SATURDAY, JUNE 9TH, 2012
ISCA 2012, PORTLAND, OR
TRENDS IN PROCESSOR DESIGN: CACHE

- Cache sizes are increasing
TRENDS IN PROCESSOR DESIGN: CORES

• Number of cores per node is increasing
  – 2001: Dual-core POWER4
  – 2005: Dual-core AMD Opteron
  – 2011: 10-core Intel Xeon Westmere-EX
  – 201x: Intel MIC Knights Corner (50+ cores)
• Design tomorrow’s processor using today’s hardware

• Simulation
  – Obtain performance characteristics for new architectures
  – Architectural exploration
  – Early software optimization
DEMANDS ON SIMULATION ARE INCREASING

- Increasing core counts
  - Linear increase in simulator workload
  - Single-threaded simulator sees a rising gap
    - workload: increasing target cores
    - available processing power: near-constant single-thread performance of host machine
  - Need to use all cores of the host machine
- Parallel simulation
DEMANDS ON SIMULATION ARE INCREASING

• Increasing cache size
  – Need a large working set to fully exercise a large cache
  – Scaled-down applications won’t exhibit the same behavior
  – Long-running simulations are required
UPCOMING CHALLENGES

• Future systems will be diverse
  – Varying processor speeds
  – Varying failure rates for different components
  – Homogeneous applications become heterogeneous

• Software and hardware solutions are needed to solve these challenges
  – Handle heterogeneity (reactive load balancing)
  – Be fault tolerant
  – Improve power efficiency at the algorithmic level (extreme data locality)

• Hard to model accurately with analytical models
# Needed Detail Depends on Focus

<table>
<thead>
<tr>
<th>Component</th>
<th>Single-event time scale</th>
<th>Required sim time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTL</td>
<td>single clock cycle</td>
<td>millions of cycles</td>
</tr>
<tr>
<td>OOO execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core memory ops</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1 cache access</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLC access</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Off-socket</td>
<td>microseconds</td>
<td>seconds</td>
</tr>
</tbody>
</table>

- **Too slow**
  - cycle-accurate models
  - interval core model
- **Not accurate enough**
  - simple core models
• Out-of-order core performance model with in-order simulation speed

INTERVAL SIMULATION

D. Genbrugge et al., HPCA’10
S. Eyerman et al., ACM TOCS, May 2009
T. Karkhanis and J. E. Smith, ISCA’04, ISCA’07
Cycle Stacks

- Where did my cycles go?
- CPI stack: cycles per instruction, broken up in components
- Normalize by either
  - Number of instructions (CPI stack)
  - Execution time (time stack)
- Different from miss rates as cycle stacks directly quantify the effect on performance
• Scaling to more cores, larger input set size

• How does execution time scale, and why?

Rodinia - SRAD

Per cent of time

8c large 8c small 16c large 16c small

- sync-barrier
- sync-crit_sect
- mem-dram
- mem-off_socket
- mem-l3
- mem-l2_neighbor
- mem-l2
- mem-l1_neighbor
- mem-l1d
- ifetch
- branch
- depend-fp
- depend-int
- dispatch_width
Fast and Accurate Simulation is Needed

• Sniper Simulator
  – Interval core model
  – Accurate structures (caches, branch predictors, etc.)
  – Parallel simulator scales with the number of simulated cores

• Key Questions
  – What is the right level of abstraction?
  – When to use these abstraction models?
MANY ARCHITECTURE OPTIONS
SIMULATION IN SNIPER

Execution-driven simulation

- memory hierarchy simulator
- branch predictor simulator

Processor cores

functional simulator (Pin)

A single-process, multithreaded workload (v1.06)

Trace-driven simulation

- Multiple, single-threaded workloads (v2.0)
**TOP SNIPER FEATURES**

- Interval Model
- CPI Stacks
- Parallel Multithreaded Simulator
- Based on Graphite infrastructure
- x86-64 and SSE2 support
- Validated against Core2, Nehalem
- Full DVFS support
- Shared and private caches
- Modern branch predictor
- Supports pthreads and OpenMP, TBB and OpenCL
- SimAPI and Python interfaces to the simulator
- Many flavors of Linux supported (Redhat, Ubuntu, etc.)
## Simulator Comparison

