

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

Thomas Wenisch
Roland Wunderlich

Anastassia Ailamaki
Babak Falsafi
James Hoe

Kun Gao
Brian Gold
Nikos Hardavellas
Jangwoo Kim

Ippokratis Pandis
Minglong Shao
Jared Smolens
Stephen Somogyi

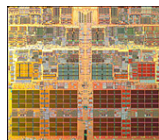


Computer Architecture Lab at
Carnegie Mellon

18 June 2006, ISCA-33

Simulation speed challenges

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




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

2

CALCM Computer Architecture Lab Carnegie Mellon

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: **quantified** 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 Carnegie Mellon

SimFlex tutorial

- Simulation sampling
 - Background & theory
 - Best practices
- Flexus full-system simulation framework
 - Simulator usage & multiprocessor sampling
 - Developing with Flexus

*Best practices for
simulation-based multiprocessor research*

4

CALCM Computer Architecture Lab Carnegie Mellon

Tutorial outline


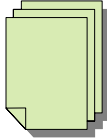
- Introduction (15 min)
- Simulation sampling (45 min)
- Multiprocessor workloads on Flexus (1 hour)
- Developing with Flexus (1 hour)

5

CALCM Computer Architecture Lab Carnegie Mellon

Tutorial materials

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



6

CALCM Computer Architecture Lab Carnegie Mellon



SimFlex terminology

- SimFlex research group
 - SMARTS sample design
 - SMARTSim, TurboSMARTS simulators
 - Flexus full-system simulation framework
 - TraceFlex, UniFlex, DSMFlex, etc. simulators

7

CALCM Computer Architecture Lab Carnegie Mellon

SMARTS & TurboSMARTS

- SMARTS: Sampling Microarchitecture Simulation
 - SMARTSim extends SimpleScalar with sampling
 - SPEC CPU2000 avg. benchmark in 7 hrs vs. 5.5 days
- TurboSMARTS: Live-point support
 - Checkpointed warming enables parallel simulation & online results
 - SPEC CPU2000 avg. benchmark in 91 seconds

8

CALCM Computer Architecture Lab CarnegieMellon

Flexus framework

- Full-system simulation of unmodified commercial apps
 - In-order timing on any Simics target
 - OoO timing for SPARC v9
- Component-based design
 - Multiple timing modes trade accuracy for speed
 - Easy composition of complex system models
- Uniprocessor, CMP, DSM system models
 - CMP, DSM hardware models based on Piranha
 - Aggressive OoO core tuned to produce high memory parallelism
- Designed-in support for simulation sampling
 - Checkpointing of arch and μ arch state
 - Statistic aggregation and confidence calculations

9

CALCM Computer Architecture Lab CarnegieMellon

Flexus timing modes

- No timing (e.g., *TraceFlex*) ~1.5 MIPS
 - High speed functional simulator
 - For trace studies and checkpoint creation
- In-order timing (e.g., *UniFlex*) ~10 kIPS
 - Simics blocked while Flexus times a memory op
 - Easily modified to support any Simics target
- Out-of-order timing (e.g., *UniFlex.OoO*) ~3 kIPS
 - Timing first simulation approach [Mauer 02]
 - SPARC v9 ISA

10

CALCM Computer Architecture Lab CarnegieMellon

Related simulators

- GEMS [U. Wisconsin]
 - Strength: coherence protocols & optimization

- M5 [U. Michigan]
 - Strength: network & I/O integration

- Liberty [Princeton]
 - Strength: structural modeling

Flexus' focus: component-based design


11

CALCM Computer Architecture Lab CarnegieMellon

Flexus usage at Carnegie Mellon

- STEMS – Spatio-Temporal Memory Streaming
- TRUSS – Total Reliability Using Scalable Servers
- ProtoFlex – Hardware prototype component verification
- Staged DB/CMP – Pipelining DB operators across CMP
- Graduate computer architecture classes

12

 **CALCM** Computer Architecture Lab CarnegieMellon


Tutorial outline

- Introduction
- **Simulation sampling**
- Multiprocessor workloads on Flexus
- Developing with Flexus

Simulation Sampling – Background & Theory

SimFlex Tutorial – Section 2 of 4

Roland Wunderlich



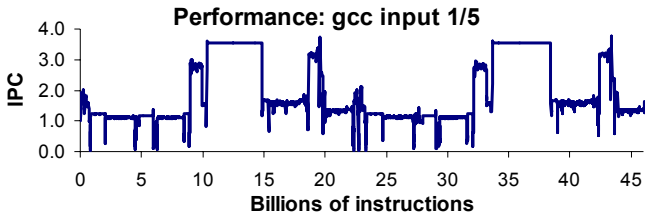
**Computer Architecture Lab at
CarnegieMellon**

18 June 2006, ISCA-33

CALCM Computer Architecture Lab CarnegieMellon

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

15

CALCM Computer Architecture Lab CarnegieMellon

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
- Live-points – checkpointed warming
 - Enables more speed, parallelism, and online results
- SMARTSim & TurboSMARTS results
 - 0.6% CPI error, 20× & 5000× speedup on SPEC CPU2000

16

CALCM Computer Architecture Lab Carnegie Mellon

Section 2 outline

- Sampling in theory
 - Sampling theory recommendations (assuming accurate measurements)

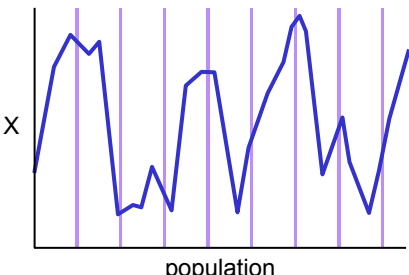
- Sampling in practice
 - Accurately measuring small units
 - Online warming: functional warming
 - Checkpoints: live-points

17

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.



