

SimFlex and ProtoFlex: Fast, Accurate, and Flexible Simulation of Computer Systems

Eric Chung
Mike Ferdman
Nikos Hardavellas

Anastasia Ailamaki
Babak Falsafi
James Hoe

Shelley Chen
Brian Gold
Jangwoo Kim
Ippokratis Pandis
Michael Papamichael

Minglong Shao
Jared Smolens
Stephen Somogyi
Evangelos Vlachos
Tom Wenisch
Roland Wunderlich



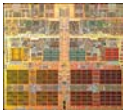
Computer Architecture Lab at
CarnegieMellon

25 Oct 2008, PACT

CALCM Computer Architecture Lab CarnegieMellon

Simulation speed challenges

- Longer benchmarks
 - SPEC 2006: *Trillions* of instructions per benchmark
- Slower simulators
 - Full-system simulation: 1000× slower than SimpleScalar
- Multiprocessor, multicore systems
 - CMP: 2× cores every processor generation




1,000,000× slowdown vs. HW → years per experiment

2

CALCM Computer Architecture Lab CarnegieMellon

Our solution: Statistical sampling

- Measure uniform or random locations



- Impact
 - Sampling: ~10,000× reduction in turnaround time
 - Independent measurements: 100- to 1000-way parallelism
 - Confidence intervals: **quantifiable** result reliability
- Challenges
 - Rapidly create warm μ arch state prior to measurements
 - Allow independent simulation of each measurement
 - Sample non-deterministic, highly variable MP applications

3

CALCM Computer Architecture Lab CarnegieMellon

Tutorial overview

- Simulation sampling
 - Background & theory
 - Best practices
 - Multiprocessor/multicore sampling
- Flexus full-system simulation framework
 - Simulator internals
 - Developing with Flexus
- ProtoFlex: FPGA-accelerated full-system MP simulation

*Best practices for
simulation-based multiprocessor research*

4

CALCM Computer Architecture Lab Carnegie Mellon



Tutorial outline

- Introduction (15 min)
- Simulation sampling (1 hour 15min)
- Developing with Flexus (1 hour)
- ProtoFlex (30 min)

5

CALCM Computer Architecture Lab Carnegie Mellon

Tutorial materials

- Hand-outs
 - Copy of this tutorial's slides
 - 'Getting Started with Flexus' quick reference 
- www.ece.cmu.edu/~simflex
 - Flexus downloads
 - Publications, tech reports
 - Discussion/Q&A mailing list 

6

CALCM Computer Architecture Lab Carnegie Mellon

Terminology

- SimFlex simulation infrastructure
 - SMARTS: sampling methodology
 - Flexus: full-system simulation framework
 - TraceFlex, UniFlex, CMPFlex, etc. simulators
 - Different HW models, trade speed for accuracy
- ProtoFlex
 - FPGA-accelerated full-system MP simulation

7

CALCM Computer Architecture Lab Carnegie Mellon

Terminology: simulation modes


- Fast-forward simulation (*Simics*) ~15 MIPS
 - Only architecturally visible state (e.g., mem, regs)
- Functional simulation (e.g., *TraceFlex*) ~0.5 MIPS
 - No timing, simulates march components (e.g., cache)
- In-order timing (e.g., *CMPFlex*) ~10 kIPS
 - Simulates all components w/ timing, in-order core
- Out-of-order timing (e.g., *CMPFlex.OoO*) ~3 kIPS
 - Simulates all components w/ timing, out-of-order core
- FPGA-accelerated functional simulation (*ProtoFlex*) ~10 - 50 MIPS
 - Same as functional in software, but 10^x - 100^x faster

8


CALCM Computer Architecture Lab Carnegie Mellon

Sampling Microarchitecture Simulation

- Many small measurements at uniform/random locations
 - SPEC CPU2000 avg. benchmark in 7 hrs vs. 5.5 days



- Checkpoints: store state of components (e.g., cache)
 - Enable parallel simulation & online results
 - SPEC CPU2000 avg. benchmark in 91 seconds



9

CALCM Computer Architecture Lab Carnegie Mellon

Flexus framework

- Full-system simulation of unmodified commercial apps
 - In-order timing on any Simics target, OoO timing for SPARC v9
- Component-based design
 - Easy composition of complex system models
- Uniprocessor, CMP, DSM hardware system models
 - Multiple timing models trade accuracy for speed
- Designed-in support for simulation sampling
 - Supports checkpointing, statistic aggregation & confidence calculations
- Reduced turnaround time with FPGAs (*ProtoFlex*)
 - Accelerated checkpoint generation

10

CALCM Computer Architecture Lab Carnegie Mellon


Tutorial outline

- Introduction
- **Simulation sampling**
- Developing with Flexus
- ProtoFlex

Simulation Sampling Theory & Practice

SimFlex Tutorial – Section 2 of 4

Nikos Hardavellas



Computer Architecture Lab at
Carnegie Mellon

25 Oct 2008, PACT

CALCM Computer Architecture Lab Carnegie Mellon

Current simulation practices

- Subset or scaled version of benchmark suite
- Single unit of ~1 billion instructions
- Selected measurements via profiling

Results not representative of workload performance, slow

13

CALCM Computer Architecture Lab Carnegie Mellon

Uniprocessor simulation sampling

- Measure many units of few instructions
 - Representative results with minimal simulation
 - No profiling for uniform sampling: immune to many types of error
- Functional warming – update state between units
 - Enables accurate measurement of small units
- Checkpointing
 - Enables more speed, parallelism, and online results

Accurate and Fast
0.6% CPI error, 5000x speedup over complete simulation on SPEC2000

14

CALCM Computer Architecture Lab Carnegie Mellon

Section 2 outline

- Simulation sampling
 - Sampling in theory
 - Sampling in practice: accurate measurements
- Multiprocessor sampling
 - Extending sampling to MP applications
 - Sampling optimizations

15

CALCM Computer Architecture Lab Carnegie Mellon

Sampling theory

Estimate the mean of a population property X — to a desired confidence — by measuring X over a sample whose size n is minimized.

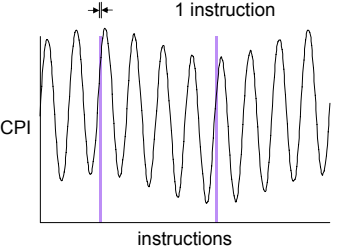
- Arbitrary distribution
- Confidence
 - e.g., 99.7% probability of $\pm 3\%$ error
- $n = f(\text{C.V.}, \text{confidence})$

16

CALCM Computer Architecture Lab Carnegie Mellon

Sampling for simulation

Defining the sampling population



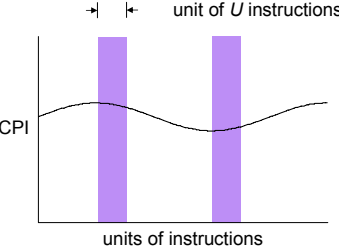
- CPI difficult to measure over 1 instruction
- Instead, define as units of U instructions
- As U changes, so does:
 - Observed C.V. of CPI
 - Required sample size n

17

CALCM Computer Architecture Lab Carnegie Mellon

Sampling for simulation

Defining the sampling population



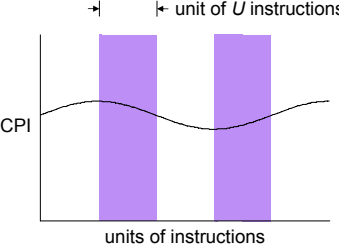
- CPI difficult to measure over 1 instruction
- Instead, define as units of U instructions
- As U changes, so does:
 - Observed C.V. of CPI
 - Required sample size n

18

CALCM Computer Architecture Lab Carnegie Mellon

Sampling for simulation

Defining the sampling population

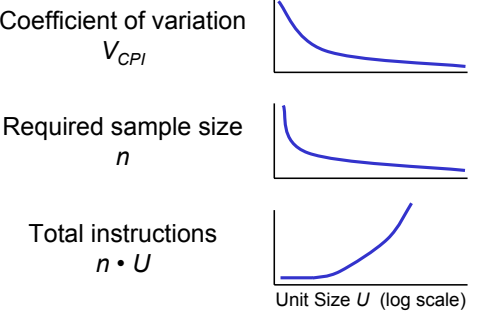


- CPI difficult to measure over 1 instruction
- Instead, define as units of U instructions
- As U changes, so does:
 - Observed C.V. of CPI
 - Required sample size n

19

CALCM Computer Architecture Lab Carnegie Mellon

Minimizing total instructions



Coefficient of variation V_{CPI}

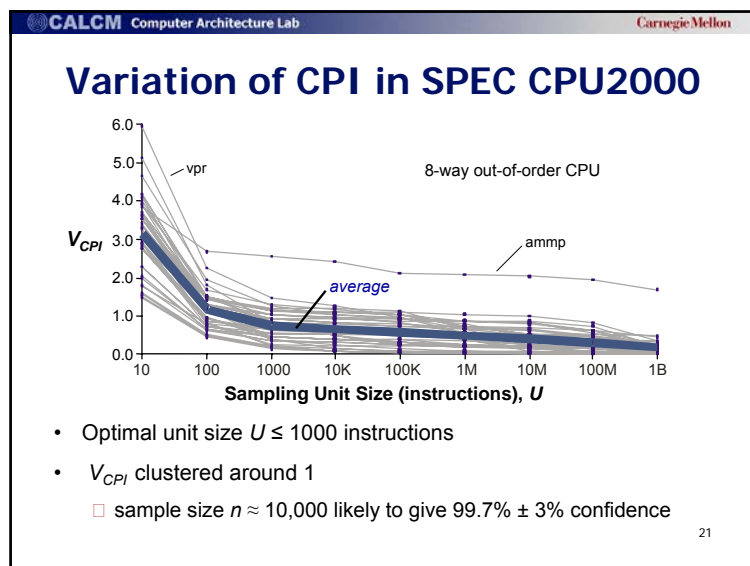
Required sample size n

Total instructions $n \cdot U$

Unit Size U (log scale)

A large sample of small units minimizes total instr.