<table>
<thead>
<tr>
<th></th>
<th>Sniper</th>
<th>Graphite</th>
<th>Gem5</th>
<th>COTSon</th>
<th>MARSSx86</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrated</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Func-directed</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>User-level</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full-system</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Archs Supported</td>
<td>x64</td>
<td>x64</td>
<td>x64</td>
<td>x64</td>
<td>x64</td>
</tr>
<tr>
<td>Alpha SPARC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel (in-node)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shared caches</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
SNIPER LIMITATIONS

• User-level
  – Perfect for HPC
  – Not the best match for workloads with significant OS involvement

• Functional-directed
  – No simulation / cache accesses along false paths

• High-abstraction core model
  – Not suited to model all effects of core-level changes
  – Perfect for memory subsystem or NoC work

• x86 only
SNIPER HISTORY

• July 2010: Branched from MIT Graphite
• November, 2011: SC’11 paper, first public release
• March 2012, version 2.0: Multi-program workloads
• May 2012, version 3.0: Heterogeneous architectures
• Today: 150+ downloads from 25+ countries
THE SNIPER MULTI-CORE SIMULATOR
INTERVAL SIMULATION

TREVOR E. CARLSON, WIM HEIRMAN,
IBRAHIM HUR AND LIEVEN ECKHOUT

HTTP://WWW.SNIPERSIM.ORG
SATURDAY, JUNE 9TH, 2012
ISCA 2012, PORTLAND, OR
OVERVIEW

• Simulation Methodologies
  – Trace, Integrated, Functional-directed

• Core Models
  – One-IPC
  – Interval

• Interval Model and Simulation Detail

• CPI-Stacks
SIMULATION METHODOLOGIES

• Trace-based Simulation
  – No wrong-path instructions nor timing-influenced results
  – Not the best for multithreaded applications

• Functional-First Simulation
  – The timing model controls wrong-path execution via checkpoints
  – Can be difficult to build

• Integrated Simulation
  – Timing and functional simulation are closely tied together
  – Timing of the core drives when instructions are fetched and executed

• Functional-Directed Simulation
  – Mispredicted path instructions are not taken into account
    • Rolling-back /check-pointing is therefore not needed
  – Timing model tends to be separate from the functional model
**Needed Detail Depends on Focus**

<table>
<thead>
<tr>
<th>Component</th>
<th>Single-event time scale</th>
<th>Required sim time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTL</td>
<td>single clock cycle</td>
<td>millions of cycles</td>
</tr>
<tr>
<td>OOO execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core memory ops</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1 cache access</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLC access</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Off-socket</td>
<td>microseconds</td>
<td>seconds</td>
</tr>
</tbody>
</table>

- Too slow
- Cycle-accurate models
- Simple core models
- Interval core model

- Not accurate enough
One-IPC Modeling – Too Simple?

• Simple high-abstraction model
• Our definition of a One-IPC core model
  – Scalar, in-order issue
  – Account for non-unit instruction exec latencies
  – Perfect branch prediction
  – L1 D-cache hits are completely hidden
  – All other cache accesses incur penalty
One-IPC Core Model

• Alternative for memory access traces
  – Aims to provide more-realistic access patterns
  – Allows for timing feedback

• Nevertheless, One-IPC core models do not exhibit MLP
  – Therefore, request rates are not as accurate as cycle-level simulators
• Out-of-order core performance model with in-order simulation speed

D. Genbrugge et al., HPCA’10
S. Eyerman et al., ACM TOCS, May 2009
T. Karkhanis and J. E. Smith, ISCA’04, ISCA’07
**Detailed Model vs. Interval Sim**

- **Interval Simulation**
- **Functional Simulator**
- **DRAM**
- **Cache Hierarchy**
- **LSQ**
- **Execution Units**
- **Commit**
- **ROB**
- **Issue Queue**
- **Decode**
- **Fetch**
- **BP**
- **I$**

Diagram showing the flow fromFetch, through Decode, to Issue Queue, LSQ, Execution Units, Commit, ROB, and finally to Interval Simulation and Functional Simulator.
Key Benefits of the Interval Model

- Models superscalar OOO execution
- Models impact of ILP
- Models second-order effects: MLP