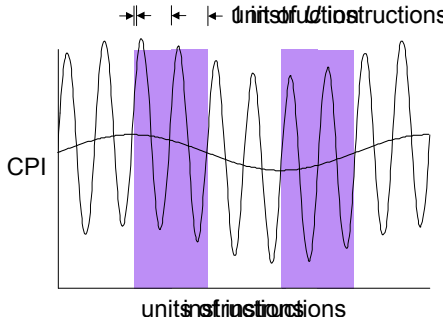
- Arbitrary distribution
- Confidence
 - e.g., 99.7% probability of $\pm 3\%$ error
- $n = f(\text{C.V.}, \text{confidence})$

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

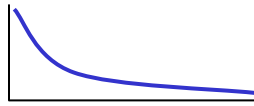
19

CALCM Computer Architecture Lab
Carnegie Mellon

Minimizing total instructions

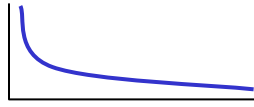
Coefficient of variation

 V_{CPI}



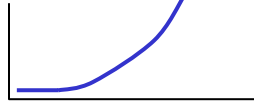
Required sample size

 n



Total instructions

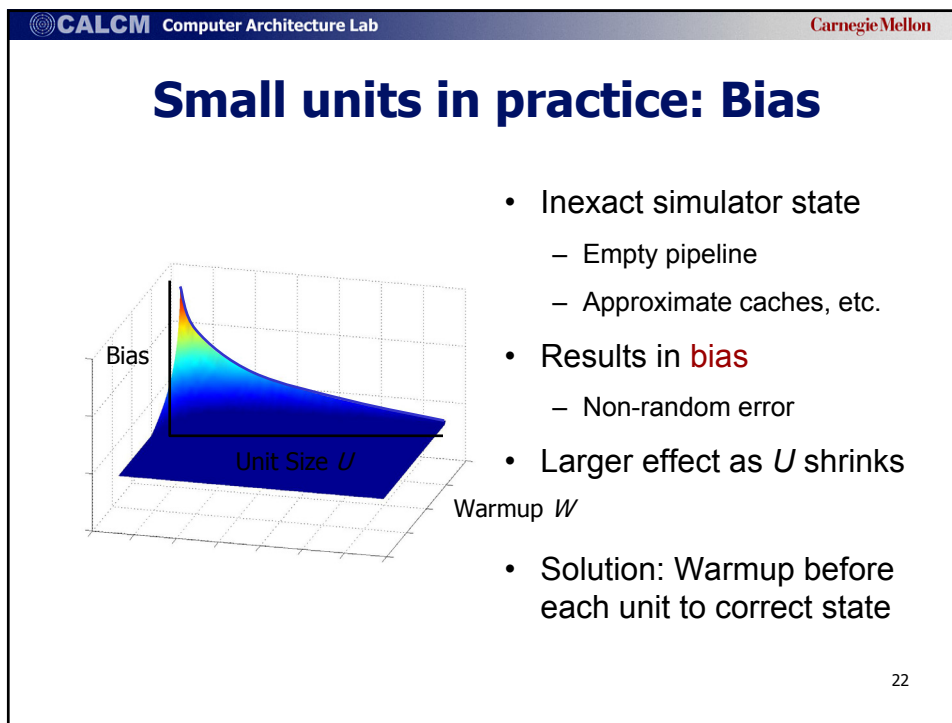
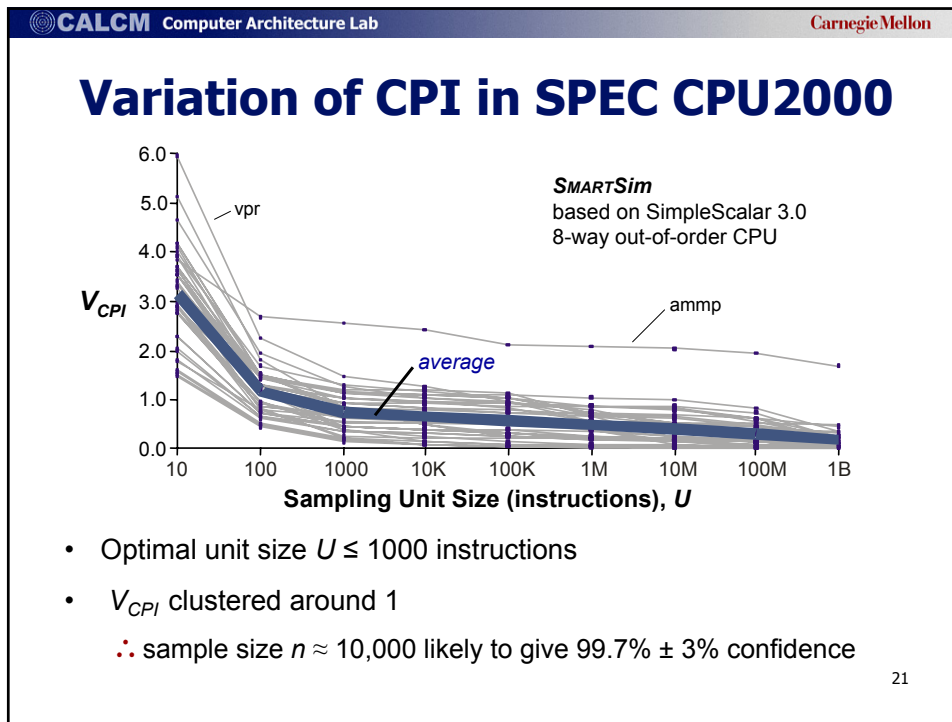
 $n \cdot U$