20



CALCM Computer Architecture Lab Carnegie Mellon

Disparity between Sampling in Theory and in Practice

- Sampling in theory
 - Assumes accurate measurements

- Sampling in practice
 - “Cold” state results in non-random error (bias)

22

CALCM Computer Architecture Lab Carnegie Mellon

Section 2 outline

- Simulation sampling
 - Sampling in theory
 - Sampling in practice: accurate measurements
- Multiprocessor sampling
 - Extending sampling to MP applications
 - Sampling optimizations

23

CALCM Computer Architecture Lab Carnegie Mellon

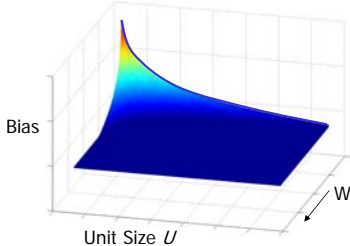

Small units in practice: Bias

- Inexact simulator state
 - Empty pipeline
 - Approximate caches, etc.
- Results in **bias**
 - Non-random error
- Larger effect as U shrinks

24

Small units in practice: Bias

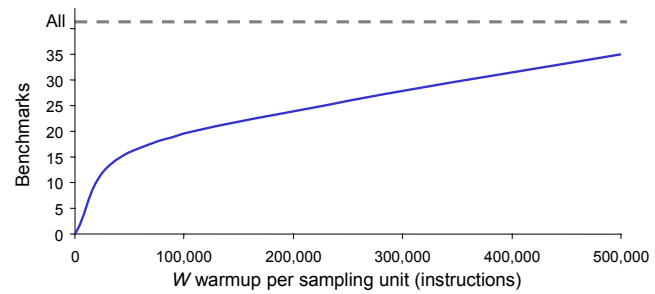
- Inexact simulator state
 - Empty pipeline
 - Approximate caches, etc.
- Results in **bias**
 - Non-random error
- Larger effect as U shrinks
- Solution: Warmup before each unit to correct state

25

Warmup requirements of SPEC CPU2000

Warmup required to achieve $\leq \pm 1.5\%$ Bias




Warmup requirements vary widely and unpredictably

26

Addressing unpredictable warmup

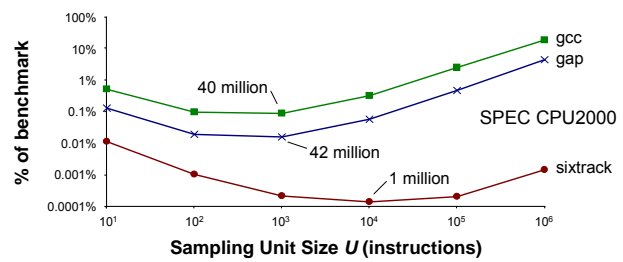
| Type of history | Warmup strategy |
|---|---|
| Short, predictable <ul style="list-style-type: none"> ROB, pipeline registers | Short detailed warmup <ul style="list-style-type: none"> Bound via worst-case analysis |
| Long, unpredictable <ul style="list-style-type: none"> Caches, branch predictors | Long functional warmup <ul style="list-style-type: none"> Update state between units |



27

Optimal sampling parameters

Percent of benchmark detail-simulated $n \cdot (U + W)$
 $W = 2000$; n chosen for 99.7% \pm 3% confidence



*Detailed simulation is nearly negligible (< 50 million instr.)
 But, still need functional warming of entire workload...*

28

Bottleneck: functional warming

- 99% of runtime spent on functional simulation
 - Average SPEC CPU2000 ref. input: **170 billion instructions**
 - Average detailed simulation w/ sampling: **25 million instructions**
- Longer benchmarks
 - Similar sample size because similar V_{CPI}
 - More functional warming

Replace functional warming with checkpoints

29

Functional warming with checkpoints

The diagram illustrates a checkpoint architecture. At the top, a horizontal line represents the simulation flow with several blue rectangular checkpoints. Below this, a 'checkpoint library' is shown as a cylinder containing green spheres. To the right, 'Experiments using checkpoints' are depicted as a sequence of blue and green blocks, indicating that the simulation can be restarted from a checkpoint in the library.

- Checkpointed warmup
 - Same sample design, accuracy, confidence
 - 100- to 1000-way parallelism: no need for MT detailed simulator
 - Ideally, checkpoint state is *reusable*

configuration A
configuration B

30

Simulation sampling conclusions

| | Complete simulation | Sampling + Functional warming | Sampling + checkpoints | |
|-----------------------------|---------------------|-------------------------------|-----------------------------------|---|
| Average (worst) CPI bias | None | 0.6% (1.6%) | 0.6% (1.6%) | <i>Accurate, Statistical guarantees</i> |
| Average benchmark runtime | 5.5 days | 7.0 hours | 91 seconds | <i>Fast</i> |
| Parallel simulation | No | No | 100- to 1000-way | <i>Parallel</i> |
| SPEC CPU2000 ckpt. Library | | | 12 GB | |
| Fixed microarch. parameters | | | Max cache, TLB, branch predictors | |

simulation sampling + checkpoints: Accurate, fast, parallel, confidence intervals

31

Section 2 outline

- Simulation sampling
 - Sampling in theory
 - Sampling in practice: accurate measurements
- Multiprocessor sampling
 - Extending sampling to MP applications
 - Sampling optimizations

32

CALCM Computer Architecture Lab Carnegie Mellon

Primary MP focus: *throughput applications*

- E.g., commercial server workloads
 - Transaction processing, web serving, ...
 - Open problems remain for general MP apps.
- What makes throughput applications hard?
 - Complex setup & tuning, large memory footprint (GBs)
 - Non-deterministic behavior
 - Performance typically measured with transaction throughput
 - Multithreaded, multiprocessor, OS & I/O intensive

Require full-system MP simulation

33

CALCM Computer Architecture Lab Carnegie Mellon

Full-system MP simulation

- Provides VM-like capabilities
 - Implements privileged-mode ISA, peripheral devices, network (client-server apps)
 - Simulates multiple cores, more RAM than host
 - Saving/restoring architecturally visible state
 - Complex warming needs
- Full-system timing simulation prohibitively slow
 - 10⁵× slower than hardware, 10³× than SimpleScalar
 - Simulation sampling + checkpoints is essential

Flexus: designed-in support for sampling + checkpoints

34

CALCM Computer Architecture Lab Carnegie Mellon

Taming the measurement challenge

- Simulation slowdown *per cpu*

| | | |
|-------------------------------------|------------|--------|
| – Real HW: | ~ 500 MIPS | 1 s |
| – Simics (fast-forward simulation): | ~ 15 MIPS | 33 s |
| – Flexus, functional (no timing): | ~ 0.5 MIPS | 16.6 m |
| – Flexus, in-order timing: | ~ 10 kIPS | 13.8 h |
| – Flexus, OoO timing: | ~ 3 kIPS | 46 h |

 - *150 years* for 1-CPU audited TPC-C run in OoO simulation
- Develop sampling approach for MP
 - Use our sampling methodology + checkpoints as a starting point
 - Re-examine each step in the MP context

Simulation sampling + checkpoints is essential

35

CALCM Computer Architecture Lab Carnegie Mellon

MP sampling challenges

- Non-deterministic & interleaved instruction streams
 - Uniprocessor population definition inappropriate
 - *Define population in terms of possible interleavings*
- Long & highly variable transaction latency
 - Unacceptable sample & measurement sizes
 - *Measure fine-grain progress metrics*
- Complex, inter-related queues (e.g., interconnect)
 - Complicated detailed warmup analysis
 - *Empirical queue warmup detection*

36

CALCM Computer Architecture Lab Carnegie Mellon

Simulation sampling summary

- Sample Design
 - Population N units of U instructions
 - Performance metric CPI
 - Detailed warming W worst-case analysis: ~2000 inst.
 - Optimal unit size U from V_{CPI} : ~1000 inst
 - Typical sample size n ~8000

Revisit each aspect for MP applications

37

CALCM Computer Architecture Lab Carnegie Mellon

Population

- **SPEC**: N units of U instructions each
- **Servers**: Instruction stream is not fixed
- Characteristics of server instruction stream
 - Unbounded
 - Non-deterministic
 - Multiple parallel streams

SPEC definition for population inappropriate

38

CALCM Computer Architecture Lab Carnegie Mellon

MP population definition

- Real HW pop.: repeat/extend runs for stable results
 - Non-determinism & interleaving problematic
 - Achieve stable results by *observing many interleavings*
- Sampling pop.: set of reachable states
 - Throughput apps – *random transaction arrivals*
 - Determine minimum run length on real HW (e.g., 30s)
 - Draw sample of reachable states from such a run

39

CALCM Computer Architecture Lab Carnegie Mellon

Constructing a sample in simulation

- Pop. defined in terms of states reachable in OoO
- However, construct sample via fn. warming
 - May give unrepresentative interleaving of start PCs

- Random transaction arrivals – any PC positions possible
- Open problem for non-throughput applications

40

CALCM Computer Architecture Lab Carnegie Mellon

Performance Metric

- **SPEC**: Cycles per instruction (CPI)
- **Servers**: CPI is not proportional to performance
 - Not all instructions make forward progress
 - Frequent spins on I/O and locks
- Desired metric characteristics
 - Proportional to forward progress
 - Responds quickly to performance change
 - Low variance at small unit sizes

41

CALCM Computer Architecture Lab Carnegie Mellon

MP performance metric

- Typical metric: Transactions completed / min. (TPM)
- Coarse progress metrics bad for sampling
 - Transaction completions infrequent
 - High variance of inter-arrival and service times