- Allows for constructing CPI stacks
MULTI-CORE INTERVAL SIMULATION

- memory hierarchy simulator
- branch predictor simulator
- functional simulator

processor cores

old window

window

next instruction to dispatch

head
tail
head
tail

dispatched instructions
upcoming instructions
Instantaneous dispatch rate is determined by the longest critical path in the old window:

\[
\text{Instantaneous dispatch rate} = \min \left( \frac{W}{L}, D \right)
\]

Little’s law
Assumes a balanced architecture

\[ L = \text{longest critical path length in cycles} \]
\[ W = \text{instructions in the old window (max = ROB length)} \]
\[ D = \text{maximum dispatch rate (processor width)} \]
LONG BACK-END MISS EVENTS

ISOLATED LONG-LATENCY LOAD

S. Eyerman et al., ACM TOCS, May 2009
LONG BACK-END MISS EVENTS
OVERLAPPING LONG-LATENCY LOADS

S. Eyerman et al., ACM TOCS, May 2009
If long-latency load (LLC miss):
  core sim time += miss latency

AND walk the window to issue independent miss events: these are hidden under the long-latency load
  – second-order effects

AND empty old window
I-CACHE MISS
(L1, L2, TLB)

S. Eyerman et al., ACM TOCS, May 2009
If I-cache or I-TLB miss:
  core sim time += miss latency
  AND empty old window
BRANCH MISPREDICTION

S. Eyerman et al., ACM TOCS, May 2009
If branch misprediction:

- `core sim time += branch resolution time`
- `+ front-end pipeline depth`
Branch resolution time = longest critical path in ‘old window’ leading to the branch
If serializing instruction:

core sim time += window drain time

window drain time = max ( W / D , L )

AND empty the old window
Cycle stacks

• Where did my cycles go?
• CPI stack
  – Cycles per instruction
  – Broken up in components
• Normalize by either
  – Number of instructions (CPI stack)
  – Execution time (time stack)
• Different from miss rates: cycle stacks directly quantify the effect on performance
CONSTRUCTING CPI STACKS

• Interval simulation: track why time is advanced
  – No miss events
    • Issue instructions at base CPI
    • Increment base component
  – Miss event
    • Fast-forward time by $X$ cycles
    • Increment component by $X$
By thread: heterogeneous behavior in a homogeneous application?
Using Cycle Stacks to Explain Scaling Behavior
**Using Cycle Stacks to Explain Scaling Behavior**

- Scale input: application becomes DRAM bound
Using Cycle Stacks to Explain Scaling Behavior

- Scale input: application becomes DRAM bound
- Scale core count: sync losses increase to 20%
THE SNIPER MULTI-CORE SIMULATOR Simulator Internals

WIM HEIRMAN, TREVOR E. CARLSON, IBRAHIM HUR AND LIEVEN EEEKHOUT

HTTP://WWW.SNIPERSIM.ORG
SATURDAY, JUNE 9TH, 2012
ISCA 2012, PORTLAND, OR
Overview

- Parallel simulation with relaxed synchronization
  - Flexible synchronization schemes between cores
  - Trade off causality errors for simulation speed
- Parallelism inside Sniper
- Hardware components
Relaxed Synchronization

- Graphite introduced relaxed synchronization with a number of different synchronization schemes
  - none: only synchronizes when the application does; for pthread calls, etc.
  - random-pairs: synchronizes random pairs of threads
  - barrier: synchronizes all threads at a given simulated time interval
- Sniper defaults to barrier synchronization with 100ns intervals
  - Multi-machine mode not supported, so tight synchronization is easier
BARRIER SYNCHRONIZATION IN ACTION

real time

simulated time

pthread_cond_wait

pthread_cond_signal

barrier
PARALLELISM INSIDE SNIPER

• Each simulated core is run inside its own thread
  – Includes functional simulation, timing models for core and cache
  – Each core model maintains its own local time

• Extra threads for network and DRAM models
  – Can process invalidation requests without interrupting the core model

• Each thread is allowed to independently make progress
  – Causality errors can occur, no rollback
  – Skew is limited to 100ns
THREADS IN SNIPER

- Application threads
- Network threads

- Cores
- L1 I/D
- L2
- L3
- DRAM
- NI
• Each memory access instantly returns latency
• Application threads maintain time
• Network threads reset time for each request
MODELING CONTENTION