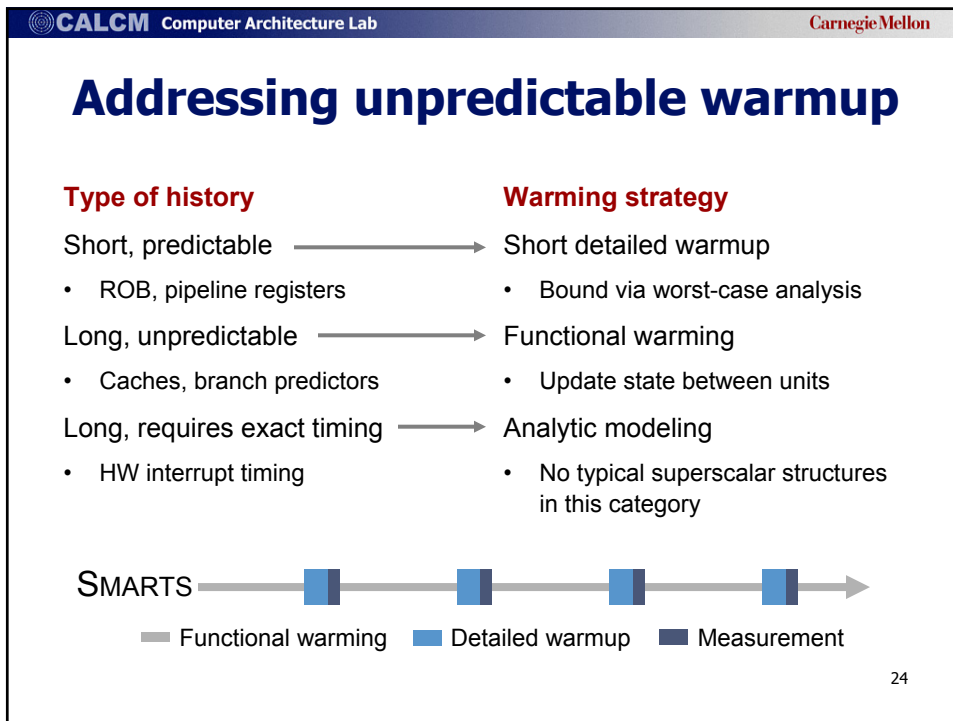


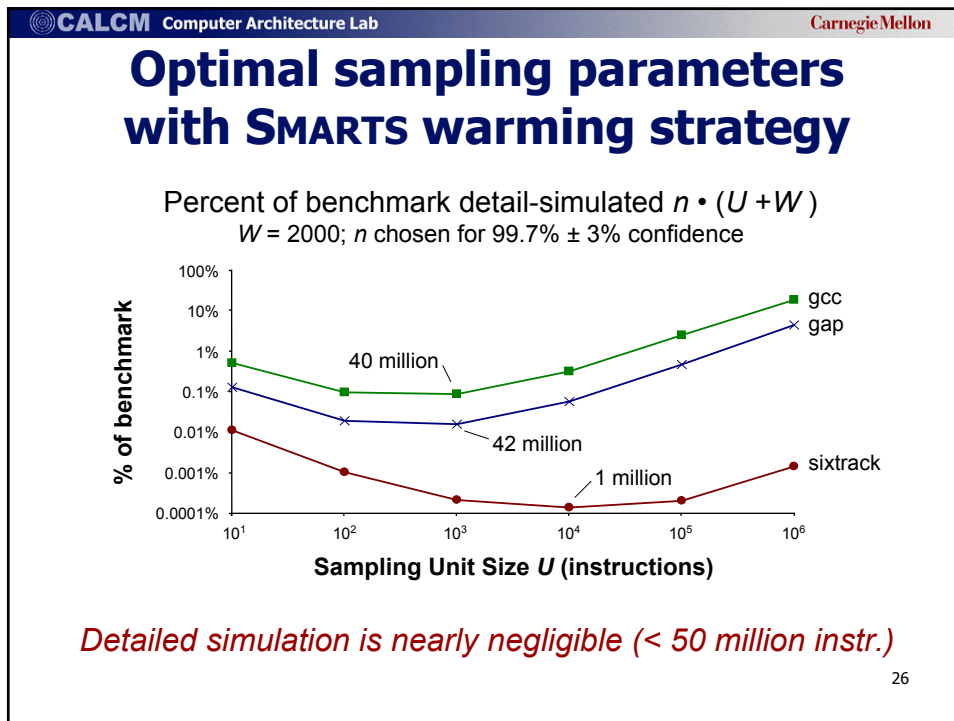
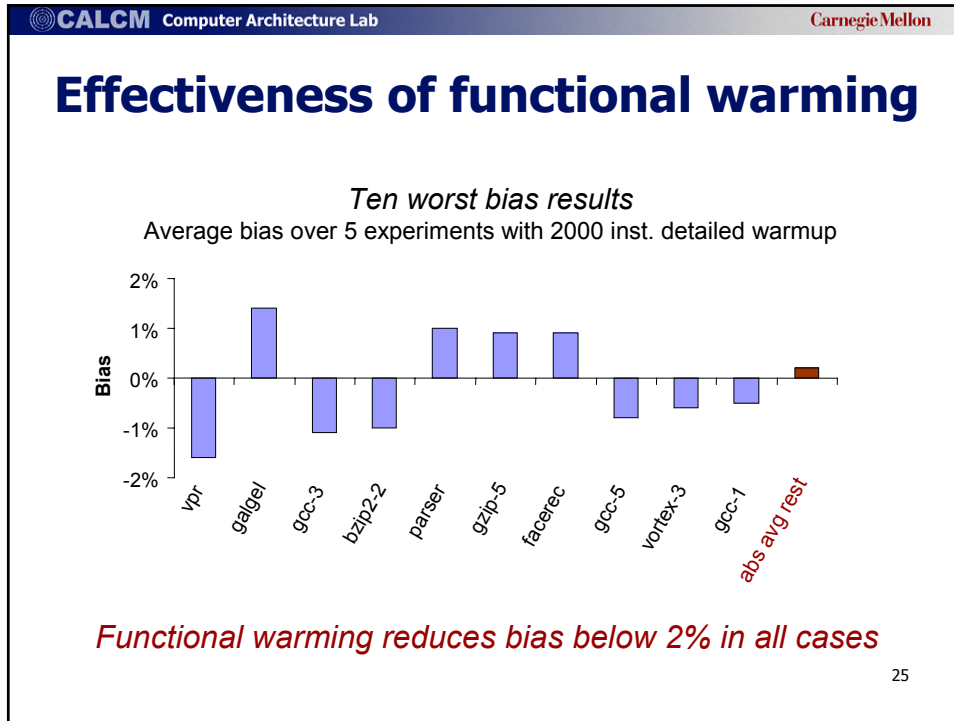
Unit Size U (log scale)

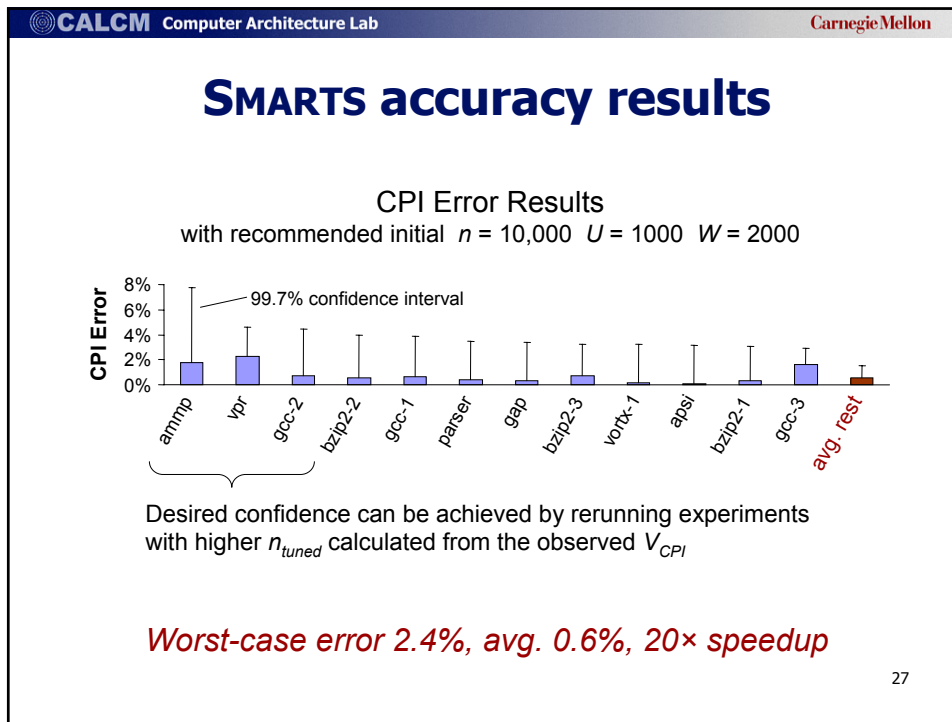
A large sample of small units minimizes total instr.