Can't assess TPM reliably with small (<10M inst.) window

42

CALCM Computer Architecture Lab Carnegie Mellon

Measure fine-grain progress

- User-mode inst. per trans. constant per app. cfg.
- Hence, user-mode $IPC \propto TPM$
 - Validated for TPC-C, SpecWEB [Hankins 2003, Wenisch 2006]
 - Most apps yield rather than spin in user mode

Impact: Same confidence with 1000x shorter measurements

43

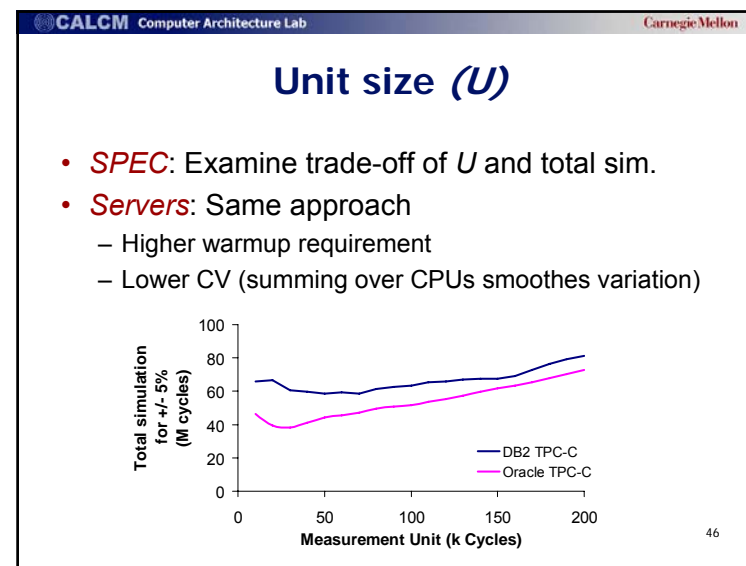
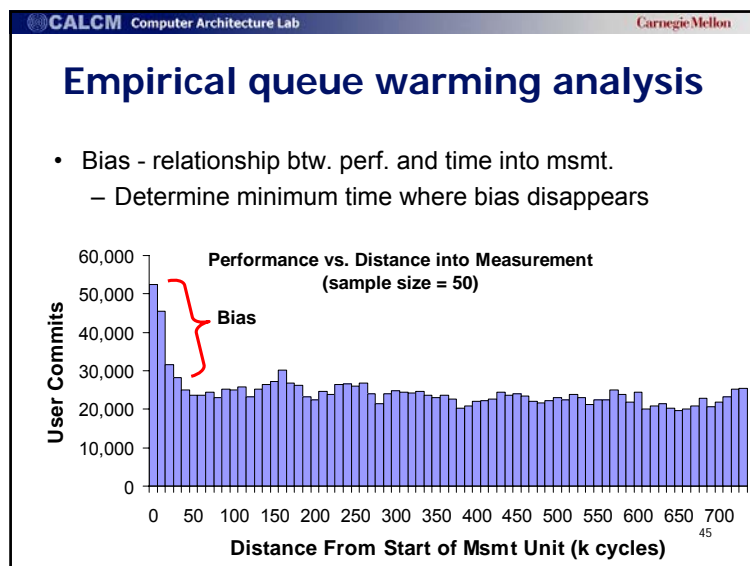
CALCM Computer Architecture Lab Carnegie Mellon

Detailed warming (W)

- **SPEC**: Worst-case analysis; ~2000 instructions
- **Servers**: Worst-case analysis is difficult
- Large queues complicate analysis
 - Miss handlers in memory system
 - Router buffers in interconnect
- Unlike caches, queue state cannot persist indefinitely
 - Queue warming brief, bounded
 - Results in steep slope in cold-start bias

Detect steep bias slope empirically

44



Determining sampling parameters in Flexus

- Construct preliminary checkpoint library
 - 30-50 checkpoints to estimate C.V., detailed warming
 - Good initial guess: C.V. for user IPC \approx 0.5
 - Determine sampling parameters

47

Typical sampling parameters

| 16-core OoO CMP | |
|----------------------------|-----------------|
| Warming | 100k cycles |
| Measurement | 50k cycles |
| Target confidence | 95% \pm 5% |
| Sample size | 200-400 |
| Sim. time per checkpoint | < 20 min |
| Experiment turnaround time | ~ 30 CPU-hours |
| Parallelism | 200- to 400-way |

48

CALCM Computer Architecture Lab Carnegie Mellon

Section 2 outline

- Simulation sampling
 - Sampling in theory
 - Sampling in practice: accurate measurements
- Multiprocessor sampling
 - Extending sampling to MP applications
 - Sampling optimizations

49

CALCM Computer Architecture Lab Carnegie Mellon

Matched-pair comparison [Ekman 05]

- Often interested in relative performance
- Change in performance across designs varies less than absolute change
- Matched pair comparison
 - Allows smaller sample size
 - Reports confidence in performance change

50

CALCM Computer Architecture Lab Carnegie Mellon

Matched-pair example

*Performance results for two microarchitecture designs
checkpoints processed in random order*

*Lower variability in performance deltas
reduces sample size by 3.5 to 150x*

51

CALCM Computer Architecture Lab Carnegie Mellon

Matched-pair in Flexus

- Simple μ Arch changes (e.g., changing latencies)
 - use same checkpoints
- Complex changes (e.g., adding components)
 - use *aligned* checkpoints
 - deterministic execution

52

CALCM Computer Architecture Lab Carnegie Mellon

SimFlex conclusions

| | SPEC 2000 (per input) | Throughput Apps. (OLTP, DSS, Web) |
|----------------------------------|--------------------------|---|
| Measurement time on hardware | 2 min | 30 sec |
| Simulation time without sampling | 4.5 CPU-days | 10-20 CPU-years |
| Time with SimFlex approach | 91 CPU-sec | 70 CPU-hours (but >100-way parallel) |
| Storage for checkpoints | 273 MB | 30 GB |

*SimFlex approach makes simulation studies tractable
Accurate, highly parallel, quantifiable result reliability*

53

CALCM Computer Architecture Lab Carnegie Mellon


Tutorial outline

- Introduction
- Simulation sampling
- **Developing with Flexus**
- ProtoFlex

Developing with Flexus

SimFlex Tutorial – Section 3 of 4

Mike Ferdman



Computer Architecture Lab at
Carnegie Mellon

25 Oct 2008, PACT

CALCM Computer Architecture Lab Carnegie Mellon

Using Flexus

- Flexus philosophy
- Fundamental abstractions
- Important support libraries
- Bringing up the infrastructure
- Conducting research with Flexus

56

CALCM Computer Architecture Lab Carnegie Mellon

Flexus philosophy

- Component-based design
 - Compose simulators from encapsulated components
- Software-centric framework
 - Flexus abstractions are not tied to hardware
- Cycle-driven execution model
 - Components receive “clock-tick” signal every cycle
- SimFlex methodology
 - Designed-in fast-forwarding, checkpointing, statistics

57

CALCM Computer Architecture Lab Carnegie Mellon

Using Flexus

- Flexus philosophy
- Fundamental abstractions
- Important support libraries
- Bringing up the infrastructure
- Conducting research with Flexus

58

CALCM Computer Architecture Lab Carnegie Mellon

Flexus organization

FLEXUS_ROOT

/components **/simulators** **/core**

59

CALCM Computer Architecture Lab Carnegie Mellon

Component interface

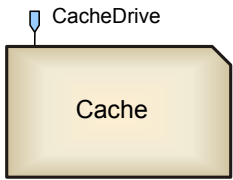
- Component interface (terminology inspired by *Asim* [Emer 02])
 - Drive: “clock-tick” control entry point to component
 - Port: specifies data flow between components

Component w/ same ports are interchangeable

60

CALCM Computer Architecture Lab Carnegie Mellon

Abstractions: Drive



```

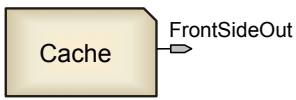
COMPONENT_INTERFACE(
    ...
    DRIVE ( Name )
    ...
);
    
```

- Control entry-point
- Function called once per cycle

61

CALCM Computer Architecture Lab Carnegie Mellon

Abstractions: Port



```

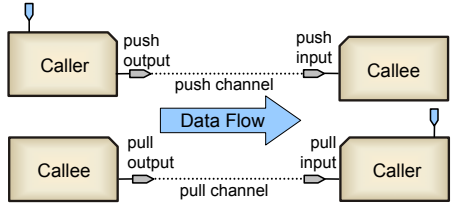
COMPONENT_INTERFACE(
    ...
    PORT ( Type, Payload, Name )
    ...
);
    
```

- Data exchange between components
- Ports connected together in simulator wiring

62

CALCM Computer Architecture Lab Carnegie Mellon

Types of ports and channels




- Type - direction of data and control flow
 - Control flow: Push vs. Pull
 - Data flow: Input vs. Output
- Payload - arbitrary C++ data type
- Type and payload must match to connect ports
- Availability - caller must check if callee is ready

63

CALCM Computer Architecture Lab Carnegie Mellon

Port and component arrays



```

COMPONENT_INTERFACE(
    ...
    DYNAMIC_PORT_ARRAY(...)
    ...
);
    
```

- 1-to- n and n -to- n connections
 - E.g., 1 interconnect -> n network interfaces
- Array dimensions can be dynamic

64

CALCM Computer Architecture Lab Carnegie Mellon

Example code using a port

SenderComponent.cpp

```
void someFunction() {
    Message msg;
    if ( FLEXUS_CHANNEL(Out).available() ) {
        FLEXUS_CHANNEL(Out) << msg;
    }
}
```

ReceiverComponent.cpp

```
bool available( interface::In ) { return true; }
void push( interface::In, Message & msg ) { ... }
```

65

CALCM Computer Architecture Lab Carnegie Mellon

Transports

- Scalable data structure
 - Made up of one or more *slices*
 - Transport object holds pointers to slices
 - Syntax similar to array access