• Events may happen out of order
• How to model bandwidth / contention?
  – History list
    • Resource in use at times 0…10, 12…17, 25…30
    • Access at 15: delay = 2
    • Access at 8, length 5: ?
• Causality errors are possible
  – Effect is limited, as long as average bandwidth is OK
  – Allows for faster simulation, easier implementation
  – Speed versus accuracy trade-off
CONFIGURABLE COMPONENTS

• Hardware options
  – Branch predictors
  – Cache hierarchies
    • Shared, private
    • Optional prefetcher

• Core options
  – Core models: interval, one-IPC, Graphite legacy
  – DVFS, heterogeneous

• Networks
Branch Predictor

- Pentium-M-style branch predictor  
  V. Uzelac, ISPASS’09
PARAMETRIC SHARED CACHE HIERARCHY
THE SNIPER MULTI-CORE SIMULATOR
SIMULATOR ACCURACY
AND HARDWARE VALIDATION

IBRAHIM HUR, TREVOR E. CARLSON,
WIM HEIRMAN AND LIEVEN ECKHOUT

HTTP://WWW.SNIPERSIM.ORG
SATURDAY, JUNE 9th, 2012
ISCA 2012, PORTLAND, OR
HARDWARE VALIDATION

• Why validation?
  – Debugging
  – Verifying modeling assumptions
  – Balance between accuracy and generality
    • e.g.: loop buffer in Nehalem/Westmere;
      uop-cache in Sandy Bridge

• Current status:
  – Validated against Core2 (internal, results @ SC’11)
  – Nehalem ongoing (public version)
EXPERIMENTAL SETUP

• Benchmarks
  – Complete SPLASH-2 suite
    • 1 to 16 threads
    • Linux pthreads API
  – Extensive use of microbenchmarks to tune parameters and track down problems

• Hardware
  – Four-socket Intel Xeon X7460 machine
  – Core2 (45nm, Penryn) with 6 cores/socket
EXPERIMENTAL SETUP: ARCHITECTURE

Diagram showing a multi-level cache hierarchy with L1, L2, and L3 levels, connected to DRAM.
HINTS FOR COMPARING TO HARDWARE

• Threads are pinned to their own core
  
  pthread_setaffinity_np()

• Steepstep is disabled

  
  echo performance > /sys/devices/system/cpu/*/cpufreq/scaling_governor

• Turbo mode, Hyperthreading disabled
  
  – BIOS setting

• Use hardware performance counters
  
  – But can be difficult to interpret
  – Overlapping cache misses (HW) vs. hits (Sniper)
The interval core model provides consistent accuracy of 25% avg. abs. error, with a minimal slowdown.
Good accuracy for the entire benchmark suite
**INTERVAL: BETTER RELATIVE ACCURACY**

- Application scalability is affected by memory bandwidth.
- Interval model provides more realistic memory request streams, which results in a more accurate scaling prediction.
APPLICATION OPTIMIZATION

- Splash2-Raytrace shows very bad scaling behavior
- CPI stack shows why: heavy lock contention
- Conversion to use locked increment instruction helps
Sniper currently scales to 2 MIPS

Typical simulators run at 10s-100s KIPS, without scaling
Variability due to relaxed synchronization is application specific.
FLEXIBILITY TO CHOOSE NEEDED FIDELITY
MANY-CORE SIMULATIONS

High simulation speed up to 1000 simulated cores

- Pin limitation (to be lifted shortly) at 1020 cores
- Efficient simulation: L1-based benchmarks execute faster
- Host system: dual-socket Xeon X5660 (6-core Westmere), 96 GB RAM
VALIDATING FOR NEHALEM

[Bar charts showing HW Measurement to Sniper Result comparison for various test cases.]

