Advanced Computer Architecture

Evaluating Systems

Fall 2016

Pejman Lotfi-Kamran

Adapted from slides originally developed by Profs. Hill, Hoe, Falsafi and Wenisch of CMU, EPFL, Michigan, Wisconsin

Where Are We?

Fr	Sa	Su	Mo	Tu
	27-Shahrivar		29-Shahrivar	
	3-Mehr		5-Mehr	
	10-Mehr		12-Mehr	
	17-Mehr		19-Mehr	
	24-Mehr		26-Mehr	
	1-Aban		3-Aban	
	8-Aban		10-Aban	
	15-Aban		17-Aban	
	22-Aban		24-Aban	
	29-Aban		1-Azar	
	6-Azar		8-Azar	
	13-Azar		15-Azar	
	20-Azar		22-Azar	
	27-Azar		29-Azar	
	4-Dey		6-Dey	

- This Lecture
  - Evaluation
- Next Lecture:
  - Evaluation

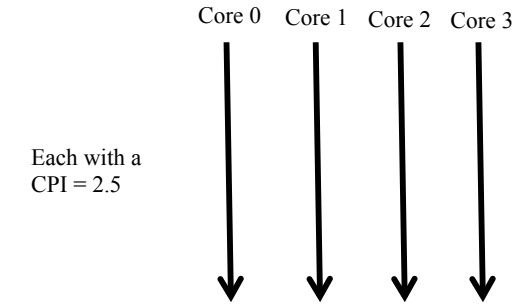
Roadmap

- ➔Metrics
- Tools
- Workloads

Performance Metric

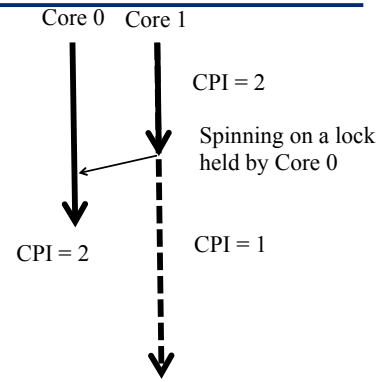
- *Uniprocessor*: Cycles per instruction (CPI)
- *Multiprocessor*: CPI is not proportional to perf.
  - Threads often synchronize through spinning
  - Waiting for others to catch up
  - Or allowing access to shared data one at a time
- Spinning threads have low CPI!!!!
  - Averaging across threads would lead to misleading results
- In the limit, if all threads are waiting all the time
  - Average CPI is really low
  - But, there is no forward progress (the program is running sequentially)
  - Not all instructions make forward progress
  - Frequent spins on I/O and locks

Example: CPI not useful for Parallel Programs



Average CPI?
Program CPI?

Example: CPI not useful for Parallel Programs



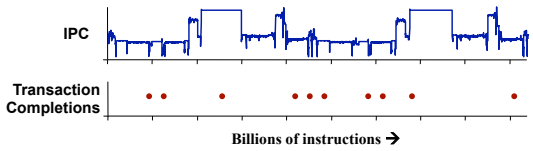
Average CPI?
Program CPI?

CPI not Useful in General

- Bottomline is execution time
- If spinning can be counted out, then CPI works
- Transactional server workloads
- Mostly spin in the OS
- User-level CPI is valid
- Can be averaged

MP performance metric

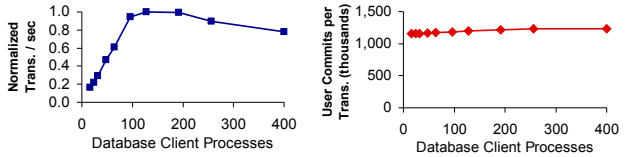
- Typical metric: Transactions completed / min. (TPM)
- Coarse progress metrics bad for sampling
  - Transaction completions infrequent
  - High variance of inter-arrival and service times



Measuring TPM requires large measurements

Fine-grain performance metric

- User-mode inst. per trans. constant per app. cfg.
- Hence, user-mode IPC ∝ 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

Other Metrics: Cost

- Cost is a function of more than just the processor.
- Cost is a complex function of many hardware components and software
- Cost is often not a "smooth" function
  - Often a function of packaging
    - how many pins on a board
    - how many processors on a board
    - how many boards in a chassis
  - Typically refers to design cost
- There is also
  - Manufacturing cost
  - Operation cost (becoming huge)

Operation Cost: Enterprise IT (Server) Energy Usage

Kenneth Brill (Uptime Institute)

- "Economic Meltdown of Moore's Law"
- In 2012: Energy/server lifetime 50% more than price/server
  - And 2% of all Carbon footprint in the US

Energy Star report to Congress:

- Datacenter energy 2x from 2000 to 2006
- Roughly 2% of all electricity & growing

TCO (Total Cost of Operation) is the metric!

Other Metrics: Reliability

Until today

- mostly dealt with at the circuit/technology level
- mostly a specialized market
  - if high reliability is important (e.g., ATM transactions)
    - Deal with it in HW
    - Large redundant mainframes for banks
    - Multiple redundant/heterogeneous computers in planes
  - Otherwise tolerate faults and/or handle in software
- in the future, technologies will be error-prone
  - May have to deal with reliability in all computing market segments
    - Embedded, handheld, desktop, server, supercomputer
    - Specialize hardware to protect only parts needed
- Related to cost (design + energy)

Roadmap

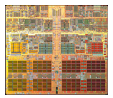
- Metrics
- ➔Tools
- Workloads

How do we evaluate?