Adding a new slice has no effect on existing code

66

CALCM Computer Architecture Lab Carnegie Mellon

Example of Transport Usage

- NIC packages multiple Slices into Transport
- Network is oblivious to Message contents

67

CALCM Computer Architecture Lab Carnegie Mellon

Reference counted pointers

- Used for object exchange between comps.
 - Reduces need to think about object lifetime
- E.g., ArchitecturalInstruction object
 - Created in InorderSimicsFeeder component
 - Preserved while any comp. references instruction
 - Destroyed once last component “forgets” instruction
- Implementation: Boost intrusive pointer
 - Documentation: www.boost.org/libs/smart_ptr

68

CALCM Computer Architecture Lab Carnegie Mellon

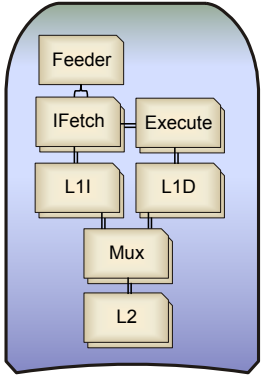
Simulator wiring

`simulators/name/Makefile.name`

- List components for link
- Indicate target support

`simulators/name/wiring.cpp`

1. Include interfaces
2. Declare configurations
3. Instantiate components
4. Wire ports together
5. List order of drives



The diagram shows a hierarchical structure of simulator components. At the top is a 'Feeder' box. Below it are two boxes: 'IFetch' on the left and 'Execute' on the right. Below 'IFetch' is 'L1I', and below 'Execute' is 'L1D'. These four boxes are connected to a central 'Mux' box. Below the 'Mux' box is 'L2'. The entire structure is enclosed in a rounded rectangular frame.

69

CALCM Computer Architecture Lab Carnegie Mellon

Simulators in Flexus 2.1.0

- UniFlex [.OoO] 1 CPU 2-level hierarchy
- CMPFlex [.OoO] private L1 / shared L2
- DSMFlex [.OoO] Distrib. Shared-Memory
- TraceFlex Uni- or DSM-trace
- TraceCMPFlex CMP-trace

OoO variants support v9, all others support v9/x86

70

CALCM Computer Architecture Lab Carnegie Mellon

Using Flexus

- Flexus philosophy
- Fundamental abstractions
- Important support libraries
- Bringing up the infrastructure
- Conducting research with Flexus

71

CALCM Computer Architecture Lab Carnegie Mellon

Important support libraries

- Statistics support library
 - Record results for use with `stat-manager`
- Debug library
 - Control and view Flexus debug messages

72

CALCM Computer Architecture Lab Carnegie Mellon

Statistics support library

- Key features
 - Generate text reports from stats database
 - Calculate formulas & confidence intervals
 - Histograms, unique counters, instance counters
- Example:

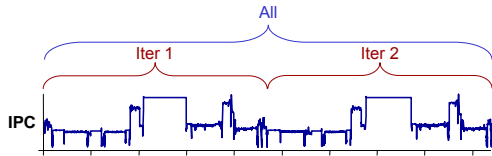

```
Stat::StatCounter myCounter( statName() + "-count" );
++ myCounter;
```

73

CALCM Computer Architecture Lab Carnegie Mellon

Flexus statistics collection model

- Multiple concurrent stat collection windows



- Measurement windows controlled via:
 - Cycle, transaction, instruction counts
 - Programmatically in Flexus code
 - Events in the simulated system (MagicBreak component)

74

CALCM Computer Architecture Lab Carnegie Mellon

stat-manager

- Generates reports using [report templates](#)
 - Text files w/ placeholders
- Example placeholders:
 - `{name}`
 - `{name@Region 002}`
 - `{histogram-name;val:2}`
 - `<EXPR:2*{name}>`
 - `<EXPR:sum{regular expression}>`

75

CALCM Computer Architecture Lab Carnegie Mellon

Aggregating sample results

- `stat-collapse`
 - Aggregate measurement windows from one stats DB
- `stat-sample`
 - Reads DBs produced by `stat-collapse`
 - Produces sum, avg, stdev, count in output DB
 - 95% confidence interval with `<EXPR:ci95{name}>`

76

CALCM Computer Architecture Lab Carnegie Mellon

Debug support library

- Filtering at compile time and runtime
 - Maximum severity set on **make** command line
 - make DSMFlex.OoO-iface
 - Can control severity, component, category at run time
 - Up to severity specified on when building
- Formatting, output to multiple destinations
 - Via debug cfg files

77

CALCM Computer Architecture Lab Carnegie Mellon

Debug severity levels

1. **Tmp** *temporary messages (cause warning)*
2. Crit critical errors
3. Dev infrequent messages, e.g., progress
4. Trace component defined – typically tracing
5. **Iface** *all inputs and outputs of a component*
6. Verb verbose output from OoO core
7. **VVerb** *very verbose output of internals*

78

CALCM Computer Architecture Lab Carnegie Mellon

Controlling debug output

- Compile time
 - make *target-severity*
- Run time
 - flexus.debug-set-severity *severity*
 - flexus.debug-enable-component *component idx*
 - flexus.debug-enable-category *category*
- Output
 - Filter on any field via debug.cfg
 - See debug.cfg and docs in distribution

79

CALCM Computer Architecture Lab Carnegie Mellon

A typical debug statement

```

DBG_(Iface,                               Severity level
     Comp(*this)                           Associate with this component
     AddCategory( Cache )                   Put this in the "Cache" category
( << "Received on FrontSideIn[0](Request): "
  << *(aMessage[MemoryMessageTag])
  )                                         Text of the debug message
     Addr(aMessage[MemoryMessageTag]->address())
);                                         Add an address field for filtering
    
```

80

CALCM Computer Architecture Lab Carnegie Mellon

Using Flexus

- Flexus philosophy
- Fundamental abstractions
- Important support libraries
- Bringing up the infrastructure
- Conducting research with Flexus

81

CALCM Computer Architecture Lab Carnegie Mellon

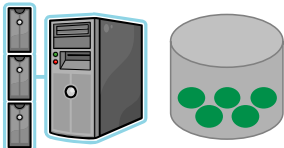
Bringing up the infrastructure

1. Install Flexus (and friends)
2. Running jobs
3. Create workloads
4. Generate Phases/Flex-Points

82

CALCM Computer Architecture Lab Carnegie Mellon

Install Flexus



- SW Requirements:
 - Simics
 - GCC and Boost
 - tcllib
- HW Recommendations:
 - 8+ cores, 2GB RAM/core
 - 1TB+ shared storage (NFS, AFS, PanFS, ...)
 - Work management system (Condor, LSF, ...)

Start with 1 machine – later expand to increase # of CPUs

83

CALCM Computer Architecture Lab Carnegie Mellon

Running jobs

```

run_job [options] <simulator|NONE> <job-spec>
-job value          Override job directory <>
-extend            Extend existing job directory
-clobber           Wipe job directory if it already exists
-cfg value         Config name <interactive>
-run value         Run generator to use <>
-runfxpt value     phase:flexpoint to run (overrides run generator) <>
-state value       FlexStates to use (comma separated) <>
-chkptset value    CheckPointSet to use <baseline>
-simicscmds value  Additional Simics commands <>
-ckpt-gen value    Generate checkpoint after running <>
-skipmissing       Do not abort on missing flexpoints
-condor            Submit jobs to condor
-reqs value        Additional Condor requirements <True>
-local            Run jobs on local machine
-norun            Set up job directory but do not run
-debug value       Debug level <iface>
-win               Skip -nowin argument to Simics (show X11 windows)
-ma               Pass -ma argument to Simics (needed for OoO jobs)
-list-specs        List available job-specs
-list-cfgs         List available configurations
-help             Print this message
    
```

Currently separate download from the web site

84

CALCM Computer Architecture Lab Carnegie Mellon

run_job configuration

- “global.run_job.rc.tcl” – global config file


```
set basedir /home/[exec whoami]/results
set specdir /nfs/job-specs
set chkptdir /nfs/checkpoints
set simicsdir /nfs/simics/simics-3.0.30
```
- “.run_job.rc.tcl” – per-user local config file


```
rungen timing {
  { apache_16cpu_40c1  baseline 0-1:5-24 $cmd(commercial) }
  { zeus_16cpu_40c1   baseline 0-1:5-24 $cmd(commercial) }
}
rungen fxptgen {
  { apache_16cpu_40c1  baseline 0-3:0  $cmd(commercial) }
  { zeus_16cpu_40c1   baseline 0-3:0  $cmd(commercial) }
}
```

85

CALCM Computer Architecture Lab Carnegie Mellon

run_job basic operation

- Create execution directory structure
- Copy executables and config files into directory
- Create “./go.sh” script
- Run “./go.sh”

Assumes “interactive” run by default, 1 job only
 Add “-norun” to avoid running automatically
 Add “-local” to run multiple jobs serially
 Add “-condor” to submit to distrib. system

86

CALCM Computer Architecture Lab Carnegie Mellon

Example job directory

```
run_job -norun -cfg istream -run timing TraceNUCAFlex em3d
```

```
12K   interactive-istream/.skel/user
20K   interactive-istream/.skel/em3d
44K   interactive-istream/.skel/global
8.5K  interactive-istream/em3d/000_001
8.5K  interactive-istream/em3d/000_002
8.5K  interactive-istream/em3d/000_003
8.5K  interactive-istream/em3d/000_004
8.5K  interactive-istream/em3d/000_005
```

87

CALCM Computer Architecture Lab Carnegie Mellon

Creating workloads

- Install and configure workload
 - Real HW or Simulated HW in Simics
 - On real HW, create disk images for Simics:


```
dd if=/dev/dsk/c0t3d0s0 of=specOMPdisk.img
craff specOMPdisk.img -o specOMPdisk.craff
```
- Create “starting” Simics checkpoints (-fast)
 - Run workload in simics, then “write-configuration”
 - Adjust: core frequency, cpu interleaving, console output, system tick frequency