79
THE SNIPER MULTI-CORE SIMULATOR
RUNNING SIMULATIONS
AND PROCESSING RESULTS

WIM HEIRMAN, TREVOR E. CARLSON,
IBRAHIM HUR AND LIEVEN ECKHOUT

HTTP://WWW.SNIPERSIM.ORG
SATURDAY, JUNE 9TH, 2012
ISCA 2012, PORTLAND, OR
OVERVIEW

• Obtain and compile Sniper
• Running
• Configuration
• Simulation results
• Interacting with the simulation
  – SimAPI: application
  – Python scripting
**RUNNING SNIPER**

- **Download Sniper**
  - [http://snipersim.org/w/Download](http://snipersim.org/w/Download)
    - Download tar.gz
    - Git clone
    ```bash
    ~/sniper$ export GRAPHITE_ROOT=$(pwd)
    ~/sniper$ make
    ```

- **Running an application**
  ```bash
  ~/sniper$ ./run-sniper -- /bin/true
  ~/sniper/test/fft$ make run
  ```
RUNNING SNIPER

• Integrated benchmarks distribution
  – [http://snipersim.org/w/Download_Benchmarks](http://snipersim.org/w/Download_Benchmarks)
  ~/benchmarks$ export BENCHMARKS_ROOT=$(pwd)
  ~/benchmarks$ make
  ~/benchmarks$ ./run-sniper -p splash2-fft \ 
    -i small -n 4

• Standardizes input sets and command lines
• Includes SPLASH-2, PARSEC
INTEGRATION WITH BENCHMARKS

• To add a new benchmark
  – Add source code
  – Add __init__.py file
    • Provides application invocation details
    • Define input sets (e.g.: test, small, large)
  – Mark the ROI region
  – Simple example: see local/pi
**Multi-programmed Workloads**

- **Recording traces (SIFT format)**
  
  ```
  $ ./record-trace -o fft -- test/fft/fft -p1
  ```

- **Limited trace, by instruction count:**
  - Fast-forward (-f), detailed length (-d), block size (-b)
  
  ```
  $ ./record-trace -o fft -f 1e9 -d 1e9 -b 1e8
  -- test/fft/fft -p1 -m20
  ```

- **Running traces**
  
  ```
  $ ./run-sniper -c gainestown -n 4
  --traces=gcc.sift,swim.sift,swim.sift,equake.sift
  ```
**REGION OF INTEREST**

- Skip benchmark initialization and cleanup
- Mark code with ROI begin / end markers
  - SimRoiStart() / SimRoiEnd() in your own application
  - $ ./run-sniper --roi -- test/fft/fft
- Already done in benchmarks distribution
  - benchmarks/run-sniper implies --roi
  - Use --no-roi to override
- Cache warming during pre-ROI period
  - Use --no-cache-warming to override
**CONFIGURATION**

- Stackable configuration files (run-sniper -c) and explicit command-line options (-g)
  - Template configurations in sniper/config/*.cfg (-c name)
  - Your own local configuration files (-c filename.cfg)
  - Explicit option: -g --section/key=value

- Multiple configuration files, and -g options, can be combined
  - Config files specified later on the command line take precedence
  - config/base.cfg is always included
  - If no -c option is provided, config/gainestown.cfg is the default (quad-core Nehalem-based Xeon)

- Complete configuration is stored in sim.cfg after each run
Example configuration: largecache.cfg

```ini
[perf_model/l3_cache]
cache_size = 16384  # KB
```

$ run-sniper -c gainestown -c largecache.cfg

Equivalent to:

$ run-sniper -c gainestown \
   -g --perfmmodel/l3_cache/cache_size=16384
SIMULATION RESULTS

• Files created after each simulation:
  – sim.cfg: all configuration options used for this run (includes defaults, all -c and -g options)
  – sim.out: basic statistics (number of cycles, instructions per core, cache access and miss rates, ...)
  – sim.stats: complete set of all recorded statistics at key points in the simulation (start, roi-begin, roi-end, stop)

• Use the sniper_lib Python package for parsing
**SIMULATION RESULTS**

sniper_lib.get_results() parses sim.cfg, sim.stats and returns configuration and statistics (roi-end – roi-begin) for all cores

```
~/sniper/tools$ python
> import sniper_lib
> results = sniper_lib.get_results(resultsdir = '..', ')
> print results
{‘config’: {‘general/total_cores’: ‘64’,
   ‘perf_model/core/frequency’: ‘2.66’, ...},
‘results’: {‘performance_model.instruction_count’:[123],
   ‘performance_model.elapsed_time’: [23000000], ...}}
```
SIMULATION RESULTS