20









CALCM Computer Architecture Lab **Carnegie Mellon**

What is SMARTS runtime problem?


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

Replace functional warming with checkpoints

28

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

29

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

30

CALCM Computer Architecture Lab Carnegie Mellon

Warm microarchitecture state

Checkpoint arch., cache & bpred state

checkpoint library

Experiments using checkpoints

- Store warm cache & branch predictor state
 - Same sample design, accuracy, confidence
 - No warming length prediction needed

Works for any microarchitecture if reusable cache & branch predictor state

31

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

32

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

33

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

34

CALCM Computer Architecture Lab CarnegieMellon

Uniprocessor sampling results

	SimpleScalar Complete simulation	SMARTSim Functional warming	TurboSMARTS Live-points
Average (worst) CPI bias	None	0.6% (1.6%)	0.6% (1.6%)
Average benchmark runtime	5.5 days	7.0 hours	91 seconds
Parallel sim. & online results	No	No	Yes
SPEC CPU2000 chpt. Library			12 GB
Fixed microarch. parameters			Max cache, TLB, branch predictors

*Simulation sampling + live-points:
Accurate, faster, enables sampling optimizations*


35

CALCM Computer Architecture Lab CarnegieMellon

Simulation sampling conclusions

- **Live-points** – small, fast loading, reusable checkpoints
- Simulation sampling + checkpoints
 - Results with confidence intervals
 - Huge speedup & parallelism
 - Online results & matched-pair comparison
- TurboSMARTS available at SimFlex website
 - Approaches uni-processor simulation speed limit

36


 **CALCM** Computer Architecture Lab Carnegie Mellon

MP sampling needs checkpoints

- Simulation sampling + checkpoints is essential
 - Full-system simulation
 - 1000× slower than SimpleScalar
 - More complex warming needs
 - Commercial server benchmarks
 - Multiprocessor + OS & I/O intensive
 - Performance often measured with transaction throughput

Flexus: designed-in support for sampling + “flex-points”

37

 **CALCM** Computer Architecture Lab Carnegie Mellon

Tutorial outline

- Introduction
- Simulation sampling
- **Multiprocessor workloads on Flexus**
- Developing with Flexus

Multiprocessor Workloads on Flexus

SimFlex Tutorial – Section 3 of 4

Thomas Wenisch



Computer Architecture Lab at
Carnegie Mellon

18 June 2006, ISCA-33

What makes server apps harder?

- Server application characteristics
 - multi-threaded, multi-processor
 - large memory footprint
 - I/O intensive
 - OS performance matters
 - client-server
 - complicated workload setup & tuning
 - non-deterministic behavior

SPEC CPU has none of these characteristics

CALCM Computer Architecture Lab Carnegie Mellon

Functional simulation challenges

- Functional simulator must support
 - multiple CPUs
 - networks of systems (client-server)
 - privileged-mode ISA
 - peripheral devices
 - more RAM than host system
 - saving/restoring architecturally visible state

Simics provides these capabilities

41

CALCM Computer Architecture Lab Carnegie Mellon

The measurement challenge: Slow full-system simulation

- Simulation slowdown *per cpu*

– Real HW:	~ 500 MIPS	1 s
– Simics:	~ 15 MIPS	33 s
– Flexus, no timing:	~ 1.5 MIPS	5.5 m
– Flexus, in-order:	~ 10 kIPS	13.8 h
– Flexus, OoO:	~ 3 kIPS	46 h

150 years for 1-CPU audited TPC-C run in OoO simulation

42

CALCM Computer Architecture Lab CarnegieMellon

Section 3 outline

- Extending sampling to MP applications
- The SimFlex experiment procedure
- Sampling optimizations

43

CALCM Computer Architecture Lab CarnegieMellon

Our approach in this tutorial

- Develop sampling approach for MP
 - Use SMARTS, Live-points as a starting point
 - Re-examine each step in the MP context
- Primary focus: **throughput applications**
 - E.g., transaction processing, web serving
 - Open problems remain for general MP apps.

44

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

45

CALCM Computer Architecture Lab Carnegie Mellon

SMARTS summary

- SMARTS 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

46

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

47

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

The diagram illustrates the execution of three CPUs (CPU 0, CPU 1, and CPU 2) over time. Each CPU's execution is represented by a horizontal bar divided into segments. Vertical blue lines indicate the arrival of transactions. A 'Transaction' label points to one such vertical line. A 'Sampling unit' label points to a vertical interval between two transaction lines. A 'Time' axis with an arrow points to the right.