88

CALCM Computer Architecture Lab Carnegie Mellon

Flex-point creation timeline

1. Spread Simics checkpoints
 - via Simics -fast
 - rapidly cover 30s
2. Collect flex-points in parallel
 - via *TraceFlex*
 - From each Simics checkpoint
 - set cache & branch pred. cfg

89

CALCM Computer Architecture Lab Carnegie Mellon

Flex-point creation

- Create “job-spec” directory for workload
 - “job-postload.simics” defines cache/branch pred.
 - Use configuration.out from Flexus as starting point
- Prepare “phase 000” checkpoint
 - Load prepared workload in Simics
 - “write-configuration \ \$CHKPT/baseline/phase_000/simics/phase_000”
- “run_job” with Flex-Point post-processing script

90

CALCM Computer Architecture Lab Carnegie Mellon

Checkpoint Library Structure

```

988K  mgrid_m_16cpu/baseline/phase_000/simics
160M  mgrid_m_16cpu/baseline/phase_000/flexpoint_001/simics
15M   mgrid_m_16cpu/baseline/phase_000/flexpoint_001/baseline
44M   mgrid_m_16cpu/baseline/phase_000/flexpoint_002/simics
16M   mgrid_m_16cpu/baseline/phase_000/flexpoint_002/baseline
44M   mgrid_m_16cpu/baseline/phase_000/flexpoint_003/simics
18M   mgrid_m_16cpu/baseline/phase_000/flexpoint_003/baseline
44M   mgrid_m_16cpu/baseline/phase_000/flexpoint_004/simics
19M   mgrid_m_16cpu/baseline/phase_000/flexpoint_004/baseline
141M  mgrid_m_16cpu/baseline/phase_000/flexpoint_005/simics
19M   mgrid_m_16cpu/baseline/phase_000/flexpoint_005/baseline
...
    
```

91

CALCM Computer Architecture Lab Carnegie Mellon

Using Flexus

- Flexus philosophy
- Fundamental abstractions
- Important support libraries
- Bringing up the infrastructure
- Conducting research with Flexus

92

CALCM Computer Architecture Lab Carnegie Mellon

Conducting Research with Flexus

- Workload statistics collection
- Design implementation and tuning
- FlexState generation
- Timing evaluation

93

CALCM Computer Architecture Lab Carnegie Mellon

Simple Example: Victim Cache Workload statistics collection

- Instrument cache to count Conflict Misses
 - components/FastCache/FastCacheImpl.cpp
- Collect baseline statistics

```
run_job -cfg victim -run trace TraceFlex oracle
```
- Review results

```
stat-manager format victim.rpt
```

victim.rpt contents:

```
L2 D-cache misses: <EXPR:sum{.*L2-Misses:User:D:Read}>
L2 D-cache conflict misses: <EXPR:sum{.*L2-Misses:User:D:Read:Conflicts}>
```

94

CALCM Computer Architecture Lab Carnegie Mellon

Simple Example: Victim Cache Design implementation and tuning

- Model victim cache in trace simulation
 - components/FastCache/FastCacheImpl.cpp
- Tune the design (sizes, replacement policy)

```
run_job -cfg victim-8-lru -run trace TraceFlex oracle
run_job -cfg victim-16-rnd -run trace TraceFlex oracle
```
- Confirm intuition and select design
 - Can reuse victim.rpt for stat-manager

95

CALCM Computer Architecture Lab Carnegie Mellon

Simple Example: Victim Cache FlexState generation

- Implement checkpoint save in TraceFlex
 - components/FastCache/FastCacheImpl.cpp
- Implement checkpoint restore in timing
 - components/Cache/CacheControllerImpl.cpp
- Generate FlexState for design with victim cache

```
run_job -cfg victim -run trace TraceFlex oracle
```

96

CALCM Computer Architecture Lab Carnegie Mellon

Simple Example: Victim Cache Timing Evaluation

- Run timing jobs


```
run_job -cfg victim -run timing TraceFlex oracle
```
- Use `stat-collapse` to select measurements
- Use `stat-sample` to compute speedup
 - generate *sets* of UIPC numbers (baseline and victim)
 - matched-pair comparison on UIPCs

97

ProtoFlex: Practical, Full-System MP Simulations Using FPGAs

Tutorial – Section 4 out of 4

Eric S. Chung



PROTOFLEX
Computer Architecture Lab at
Carnegie Mellon

25 Oct 2008, PACT

CALCM Computer Architecture Lab Carnegie Mellon

Overview

- **Part I: Basic ProtoFlex Concepts**
 - FPGA Virtualization techniques
 - BlueSPARC simulator
 - FPGA-accelerated Instrumentation
- **Part II: Demo**
 - Access CMU's remote FPGA infrastructure
 - Compile and run code within an FPGA-accelerated simulation of 16-CPU UltraSPARC III server
 - Run real-time CMP cache simulations

99

CALCM Computer Architecture Lab Carnegie Mellon

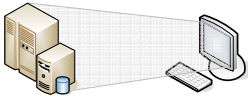
Part I: Basic Concepts

100

CALCM Computer Architecture Lab Carnegie Mellon

Full-system Functional Simulators

- **Exploration of real (or future) HW**
 - Can boot OS, run commercial apps



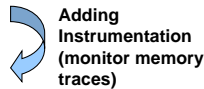
- **But too slow for big (>100-way) MP studies**
 - Existing functional simulators single-threaded
 - High instrumentation overhead

101

CALCM Computer Architecture Lab Carnegie Mellon

Impact of Slow Functional Simulation

- **A Comparison of Simulation Speeds**
 - Simics: ~ 15 MIPS
 - TraceFlex: ~ 0.5 MIPS

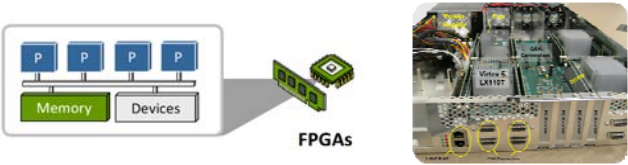


- **Impact of low TraceFlex performance**
 - For 16-core workload with 5B cycles per phase, takes **40 hours** to generate flex-points in SimFlex
- **Unfortunately, cannot parallelize this process**
 - How to address this bottleneck in SimFlex?

102

CALCM Computer Architecture Lab Carnegie Mellon

FPGA-accelerated simulation

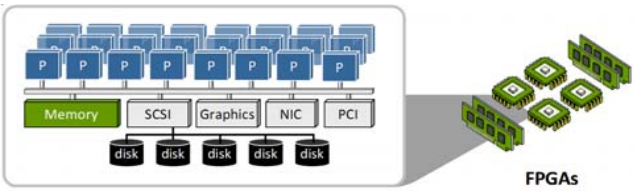


- **Advantages**
 - Large, fast (~100MHz) and reconfigurable
 - Low HW instrumentation performance overhead

103

CALCM Computer Architecture Lab Carnegie Mellon

FPGA-accelerated simulation



- **Unfortunately, FPGAs introduce caveats**
 - Large-MP configurations (>100-cpu) → *nontrivial to build*
 - Full-system support → *need device models + full ISA (also nontrivial)*

104

CALCM Computer Architecture Lab Carnegie Mellon

ProtoFlex: Reducing Complexity

① Hybrid Full-System Simulation

Common-case behaviors Uncommon behaviors

② Multiprocessor Host Interleaving

105

CALCM Computer Architecture Lab Carnegie Mellon

Outline

- Hybrid Full-System Simulation
- Multiprocessor Host Interleaving
- BlueSPARC Implementation
- FPGA-accelerated Instrumentation

106

CALCM Computer Architecture Lab Carnegie Mellon

Hybrid Full-System Simulation

FPGA host PC Host Simulator

- 3 ways to map target components
 - FPGA-only ① Simulation-only ② Transplantable ③
- CPUs can fallback to SW by “transplanting” between hosts
 - Only common-case instructions/behaviors implemented in FPGA
 - Complex, rare behaviors relegated to software

Transplants reduce full-system complexity

107

CALCM Computer Architecture Lab Carnegie Mellon

Multiprocessor Host Interleaving

Advantages:

- Trade away FPGA throughput for smaller implementation
- Decouple logical simulated size from FPGA host size
- Host processor in FPGA can be made very simple

4-to-1 host interleaving

108

CALCM Computer Architecture Lab Carnegie Mellon

FPGA Host Processor

- **Architecturally executes multiple contexts**
 - Existing multithreaded micro-architectures are candidates
- **Our design: Instruction-Interleaved Pipeline**
 - Switch in new processor context on each cycle
 - Simple, efficient design w/ no stalling or forwarding
 - Long-latency tolerance (e.g., cache miss, transplants)
 - Coherence “free” between contexts within same pipeline
 - Achieves fine-grained instruction interleaving

109

CALCM Computer Architecture Lab Carnegie Mellon

Outline


- Hybrid Full-System Simulation
- Host MP Interleaving
- **BlueSPARC Implementation**
- **FPGA-accelerated Instrumentation**

110

CALCM Computer Architecture Lab Carnegie Mellon

BlueSPARC Simulator

- **Func simulator of 16-cpu UltraSPARC server**
 - HW components hosted on BEE2 FPGA platform