• Let’s compute the IPC for core 0
• Core frequency is variable (DVFS)
  so cycle count has to be computed
  – Time is in femtoseconds, frequency in GHz

```python
> instrs = results['results']['performance_model.instruction_count'][0]
> cycles = results['results']['performance_model.elapsed_time'][0]
  * float(results['config']['perf_model/core/frequency'])
  * 1e-6  # femtoseconds -> nanoseconds
> ipc = instrs / cycles
2.0
```
**SIMULATION RESULTS**

- CPI stacks (user of sniper_lib)

```bash
$ ./tools/cpistack.py [--time|--cpi|--abstime]
```

<table>
<thead>
<tr>
<th>Core 0</th>
<th>CPI</th>
<th>CPI %</th>
<th>Time %</th>
</tr>
</thead>
<tbody>
<tr>
<td>depend-int</td>
<td>0.20</td>
<td>23.42%</td>
<td>23.42%</td>
</tr>
<tr>
<td>depend-fp</td>
<td>0.16</td>
<td>18.94%</td>
<td>18.94%</td>
</tr>
<tr>
<td>branch</td>
<td>0.12</td>
<td>14.04%</td>
<td>14.04%</td>
</tr>
<tr>
<td>ifetch</td>
<td>0.04</td>
<td>4.16%</td>
<td>4.16%</td>
</tr>
<tr>
<td>mem-l1d</td>
<td>0.21</td>
<td>24.41%</td>
<td>24.41%</td>
</tr>
<tr>
<td>mem-l3</td>
<td>0.02</td>
<td>2.72%</td>
<td>2.72%</td>
</tr>
<tr>
<td>mem-dram</td>
<td>0.05</td>
<td>5.73%</td>
<td>5.73%</td>
</tr>
<tr>
<td>sync-mutex</td>
<td>0.02</td>
<td>2.59%</td>
<td>2.59%</td>
</tr>
<tr>
<td>sync-cond</td>
<td>0.03</td>
<td>3.01%</td>
<td>3.01%</td>
</tr>
<tr>
<td>other</td>
<td>0.01</td>
<td>0.97%</td>
<td>0.97%</td>
</tr>
<tr>
<td>total</td>
<td>0.84</td>
<td>100.00%</td>
<td>0.00s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core 1</th>
<th>CPI</th>
<th>CPI %</th>
<th>Time %</th>
</tr>
</thead>
<tbody>
<tr>
<td>depend-int</td>
<td>0.20</td>
<td>23.92%</td>
<td>23.92%</td>
</tr>
<tr>
<td>depend-fp</td>
<td>0.16</td>
<td>18.79%</td>
<td>18.79%</td>
</tr>
<tr>
<td>branch</td>
<td>0.12</td>
<td>13.72%</td>
<td>13.72%</td>
</tr>
<tr>
<td>mem-l1d</td>
<td>0.20</td>
<td>24.06%</td>
<td>24.06%</td>
</tr>
<tr>
<td>mem-l3</td>
<td>0.06</td>
<td>6.79%</td>
<td>6.79%</td>
</tr>
<tr>
<td>mem-dram</td>
<td>0.04</td>
<td>5.22%</td>
<td>5.22%</td>
</tr>
<tr>
<td>sync-mutex</td>
<td>0.05</td>
<td>5.60%</td>
<td>5.60%</td>
</tr>
<tr>
<td>sync-cond</td>
<td>0.02</td>
<td>1.89%</td>
<td>1.89%</td>
</tr>
<tr>
<td>other</td>
<td>0.01</td>
<td>0.97%</td>
<td>0.97%</td>
</tr>
<tr>
<td>total</td>
<td>0.85</td>
<td>100.00%</td>
<td>0.00s</td>
</tr>
</tbody>
</table>
SIMAPI IMPLEMENTATION

• Magic instructions allow the application to talk to the simulator directly

```c
__asm__ __volatile__ (  
    "xchg %%bx, %%bx\n"
    : "=a" (_res)  /* output */
    : "a" (_cmd),
     "b" (_arg0),
     "c" (_arg1)  /* input */
    );  /* clobbered */
```

• Pin intercepts this instruction and passes control to the simulator

• Command and arguments passed through rax/rbx/rcx registers, result in rax
APPLICATION SIMAPI