48

CALCM Computer Architecture Lab CarnegieMellon

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

Out-of-order model performance

IPC 1 model performance

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

49

CALCM Computer Architecture Lab CarnegieMellon

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

50

CALCM Computer Architecture Lab CarnegieMellon

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

Billions of instructions →

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

51

CALCM Computer Architecture Lab CarnegieMellon

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

52

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

53

CALCM Computer Architecture Lab Carnegie Mellon

Empirical queue warming analysis

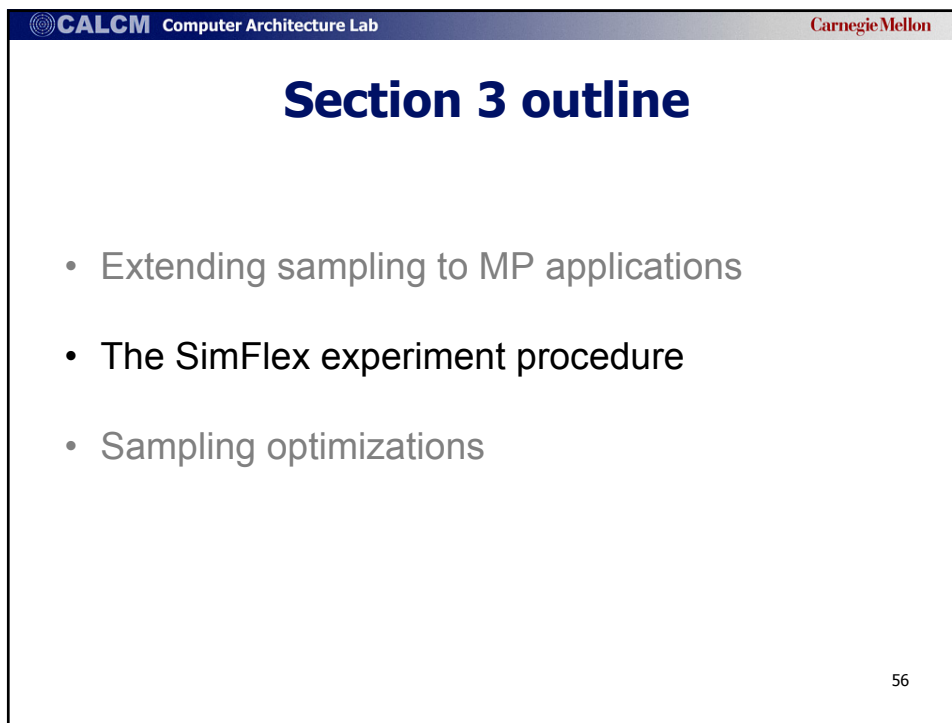
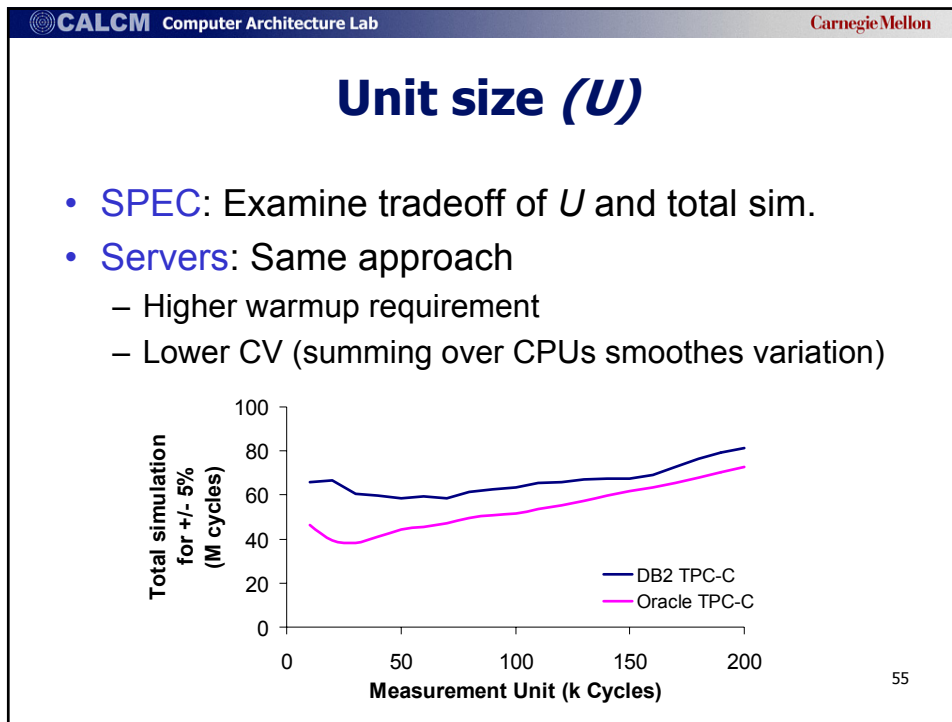
- Bias - relationship btw. perf. and time into msmt.
 - Determine minimum time where bias disappears

Performance vs. Distance into Measurement
(sample size = 50)

User Commits

Distance From Start of Msmt Unit (k cycles)

54



CALCM Computer Architecture Lab Carnegie Mellon

The simplified SimFlex procedure

1. Prepare workload for simulation
 - port workload into Simics
2. Measure baseline variance
 - determine required library size
3. Collect checkpoints
 - via functional warming
4. Detailed simulation
 - estimate performance results

Procedure with sampling optimizations later

57

CALCM Computer Architecture Lab Carnegie Mellon

1. Preparing a workload for simulation

- **SMARTSim**: Compile
- **Flexus**: Bring application up in Simics
 - Install OS, application
 - Construct workload (e.g., load DB)
 - Prepare disk images
 - Tune workload parameters

Ideally, do as much as possible on Real HW

58

CALCM Computer Architecture Lab CarnegieMellon

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

59

CALCM Computer Architecture Lab CarnegieMellon

2. Determining sampling parameters

- **SMARTSim**: Use a preliminary SMARTS run
- **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

60

	SMARTSim (8-way OoO)	Flexus (16-CPU DSMFlex.OoO)
Warming	2000 inst.	100k cycles
Measurement	1000 inst.	50k cycles
Target confidence	99.7% ± 3%	95% ± 5%
Sample size	8000	200-400
Sim. time per checkpoint	10 ms	< 20 min
Experiment turnaround time	91 CPU-sec	~ 30 CPU-hours

61

	SMARTSim	Flexus
Warming	2000 inst.	100k cycles
Measurement	1000 inst.	50k cycles
Target confidence	99.7% ± 3%	95% ± 5%
Sample size	8000	200-400
Sim. time per checkpoint	10 ms	< 20 min
Experiment turnaround time	91 CPU-sec	~ 30 CPU-hours

61

3. Collect checkpoints

- SMARTSim: One run with `sim-mkckpt`
- Flexus: Single run too slow to cover 30s
- Need multi-tier approach optimized to:
 - Leverage speed of Simics “fast” mode
 - Parallelize flex-point creation across CPUs
 - Minimize storage
 - Manage dependence between flex-point files

62

CALCM Computer Architecture Lab Carnegie Mellon

Constructing flex-points

- Use *TraceFlex* or *TraceCMPFlex*
 - High speed (1.5 MIPS)
 - 100 insn / CPU Simics simulation quantum
 - Minimal code paths to collect cache, bpred state
 - Fully deterministic
 - No timing feedback into Simics
 - i.e., execution in Simics as if Flexus not present