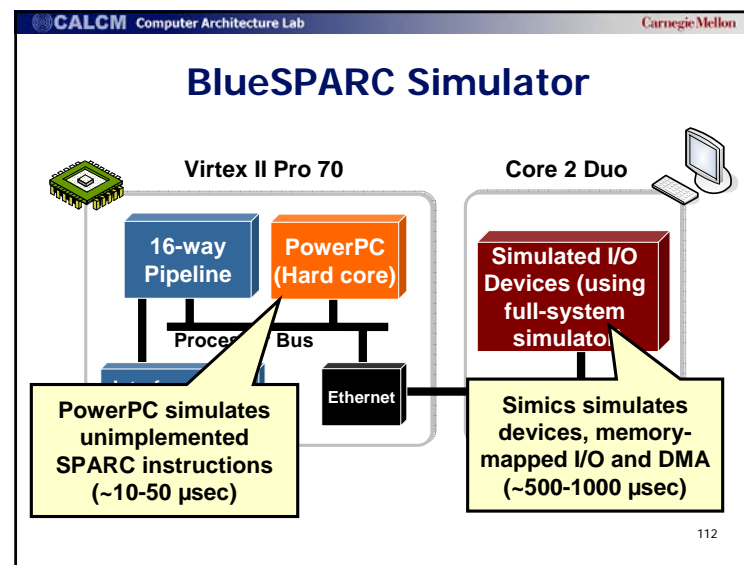


Berkeley Emulation Engine 2

- 5 Virtex-II Pro 70s (~66 KLUTs)
- 2 embedded PPCs per FPGA
- Only use 2 out of 5 FPGAs (pipeline + DDR controllers)

- **Simics used as full-system I/O simulator**

111



CALCM Computer Architecture Lab Carnegie Mellon

Hybrid host partitioning choices

| BlueSPARC | PowerPC405 | Simics on PC |
|--|--|--|
| <ul style="list-style-type: none"> • Integer ALU • Register Windows • Traps/Interrupts • I-/D-MMU • Memory + Atomics • 36% special SPARC insns | <ul style="list-style-type: none"> • Multimedia • Floating Point • Rare TLB operations • 64% special SPARC insns | <ul style="list-style-type: none"> • PCI bus • ISP2200 Fibre Channel • I21152 PCI bridge • IRQ bus • Text Console • SBBC PCI device • Serengeti I/O PROM • Cheerio-hme NIC • SCSI bus |

Implementation = 60% of basic ISA,
36% of special SPARC insns, 0% devices

113

CALCM Computer Architecture Lab Carnegie Mellon

BlueSPARC host microarchitecture

64-bit ISA, SW-visible MMU, complex memory
→ high # of pipeline stages

114

CALCM Computer Architecture Lab Carnegie Mellon

BlueSPARC Simulator (continued)

| | |
|---------------------------------|--|
| Processing Nodes | 16 64-bit UltraSPARC III contexts 14-stage instruction-interleaved pipeline |
| L1 caches | Split I/D, 64KB, 64B, direct-mapped, writeback Non-blocking loads/stores 16-entry MSHR, 4-entry store buffer |
| Clock frequency | 90MHz on Xilinx V2P70 |
| Main memory | 4GB total |
| Resources (Xilinx V2P70) | 33.5 KLUTs (50%), 222 BRAMs (67%) w/o stats+debug 43.2 KLUTs (65%), 238 BRAMs (72%) |
| Instrumentation | All internal state fully traceable Attachable to FPGA-based CMP cache simulator |
| Statistics | 25K lines Bluespec HDL , 511 rules, 89 module types |
| Software | Runs unmodified Solaris + closed-source binaries ¹¹⁵ |

115

CALCM Computer Architecture Lab Carnegie Mellon

Outline

- Hybrid Full-System Simulation
- Host MP Interleaving
- BlueSPARC Implementation
- **FPGA-accelerated Instrumentation**

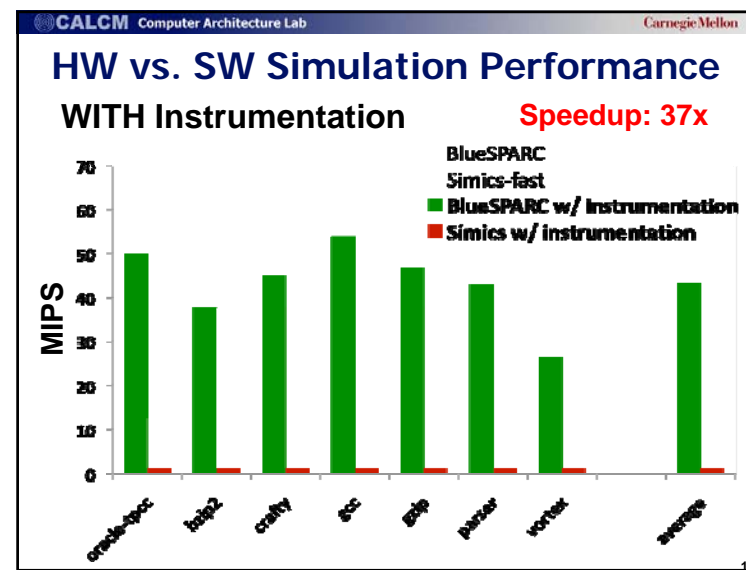
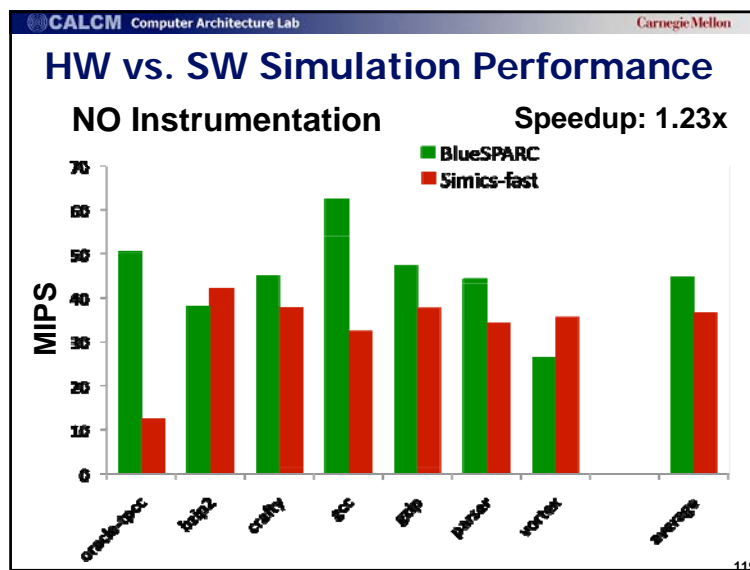
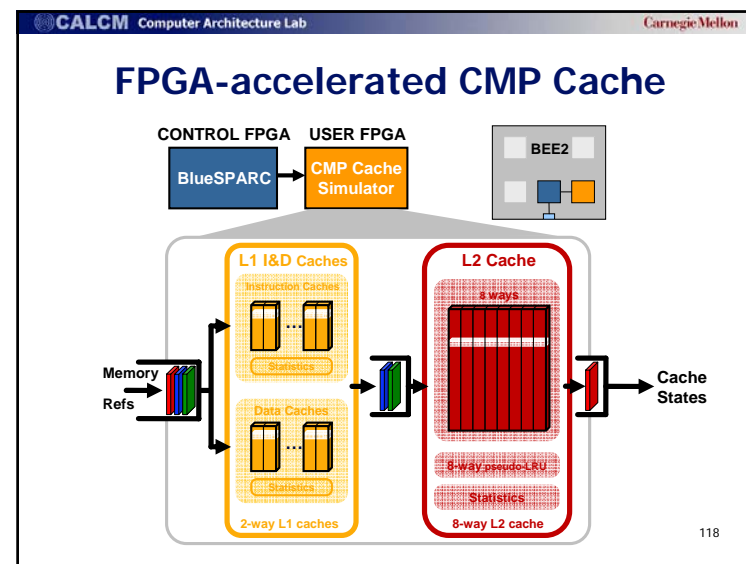
116

CALCM Computer Architecture Lab Carnegie Mellon

FPGA-Accelerated Instrumentation

- **Unlike SW, instrumentation is 'cheap' w/ HW**
 - Fast, parallel monitoring of instruction traces
- **Example: FPGA-accelerated TraceFlex**
 - Model CMP caches and branch predictors in FPGA
 - **10-100x faster than software TraceFlex**

117



CALCM Computer Architecture Lab Carnegie Mellon

Speeding up SimFlex

- **Recall TraceFlex performance: ~0.5 MIPS**
 - For 16-way workload with 5B cycles per phase, takes **40 hours** to generate flex-points in SimFlex
- **With ProtoFlex: ~10-50 MIPS**
 - At 10 MIPS for same workload, takes **2 hours**

121

CALCM Computer Architecture Lab Carnegie Mellon

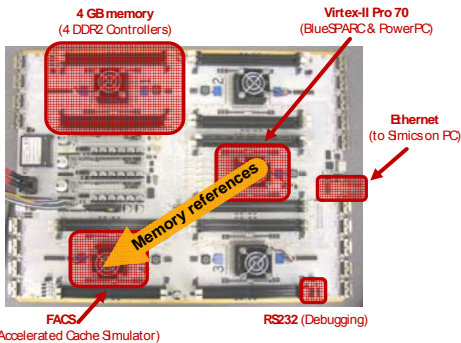
Part II: Demo

122

CALCM Computer Architecture Lab Carnegie Mellon

BlueSPARC Demo on BEE2

- **Demo application**
 - OLTP benchmark (TPC-C) in Oracle
 - Runs in Solaris 8 (unmodified binary)
 - FPGA + Memory directly loaded from Simics checkpoint
 - CMP cache simulator runs on separate FPGA



123

CALCM Computer Architecture Lab Carnegie Mellon



124

CALCM Computer Architecture Lab Carnegie Mellon

Live CMP Cache Statistics

- **Statistics updated in real-time on webpage:**
 - <http://www.ece.cmu.edu/~prototflex/pact08>
- **Click on tabs to watch statistics for any of the BEE2 boards as they run simulations**

125

CALCM Computer Architecture Lab Carnegie Mellon

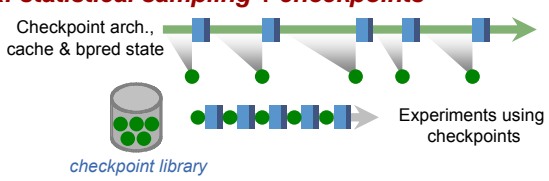
ProtoFlex Summary

- **Two techniques for reducing complexity**
 - Hybrid full-system simulation
 - MP host interleaving
- **Future work**
 - Timing extensions
 - Larger-scale implementation (hundreds of CPUs)
 - Run-time instrumentation tools
- **Distribution**
 - Planned source release in 2009