- Model it analytically (analytic models)
  - Fast
  - Not accurate
- Model it in software (simulator)
  - REAL SLOW
  - More accurate
- Model it in FPGA
  - Fast
  - Limited in model size
  - Slow to program
- Hardware counters
  - Fast
  - Not flexible!

Simulation Speed Challenges

- Longer benchmarks
  - SPEC 2006: Trillions of instructions per benchmark
- Slower simulators
  - Full-system simulation: 1000× slower than SimpleScalar



- Multiprocessor systems
  - CMP: 2x cores every processor generation

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

Full-system simulation is slow

- Simulation slowdown per cpu
  - Real HW: ~ 2 GIPS 1 s
  - Simics: ~ 30 MIPS 66 s
  - Flexus, no timing: ~ 900 KIPS 37 m
  - Flexus, OoO: ~ 24 KIPS 23 h

2 years to simulate 10 seconds of a 64-core workload!

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 μarch state prior to measurements
  - Allow independent simulation of each measurement
  - Sample non-deterministic, highly variable MP applications

Functional Simulation

- Functional simulation is faster than detailed simulation
  - Flexus (no timing) is 38 times faster than Flexus (OoO)
- Use functional simulation for "warmup"
  - Memory (guarantees correctness)
  - Cache hierarchy (avoids bias)
  - Branch predictor (avoids bias)

No state for core microarchitecture ➔ Bias

Handling Bias

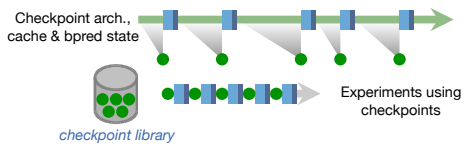
- Core micro-architecture can be warmed up rapidly
  - Detailed simulation to warmup core micro-architecture
- Perform warmup prior to measurement
  - Functional warming during fast-forwarding
  - Detailed warmup before each simulation window

Simulation Speedup

- 10 seconds of a 64-core workload
  - Normal execution: 2 years
  - With sampling: 20 days

- 37x improvement in speed but not enough
- Solution
  - Avoid functional simulation (which lasts 17 days)
  - Accelerate detailed simulation (which lasts 3 days)

Avoiding Functional Simulation

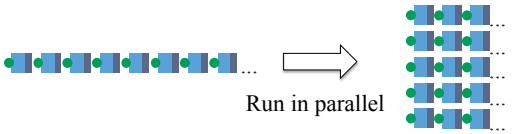


- Store warm cache & branch predictor state
  - Same sample design, accuracy, confidence
  - No warming length prediction needed

Works for any microarchitecture

Accelerating Detailed Simulation

- Checkpoint library makes measurement independent
- Run multiple measurements in parallel

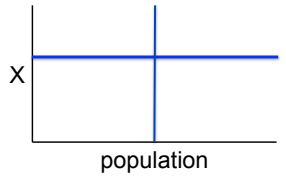


Simulation Speedup

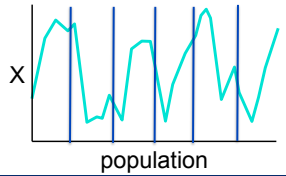
- Sampling without a checkpoint library:
  - 10 seconds of a 64-core workload: 20 days

- Sampling with a checkpoint library:
  - 10 seconds of a 64-core workload: 3 hours with 100 cores

How to Choose the Sample Size?



Low variability ➔ Small sample size

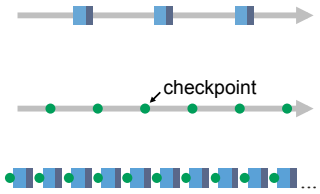


High variability ➔ Large sample size

Variability determines sample size

Steps for Timing Simulation

1. Prepare workload for simulation
   - Port workload into Simics
2. Measure baseline variance
   - Determine required library size
3. Collect checkpoints
   - Via functional warmup
4. Detailed Simulation
   - Estimate performance results



2. Determine Sampling Parameters

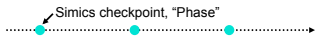
- Guess variability
- Generate flexpoints for the variability
- Run timing simulation
- Measure error and correct the guess

Typical Sampling Parameters

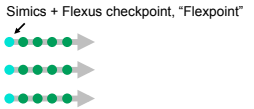
	Flexus (64-CPU CMP.OoO)
Warming	100k cycles
Measurement	50k cycles
Target confidence	95%
Sample size	800
Sim. time per checkpoint	~ 20 min

3. Checkpoint Creation

- Spread Simics checkpoints
  - Simics fast mode rapidly covers 10 seconds



- Collect flexpoints in parallel
  - Via CMP.L2Shared.Trace
  - From each Simics checkpoint



4. Detailed Simulation

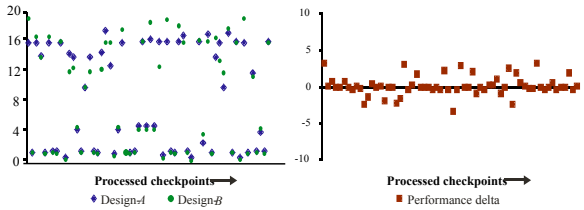
- Run detailed simulation with OoO simulators
- Process all flexpoints, aggregate offline
- Manipulate results with *stat-manager*
  - Each run creates binary `stats_db.out` database
  - Offline tools to select subsets; aggregate
  - Generate text reports from simple templates
  - Compute confidence intervals for mean estimates

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

Matched-pair example

Performance results for two microarchitecture designs
checkpoints processed in random order



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

Matched-pair with Flexus

- Simple μArch changes (e.g., changing latencies)
  - use same flex-points
- Complex changes (e.g., adding components)