63

CALCM Computer Architecture Lab Carnegie Mellon

Simics checkpoints


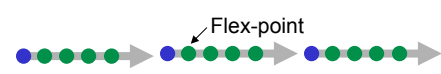
- Simics requires complete arch. state
 - Full checkpoints
 - Proportional to system RAM, ~2GB
 - Independent
 - Delta checkpoints
 - Size proportional to memory updates; 50-300MB
 - Dependant – requires all files in chain
 - Unix file descriptors limit delta chain length (~25)

Storage constraints limit sample sizes to 100's

64

CALCM Computer Architecture Lab CarnegieMellon

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

65

CALCM Computer Architecture Lab CarnegieMellon

4. Detailed simulation

- **SMARTSim**: Process live-points & print results
- **Flexus**: Process all flex-points, aggregate offline
- Manipulate results & stats with `stat-manager`
 - Each run creates binary `stats_db.out` database
 - Offline tools to select subsets; aggregate
 - Generate text reports from simple templates
 - Compute confidence intervals for mean estimates

66

CALCM Computer Architecture Lab Carnegie Mellon

Section 3 outline

- Extending sampling to MP applications
- The SimFlex experiment procedure
- Sampling optimizations

67

CALCM Computer Architecture Lab Carnegie Mellon

Sampling optimizations

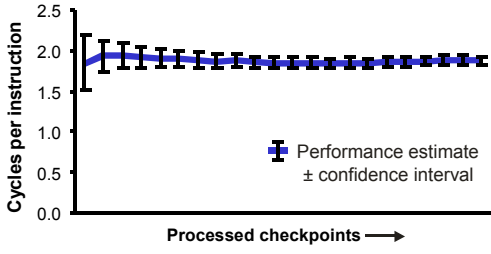
- Online results
 - Report results while simulation in progress
 - Process precise sample size for target confidence
 - Rapid feedback to detect “broken” experiments
- Matched-pair comparison
 - Reduced sample size for comparative experiments
 - Faster turnaround
 - Lower storage requirements

68

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

69

CALCM Computer Architecture Lab Carnegie Mellon

Randomizing a checkpoint library

- SMARTSim: Shuffle live-points in advance
 - Individual live-points ~40KB compressed
 - Sequential I/O improves sim. speed
- Flexus: Shuffle flex-points on use
 - Individual flex-points 10-200MB compressed
 - I/O perf. unaffected by shuffling in advance

70

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

71

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 150×

72

CALCM Computer Architecture Lab **CarnegieMellon**

Matched-pair with Flexus

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

Simics checkpoints●.....●.....●.....→

Flex-points for design A ●●●●●→●●●●●→●●●●●→

Flex-points for design B ●●●●●→●●●●●→●●●●●→

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

73

CALCM Computer Architecture Lab **CarnegieMellon**

The complete SimFlex procedure

1. Prepare workload for simulation
 - port workload into Simics
2. Measure baseline variance
 - determines needed library size
3. Collect checkpoints
 - functional warming used
4. Shuffle live-point library
 - randomize for online results
5. Baseline experiment
 - estimates absolute perf.
6. Matched-pair experiments
 - estimates comparative perf.

74

CALCM Computer Architecture Lab Carnegie Mellon

SimFlex results

	SPEC 2000 (per input)	Commercial 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
Storage for checkpoints	273 MB	30 GB

SimFlex approach makes simulation studies tractable

75

CALCM Computer Architecture Lab Carnegie Mellon

Tutorial outline

- Introduction
- Simulation sampling
- Multiprocessor workloads on Flexus
- **Developing with Flexus**

Developing with Flexus

SimFlex Tutorial – Section 4 of 4

Thomas Wenisch



18 June 2006, ISCA-33

Developing with Flexus

- Flexus philosophy
- Fundamental abstractions
- Important support libraries
- CMU components and what they model

78

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

79

CALCM Computer Architecture Lab Carnegie Mellon

Component-based design

Simulators assembled from composable parts

<ul style="list-style-type: none">• Motivation<ul style="list-style-type: none">– Flexibility– Encapsulation– Model refinement– Parallel development	<ul style="list-style-type: none">• Implementation<ul style="list-style-type: none">– Explicit interface specs– Separate source code
---	---

80

CALCM Computer Architecture Lab Carnegie Mellon

Software-centric framework

Flexus abstractions need not map to HW structure

<ul style="list-style-type: none">• Motivation<ul style="list-style-type: none">– Behavioral modeling not structural– SW-only components e.g., trace generators	<ul style="list-style-type: none">• Implementation<ul style="list-style-type: none">– High-level components “Cache”; not “Sub-array”– Zero-latency connections
--	---

81

CALCM Computer Architecture Lab Carnegie Mellon

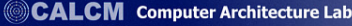

Cycle-driven execution

Components receive “clock-tick” every cycle

<ul style="list-style-type: none">• Motivation<ul style="list-style-type: none">– Component isolation– Cooperating FSMs	<ul style="list-style-type: none">• Implementation<ul style="list-style-type: none">– “Drive” interfaces– Specified clocking order
--	---

- Some components event-driven internally
 - E.g., interconnect component

82

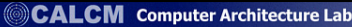

 

SimFlex methodology

Flexus designed to support simulation sampling

- Motivation
 - Measure MP apps
 - Leverage Simics speed
- Implementation
 - Flex-points
 - `stat-manager`