126

CALCM Computer Architecture Lab Carnegie Mellon

Concluding Remarks

- 1,000,000× slowdown vs. HW → years per experiment
- **SimFlex: statistical sampling + checkpoints**
 - Sampling: ~10,000× reduction in turnaround time
 - Independent measurements: 100- to 1000-way parallelism
 - Confidence intervals: quantifiable result reliability
 - **ProtoFlex: FPGA-accelerated checkpoint collection**
- **Impact**
 - Sampling: ~10,000× reduction in turnaround time
 - Independent measurements: 100- to 1000-way parallelism
 - Confidence intervals: quantifiable result reliability
 - **ProtoFlex: FPGA-accelerated checkpoint collection**

127

CALCM Computer Architecture Lab Carnegie Mellon

Reference slides

- Flexus component inventory
- Configuring components
- Debug system details
- Controlling Flexus in code
- Flex-point implementation
- Live-point implementation
- What needs to be checkpointed?
- The SimFlex experiment procedure
- Simulation sampling results
- Workload scaling: DBmbench
- Related simulators
- Citations
- SimFlex team

128

CALCM Computer Architecture Lab Carnegie Mellon

Flexus component inventory (1)

- BPWarm – branch predictor for checkpoint creation
- Cache – cache implementation for timing simulation
- CMPCache – shared L2 cache component for timing simulation
- Common – definitions for slices, transports, other shared code
- DecoupledFeeder – interacts with Simics for non-timing simulation
- Directory – DSM directory state storage
- Execute – in-order timing instruction execution unit
- FastBus – coherence/snooping bus for non-timing simulation
- FastCache – non-CMP cache for non-timing simulation

129

CALCM Computer Architecture Lab Carnegie Mellon

Flexus component inventory (2)

- FastCMPCache – CMP cache for non-timing simulation
- FastMemoryLoopback – main memory for non-timing simulation
- FetchAddressGenerate – branch predictor for OoO simulation
- IFetch – instruction fetch unit for in-order simulation
- InorderSimicsFeeder – interacts with Simics for in-order simulation
- LocalEngine – manages local memory interaction in DSM
- MagicBreak – intercepts magic breakpoints & simulated sys. events
- MemoryLoopback – main memory for timing simulation
- MemoryMap – controls assignment of memory pages to DSM nodes

130

CALCM Computer Architecture Lab Carnegie Mellon

Flexus component inventory (3)

- MTManager – coordination component for fine-grain multi-threading
- NetShim – interconnect simulator for DSM
- Nic – network interface for connecting to NetShim
- ProtocolEngine – micro-coded DSM coherence protocol engines
- TraceTracker – constructs event traces for cache components
- uArch/v9Decoder – out-of-order core implementation
- uFetch – fetch unit and L1 I-cache for OoO timing

131

CALCM Computer Architecture Lab Carnegie Mellon

Configuring components

- Configurable settings associated with component
 - Declared in component specification
 - Can be std::string, int, long, long long, float, double, enum
 - Declaration: `PARAMETER(BlockSize, int, "Cache block size", "bsize", 64)`
 - Use: `cfg.Associativity`
- Each component instance associated with **configuration**
 - Configuration declared, initialized in simulator wiring file
 - Complete name is `<configuration name>:<short name>`
- Usage from Simics console
 - `flexus.print-configuration` `flexus.write-configuration "file"`
 - `flexus.set "-L2:bsize" "64"`

132

CALCM Computer Architecture Lab Carnegie Mellon

Debug severity levels

1. Tmp temporary messages (cause warning)
2. Crit critical errors
3. Dev infrequent messages, e.g., progress
4. Trace component defined – typically tracing
5. Iface all inputs and outputs of a component
6. Verb verbose output from OoO core
7. VVerb very verbose output of internals

133

CALCM Computer Architecture Lab Carnegie Mellon

Controlling debug output

- Compile time
 - make *target-severity*
- Run time
 - flexus.debug-set-severity *severity*
 - flexus.debug-enable-component *component idx*
 - flexus.debug-enable-category *category*
- Output
 - Filter on any field via debug.cfg
 - See debug.cfg and docs in distribution.
 - BNF in core/debug/parser.cpp

134

CALCM Computer Architecture Lab Carnegie Mellon

Controlling Flexus in code

- theFlexus object – overall simulation control
- Controls simulation, interaction with Simics
 - Get cycle number: `theFlexus->cycleCount()`
 - Stop simulation: `theFlexus->terminateSimulation()`
 - see `core/flexus.hpp`, `core/flexus.cpp`
- Add commands to Simics console interface
 - see `Flexus_Obj::defineClass()` in `flexus.cpp`

135

CALCM Computer Architecture Lab Carnegie Mellon

Components' flex-point support

- Store μ arch state alongside Simics checkpoints
- Store only non-transient state
 - Global “quiesce” flag halts Flexus instruction fetch
 - Each comp. queried to see if it has transient state
 - Save after transient state has drained
- Eases state **portability** across timing models
 - OoO and trace components share format

136

CALCM Computer Architecture Lab Carnegie Mellon

Flex-point example: Cache component

- Permanent state (saved in live-point):
 - Tag array contents
 - Cache line state
 - Replacement order
- Transient state (drained before save):
 - Miss status register contents
 - Input/output request queue contents

All in-flight memory ops drained during quiesce

137

CALCM Computer Architecture Lab Carnegie Mellon

Uncompressed live-point contents

- Live-state stores in each live-point
 - Touched memory data from committed instruction stream
 - Complete cache tag arrays, approximates wrong-path schedule

138

CALCM Computer Architecture Lab Carnegie Mellon

Identifying live-state subset

- At checkpoint creation time
 - Touched subset known only for committed instruction stream
 - Not known for wrong-path (speculative) instructions
- Effects of live-state on simulation
 - Wrong-path instruction latency affects scheduling
 - Pipeline resource contention
 - Wrong-path operand values rarely affect instruction throughput

*We store only state required for correct path execution,
approximate wrong-path scheduling*

139

CALCM Computer Architecture Lab Carnegie Mellon

What needs to be checkpointed?

- Functional warming for multiple configurations
 - Same architectural state
 - Different microarchitectural state
- If checkpoints replace functional warming
 - SMARTS accuracy & confidence in results
 - Huge speedup & parallelism
 - Online results & matched-pair comparison

140

CALCM Computer Architecture Lab Carnegie Mellon

Checkpoint requirements

1. Reusable warm microarchitecture state
 - Cache hierarchy
 - Branch predictor
2. Small & fast loading
 - Average SPEC CPU2000 memory footprint: 105 MB
 - Must process in < 2 sec/checkpoint to improve SMARTS
3. Independent – no sequential dependency
 - Parallelism
 - Sampling optimizations

141

CALCM Computer Architecture Lab Carnegie Mellon

Independent checkpoints

- 100- to 1000-way parallelism
 - Distribute checkpoints across compute cluster
 - No need to multithread detailed simulator
- Online results
 - Report results while simulation in progress
 - Results converge as checkpoints are processed
- Matched-pair comparison
 - Comparative experiments need smaller sample
 - Provides confidence in performance delta

142

CALCM Computer Architecture Lab Carnegie Mellon

Reusable cache state

- Goal – reconstruct multi-configuration warm caches
 - Replay minimal memory trace into cache
 - Concurrently developed: Memory Timestamp Record [Barr 2005]
- Checkpoint creation
 - Simulate maximum interesting cache size, associativity
 - Track cache line access times
 - Store time ordered list of cache-resident memory addresses
- Checkpoint usage: replay memory trace

143

CALCM Computer Architecture Lab Carnegie Mellon

Remaining problems

- Reusable branch predictor state
 - Current solution: store multiple warm branch predictor configs
 - Branch trace compression [Barr 2006]
 - Efficiently stores branch outcome trace to train predictor
- Main memory data size
 - Average SPEC CPU2000 memory footprint: 105 MB
 - Results in storage problem & slow loading checkpoints
 - Store non-speculative memory subset, complete cache tags

144

CALCM Computer Architecture Lab Carnegie Mellon

The SimFlex experiment procedure

1. Prepare workload for simulation
 - load tuned workload into Simics
2. Measure baseline variance
 - determines sampling parameters
3. Collect checkpoints
 - via functional warming
4. Shuffle live-point library
 - randomize for online results
5. Baseline experiment
 - estimates absolute performance
6. Matched-pair experiments
 - estimates relative performance

145

CALCM Computer Architecture Lab Carnegie Mellon

1. Prepare a workload for simulation

- Load application in Simics
 - Install & configure workload on real system
 - Tune workload parameters
 - Collect disk images from real system
 - Up to 30GB per workload for disk images
 - Collect Simics checkpoint of warm application (e.g., warm DB buffer pool)
- For instructions, see
 - *Simics User Guide* §8
 - *Simics Serengeti Target Guide* §6
 - Tutorial Part 3 of 4

Ideally, do as much as possible on Real HW

146

CALCM Computer Architecture Lab Carnegie Mellon

Preparing Simics disk images

- Best approach: collect image from real system
 - Install & configure workload on real system
 - Collect images of disk partitions with `dd` & `craff`
 - See *Simics User Guide* §8
- Alternative: install in simulation
 - See *Simics Serengeti Target Guide* §6
 - Not recommended for commercial server software

Up to 30GB per workload for disk images

147

CALCM Computer Architecture Lab Carnegie Mellon

2. Determine sampling parameters

- Flexus can't switch modes during simulation
 - Simics runs in different modes for each timing model
- Instead, construct preliminary flex-point library
 - 30-50 flex-points to estimate C.V., detailed warming
 - Good initial guess: C.V. for user IPC ≈ 0.5
 - Determine sampling parameters