• Calling simulator API functions from your C program

```c
#include <sim_api.h>

– SimInSimulator()
  • Return 1 when running inside Sniper, 0 when running natively

– SimGetProcId()
  • Return processor number of caller

– SimRoiStart() / SimRoiEnd()
  • Start/end detailed mode (when using ./run-sniper --roi)

– SimSetFreqMHz(proc, mhz) / SimGetFreqMHz(proc)
  • Set / get processor frequency (integer, in MHz)

– SimUser(cmd, arg)
  • User-defined function
```
**Python Scripting**

- Scripts are run on simulator startup
  - Register hooks: callbacks when certain events happen during the simulation
  - See common/system/hooks_manager.h for all available hooks

- Use an existing script from sniper/scripts/*.py:
  .\run-sniper -s scriptname

- Or your own script:
  .\run-sniper -s myscriptname.py

- Use sim package for convenience wrappers
• Low-level script
• Execute “foo” at each barrier synchronization

import sim_hooks
def foo(t):
    print ‘The time is now’, t
sim_hooks.register(sim_hooks.HOOK_PERIODIC, foo)
• Higher-level script

• Execute “foo” at each barrier synchronization

```python
import sim
class Class:
    def hook_periodic(self, t):
        print ‘The time is now’, t
sim.util.register(Class())
```
• High-level script: execute “foo” every X ms
• Pass in parameter using
  ./run-sniper -s myscript.py:X

```python
import sim
class Class:
    def setup(self, args):
        sim.util.Every(long(args)*sim.util.Time.MS, self.periodic)
    def periodic(self, t, t_delta):
        print 'The time is now', t
        print 'Elapsed time since last call', t_delta
sim.util.register(Class())
```

- Access configuration, statistics, DVFS
- Live periodic IPC trace:
  - See scripts/ipctrace.py for a more complete example

```python
class IPCTracer:
    def setup(self, args):
        sim.util.Every(1*sim.util.Time.US, self.periodic)
        self.instrs_prev = 0
    def periodic(self, t, t_delta):
        freq = sim.dvfs.get_frequency(0)
        cycles = t_delta * freq * 1e-9  # fs * MHz -> cycles
        instrs = long(sim.stats.get('performance_model', 0, 'instruction_count'))
        print 'IPC =', (instrs - self.instrs_prev) / cycles
        self.instrs_prev = instrs
```
• Communicate information between application and Python script
  – E.g.: simulated hardware performance counters

• Application:
  
  ```
  uint64_t ninstrs = SimUtil(0xdeadbeef, SimGetProcId())
  ```

• Python script:
  
  ```
  class PerfCtr:
    def setup(self):
      sim.util.register_command(0xdeadbeef, self.compute)
    def compute(self, arg):
      return sim.stats.get('performance_model', arg,
                           'instruction_count')
  ```
Near Term Ideas

• Multiple processes
  – Multiple multi-threaded applications, MPI support

• Heterogeneous cores at run-time
  – Big: 4-issue processor
  – Small: 2-issue processor
  – Now supported in Sniper v3.0

• Scheduling/Migration support

• Power modeling (McPAT)

• Multiple processor configurations
  – Currently the simulator is compiled to support a single type of processor (Core2 vs. Nehalem vs. Sandy Bridge)
THE SNIPER MULTI-CORE SIMULATOR HANDS-ON DEMO

WIM HEIRMAN, TREVOR E. CARLSON,
IBRAHIM HUR AND LIEVEN EECKHOUT

HTTP://WWW.SNIPERSIM.ORG
SATURDAY, JUNE 9TH, 2012
ISCA 2012, PORTLAND, OR
SNIPER DEMO

• Downloading
• Compiling
• Running a demo application
• Evaluating Performance
  – CPI Stacks
• Configuration and Run-time Modifications
  – Configuration files
  – Python scripting
  – ROI markers and Magic instructions
REFERENCES

• Sniper website
  – http://snipersim.org/

• Download
  – http://snipersim.org/w/Download
  – http://snipersim.org/w/Download_Benchmarks

• Getting started
  – http://snipersim.org/w/Getting_Started

• Questions?
  – http://groups.google.com/group/snipersim
  – http://snipersim.org/w/Frequently_Asked_Questions