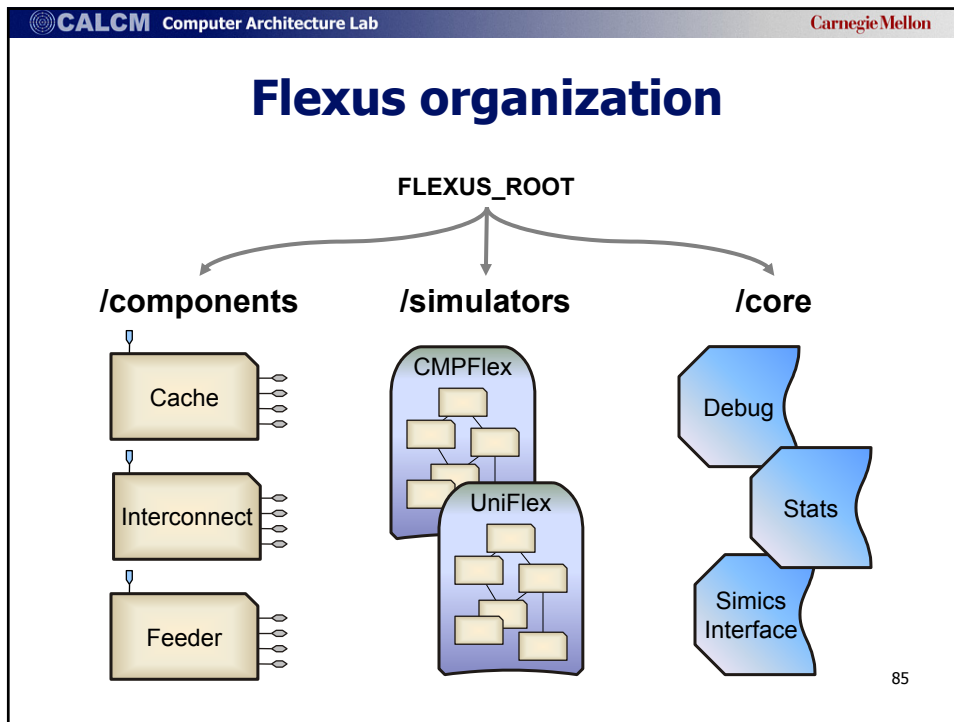
83

Developing with Flexus

- Flexus philosophy
- **Fundamental abstractions**
- Important support libraries
- CMU components and what they model

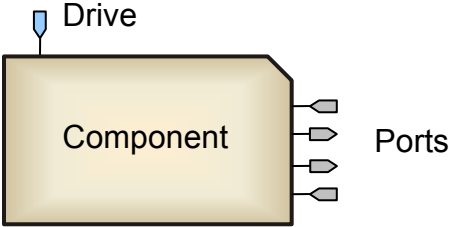
84



-
- The slide is titled **Fundamental abstractions** and lists the following items:
- **Component**
 - Component interface
 - Specifies data and control entry points
 - Component parameters
 - Configuration settings available in Simics or cfg file
 - **Simulator**
 - Wiring
 - Specifies which components and how to connect
 - Specifies default component parameter settings
- The slide includes the CALCM Computer Architecture Lab logo and Carnegie Mellon University name in the top left and right corners, respectively. The number 86 is located in the bottom right corner.

CALCM Computer Architecture Lab Carnegie Mellon

Component interface



The diagram shows a yellow trapezoidal component labeled "Component". A blue pin labeled "Drive" is connected to the top-left corner. On the right side, four grey pins labeled "Ports" are connected to the component.

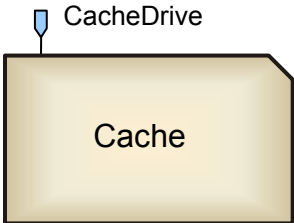
- **Component interface** (terminology inspired by *Asim* [Emer 02])
 - Drive: “clock-tick” control entry point to component
 - Port: specifies data flow between components

Components w/ same ports are interchangeable

87

CALCM Computer Architecture Lab Carnegie Mellon

Abstractions: Drive



The diagram shows a yellow trapezoidal component labeled "Cache". A blue pin labeled "CacheDrive" is connected to the top-left corner.

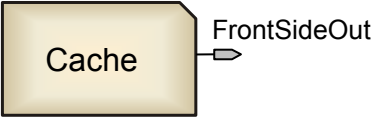
```
COMPONENT_INTERFACE(  
...  
    DRIVE ( Name )  
...  
);
```

- Control entry-point
- Function called once per cycle

88

CALCM Computer Architecture Lab CarnegieMellon

Abstractions: Port



A rectangular box labeled "Cache" with a small arrow-shaped port on its right side labeled "FrontSideOut".

```

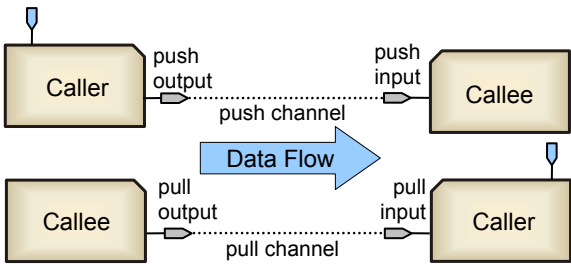
COMPONENT_INTERFACE(
    ...
    PORT ( Type, Payload, Name )
    ...
);
                
```

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

89

CALCM Computer Architecture Lab CarnegieMellon

Types of ports and channels



The diagram shows two scenarios of data flow between a Caller and a Callee. In the top scenario, a 'push channel' connects the 'push output' of the Caller to the 'push input' of the Callee. In the bottom scenario, a 'pull channel' connects the 'pull output' of the Callee to the 'pull input' of the Caller. A large blue arrow labeled 'Data Flow' points from left to right between the two scenarios.

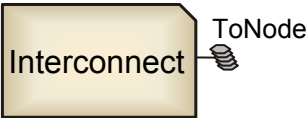
- Type - direction of data and control flow
 - Control flow: Push vs. Pull
 - Data flow: Input vs. Output
- Payload - arbitrary C++ data type
- Type and payload must match to connect ports
- Availability - caller must check if callee is ready

90

CALCM Computer Architecture Lab CarnegieMellon

Port and component arrays

Interconnect



ToNode

```

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

91

CALCM Computer Architecture Lab CarnegieMellon

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) { ... }
                
```

92

CALCM Computer Architecture Lab **CarnegieMellon**

Component-based design & payloads

- Problem: new comps add fields to messages

```

    graph LR
      subgraph Left
        M1[Memory Ctrl] --- NIC1[NIC]
        I1[Interrupt Ctrl] --- NIC1
      end
      subgraph Right
        M2[Memory Ctrl] --- NIC2[NIC]
        I2[Interrupt Ctrl] --- NIC2
      end
      NIC1 --- Network[Network]
      NIC2 --- Network
      Network --- NM[NetworkMessage]
      style NM fill:#ffff00
      NM --- NM_fields["request type  
address  
data  
interrupt #"]
  
```

- Bad solution: modify struct for message
 - Breaks compatibility w/ existing components
 - Leads to conflicts in header files

93

CALCM Computer Architecture Lab **CarnegieMellon**

Transports

```

    graph LR
      NT[NetworkTransport] --- MM[MemoryMessageTag]
      NT --- IM[InterruptMessageTag]
      MM --> MemMsg[MemoryMessage]
      IM --> IntMsg[InterruptMessage]
      style MemMsg fill:#ffff00
      style IntMsg fill:#ffff00
      MemMsg --- MemMsg_fields["request type  
address  
data"]
      IntMsg --- IntMsg_fields["interrupt #"]
  