148

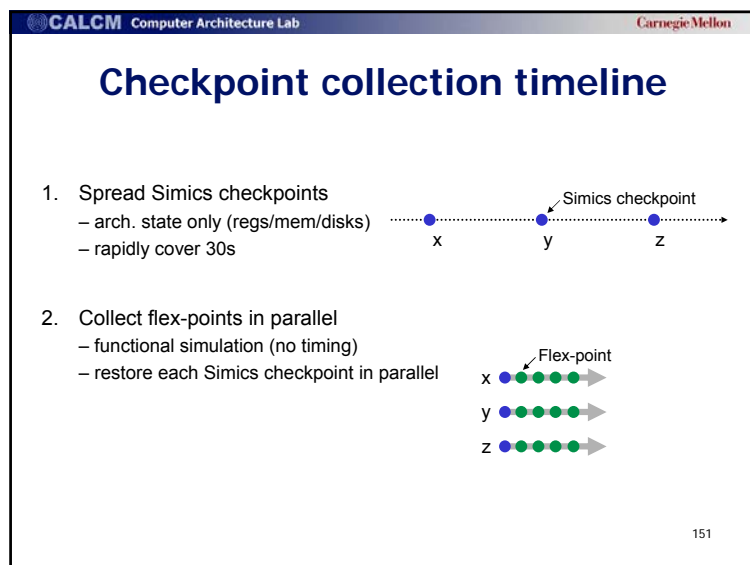
CALCM Computer Architecture Lab Carnegie Mellon

Typical sampling parameters

| 16-core OoO CMP | |
|----------------------------|-----------------|
| Warming | 100k cycles |
| Measurement | 50k cycles |
| Target confidence | 95% ± 5% |
| Sample size | 200-400 |
| Sim. time per checkpoint | < 20 min |
| Experiment turnaround time | ~ 30 CPU-hours |
| Parallelism | 200- to 400-way |

149

- CALCM Computer Architecture Lab Carnegie Mellon
- ## 3. Collect checkpoints
- Single run too slow to cover 30s of execution time
 - Two-tier approach: Simics ckpts (fast), then flex-points (in parallel)
 - Simics ckpts.: complete architectural state
 - Full checkpoints: size \propto RAM (~ 2GB)
 - Delta checkpoints: size \propto memory updates (~300MB)
 - Flex-points: large μ Arch state (e.g., cache, bpred)
 - Fully deterministic (no timing feedback into Simics)
 - Fast (~0.5 MIPS)
 - 100 insn / CPU Simics simulation quantum
 - *TraceFlex*, *TraceDSMFlex* or *TraceCMPFlex*
- Storage constraints limit sample sizes to 100's*
- 150



- CALCM Computer Architecture Lab Carnegie Mellon
- ## 4. Randomize the checkpoint library
- Shuffle flex-points on use
 - Individual flex-points 10-200MB compressed
 - I/O perf. unaffected by shuffling in advance
- 152

CALCM Computer Architecture Lab Carnegie Mellon

5. Detailed simulation of baseline

- Detailed measurement after each flex-point
 - Fast (small units), parallel (100- to 1000-way)
- Aggregate results offline
- Manipulate results & compute statistics
 - Offline tools to select subsets
 - Generate text reports from simple templates
 - Compute confidence intervals for mean estimates

More details at Section 3 of this tutorial

153

CALCM Computer Architecture Lab Carnegie Mellon

6. Matched-pair comparative evaluation

- Simple μ Arch changes (e.g., changing latencies)
 - use same flex-points
- Complex changes (e.g., adding components)
 - use *aligned* flex-points

- *TraceFlex* is fully deterministic
 - Produces identical insn. stream across simulations
 - Flex-points can share Simics state

154

CALCM Computer Architecture Lab Carnegie Mellon

Sampling optimizations

- Online results
 - Report results while simulation in progress
 - Process precise sample size for target confidence
 - Rapid feedback to detect “broken” experiments
- Matched-pair comparison
 - Reduced sample size for comparative experiments
 - Faster turnaround
 - Lower storage requirements

155

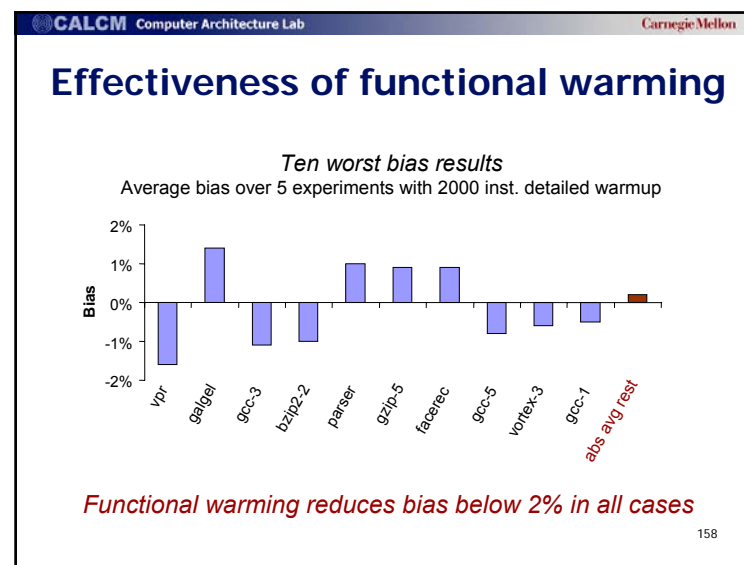
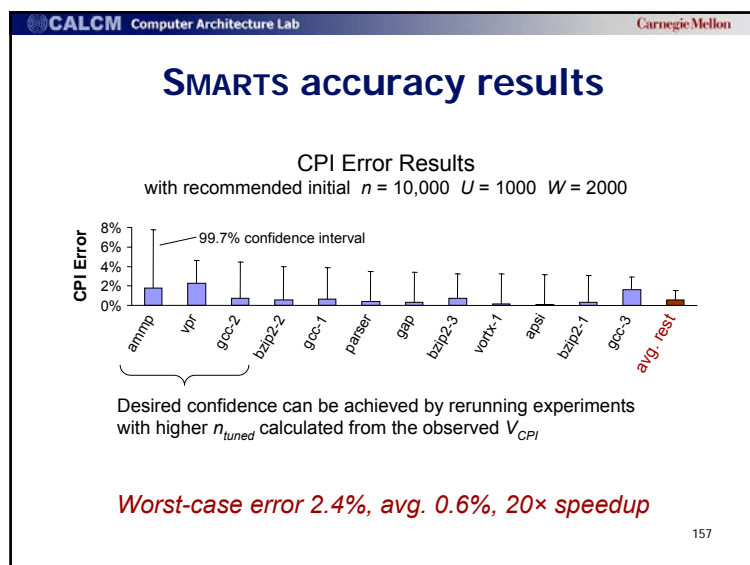
CALCM Computer Architecture Lab Carnegie Mellon

Online results

- Checkpoints can be processed in arbitrary order
 - Random subset of checkpoints forms valid sample
 - Simulating in random order allows online results

As checkpoints are processed, results converge toward their final values & confidence improves

156



- An Orthogonal Approach:
Workload Scaling**
- Reduce simulation time by simplifying workload
 - Reduces performance variance
 - Single measurement may capture reachable states
 - Macro perf. often dominated by common cases
 - Create simple workloads that stress common case
 - A.k.a. Micro-benchmarks
 - Caveat 1: Scaling must not alter behavior
 - Caveat 2: Results **do not represent** full workload
- 159

- DBmbench**
- Database μ marks for μ arch research
 - Identify dominant DB operators in TPC-C and TPC-H
 - Tune simple queries to produce same μ Arch behavior
 - Validated using HW perf. counters on Pentium III
 - Key ideas for successful scaling
 - Scale for a specific goal – e.g., μ arch behavior
 - Validate relevant metrics – e.g., miss & CPI breakdown
- 160

CALCM Computer Architecture Lab Carnegie Mellon

Related simulators

- GEMS [U. Wisconsin]
 - Strength: coherence protocols, transactional memory
- M5 [U. Michigan]
 - Strength: network & I/O integration
- Liberty [Princeton]
 - Strength: structural modeling
- PTLSim [Binghamton]
 - Strength: full-system cycle-accurate x86

Flexus' focus: component-based design

161

CALCM Computer Architecture Lab Carnegie Mellon

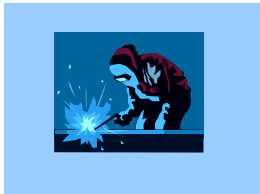
Citations

- T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, J. C. Hoe, "SimFlex: Statistical Sampling of Computer Architecture Simulation," *IEEE Micro* Special Issue on Computer Architecture Simulation, July/August 2006.
- T. F. Wenisch, R. E. Wunderlich, B. Falsafi, J. C. Hoe, "Simulation Sampling with Live-points," Proceedings of the *International Symposium on Performance Analysis of Software and Systems (ISPASS)*, March 2006.
- R. E. Wunderlich, T. F. Wenisch, B. Falsafi, J. C. Hoe, "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling," Proceedings of the *30th International Symposium on Computer Architecture (ISCA)*, June 2003.
- Complete list on SimFlex website

162

CALCM Computer Architecture Lab Carnegie Mellon

SimFlex developers




- Students
 - Eric Chung
 - Mike Ferdman
 - Brian Gold
 - Nikos Hardavellas
 - Ippokratis Pandis
 - Michael Papamichael
 - Stephen Somogyi
 - Evangelos Vlachos
 - Roland Wunderlich
- Faculty
 - Anastassia Ailamaki
 - Babak Falsafi
 - James Hoe
- Alumni
 - Shelley Chen
 - Jangwoo Kim
 - Minglong Shao
 - Jared Smolens
 - Tom Wenisch

163

Thank You!

Questions?



PROTOFLEX
Computer Architecture Lab at
Carnegie Mellon

25 Oct 2008, PACT