```

- Scalable data structure
 - Made up of one or more **slices**
 - Transport object holds pointers to slices
 - Reference counting to manage memory
 - Implemented via templates (typesafe; low overhead)
 - Syntax similar to array access

Adding a new slice has no effect on existing code

94

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

95

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

```
graph TD; Feeder --> IFetch; Feeder --> Execute; IFetch --> L1I; Execute --> L1D; L1I --> Mux; L1D --> Mux; Mux --> L2;
```


96

 **CALCM** Computer Architecture Lab Carnegie Mellon

Simulator wiring implementation

- Wiring implemented via C++ overload resolution
 - Push-out, pull-in port accesses call stub functions
 - Stub functions emitted when compiling wiring.cpp
- Advantages
 - Type safety
 - Linker resolves component interconnection
 - Constant time; independent of # wires, # components

97

 **CALCM** Computer Architecture Lab Carnegie Mellon

Developing with Flexus

- Flexus philosophy
- Fundamental abstractions
- **Important support libraries**
- CMU components and what they model

98

CALCM Computer Architecture Lab CarnegieMellon

Critical support libraries in /core

- Statistics support library
 - Record results for use with `stat-manager`

- Debug library
 - Control and view Flexus debug messages

99

CALCM Computer Architecture Lab CarnegieMellon

Statistics support library

- Key features
 - Support tuning of warming, measurement windows
 - Generate text reports from stats database
 - Calculate formulas & confidence intervals
 - Histograms, unique counters, instance counters

- Example:

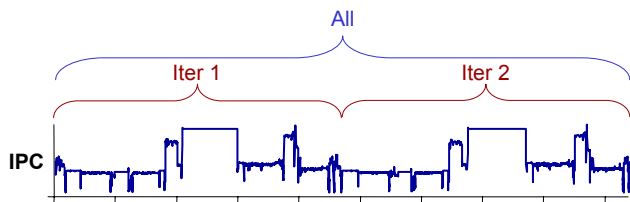
```
Stat::StatCounter myCounter( statName() + "-count" );  
++ myCounter;
```

100

CALCM Computer Architecture Lab Carnegie Mellon

Flexus statistics collection model

- Multiple concurrent stat collection windows



- Can't ask for "current value" of a stat
- Measurement windows controlled via:
 - Cycle, transaction, instruction counts
 - Programmatically in Flexus code
 - Events in the simulated system (MagicBreak component)

101

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}>`

102

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}>`

103

CALCM Computer Architecture Lab Carnegie Mellon

Debug support library

- Filtering at compile time and runtime
 - Via make command line by severity
 - Via simics console by severity, component, category
 - Via `debug.cfg` files by any field
- Formatting, output to multiple destinations
 - Via `debug cfg` files
- Implementation: preprocessor macros

104

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
```

105

CALCM Computer Architecture Lab Carnegie Mellon

Developing with Flexus

- Flexus philosophy
- Fundamental abstractions
- Important support libraries
- **CMU components and what they model**

106

CALCM Computer Architecture Lab CarnegieMellon

Simulators in Flexus 2.1.0

- UniFlex [.OoO] 1 CPU 2-level hierarchy
- CMPFlex [.OoO] private L1 / shared L2
- DSMFlex [.OoO] micro-coded coherence
- TraceFlex Uni or DSM live-points
- TraceCMPFlex CMP live-points

OoO variants support v9, all others support v9/x86

107

CALCM Computer Architecture Lab CarnegieMellon

Memory hierarchy

- “top”, “front” = closer to CPU
- Optimized for high MLP
 - Non-blocking, pipelined accesses
 - Hit-under-miss within set
- Coherence protocol support
 - Valid, modifiable, dirty states
 - Explicit “dirty” token tracks newest value
 - Non-inclusive
 - Supports “Downgrade” and “Invalidate” messages
 - Request and snoop virtual channels for progress guarantees

108

CALCM Computer Architecture Lab Carnegie Mellon

CMP memory hierarchy

- Shared L2 based on Piranha
 - L2 acts as L1 victim cache
 - L2 maintains directory, duplicate tags
 - Both L1I and L1D cache coherent
 - Private L1's behave as in uniprocessor hierarchy

109

CALCM Computer Architecture Lab Carnegie Mellon

Distributed shared memory system

- Micro-coded coherence protocol engines
[contributed by Andreas Nowatzyk]
 - Protocol, engine design, microcode based on Piranha
 - See Piranha publications for overview
- Point-to-point switched interconnect
 - User specifiable interconnect topology, link BW
 - Currently supports any source routed algorithms
 - Deadlock freedom via escape virtual channels

110

CALCM Computer Architecture Lab Carnegie Mellon

In-order execution

- Assumes one cycle for non-memory instructions
- Supports TSO memory model under SPARC
 - Stores retire to store buffer
 - Store buffer drained in order with store prefetching

111

CALCM Computer Architecture Lab Carnegie Mellon

Out-of-order execution

- Timing-first simulation approach [Mauer 2002]
 - OoO components interpret SPARC ISA
 - Flexus validates its results with Simics
- Idealized OoO to maximize memory pressure
 - Decoupled front end
 - Data flow execution (no functional unit constraints)
 - Precise squash & re-execution
 - Configurable ROB, LSQ capacity; dispatch, retire rates
- Memory order speculation (similar to [Gniady 99])

112

CALCM Computer Architecture Lab CarnegieMellon

Reference slides

- Flexus component inventory
- Configuring components
- Debug system details
- Controlling Flexus in code
- Flex-point implementation
- Live-point implementation
- Workload scaling: DBmbench
- Citations
- SimFlex team

113

CALCM Computer Architecture Lab CarnegieMellon

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

114

CALCM Computer Architecture Lab CarnegieMellon

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

115

CALCM Computer Architecture Lab CarnegieMellon

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

116

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"`

117

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

118

CALCM Computer Architecture Lab CarnegieMellon

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

119

CALCM Computer Architecture Lab CarnegieMellon

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

120

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

121

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

122

CALCM Computer Architecture Lab **CarnegieMellon**

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

123

CALCM Computer Architecture Lab **CarnegieMellon**

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*

124

CALCM Computer Architecture Lab CarnegieMellon

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

125

CALCM Computer Architecture Lab CarnegieMellon

DBmbench

- Database μ bmarks 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

126

CALCM Computer Architecture Lab CarnegieMellon

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

127

CALCM Computer Architecture Lab CarnegieMellon

SimFlex developers

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



Flexus developers
SMARTS research

128