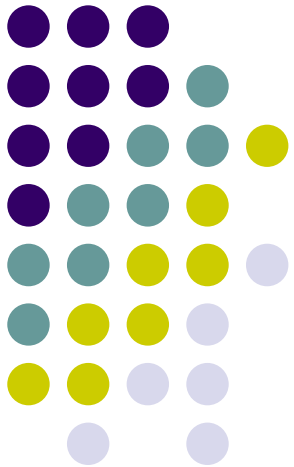
Transforming Computer System Design

Derek Chiou

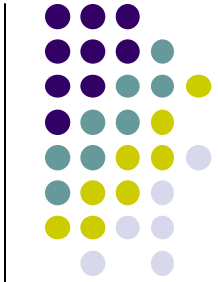
University of Texas at Austin

Electrical and Computer Engineering



Supported in part by DOE, NSF, SRC, Bluespec, Intel, Xilinx, IBM, and Freescale

First, Some Terminology



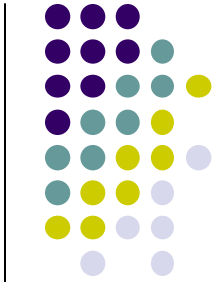
- Host
  - The system on which a simulator runs
  - E.g.,
    - Dell 390 with a single 1.8GHz Core 2 Duo, 4GB of RAM, 10K RPM Seagate HD
    - A Xilinx XUP board
- Target
  - The system being modeled
  - Eg.,
    - Alpha 21264 processor
    - Dell 390 with a single 1.8GHz Core 2 Duo, 4GB of RAM, 10K RPM Seagate HD

Simulation

Many large computers used to simulate physical world
Simulators (generally) get faster as computers get faster
Real world does not increase in complexity

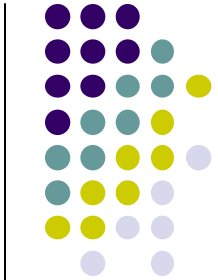
Such simulators are fantastically capable, and getting better, quickly

Except For Simulating Computers



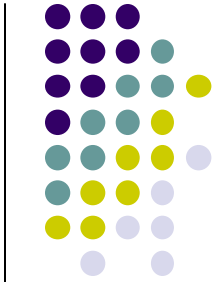
- Accurate simulators are slow
  - x86 1KIPS-10KIPS (Intel/AMD)
    - 3GIPS processor for 1 sec take 83 hours at 10KIPS
      - Simulating 2 minutes takes over 1 year
- Getting slower
  - Unlike physical world, computers grow in complexity faster than they get faster
    - More complex cores, more cores, more features, etc.
  - Slowing down by a factor of 2, relative to target, per year (Murkherjee, Intel)

Why Faster Simulators?



- Why not run many short simulations in parallel?
  - Cannot run full, unmodified, unbenchmarked, software
- Better architect computer systems ***before*** they are built
  - Speed enables longer , heavier evaluation of software on future hardware
    - what architectural mechanisms improve database/game speeds?
- Facilitate ***during*** co-design of hardware and software
  - E.g., Intel and Microsoft making a system that works well together
    - Need to be able to execute sufficient cycles to run Microsoft software
  - Reduce/eliminate work by transforming simulator to implementation
- Tune software for correctness, performance, and power ***after*** real system exists
  - Provide full visibility at useable speeds
  - Difficult/impossible to achieve on a real system
- ***Requires Unified Simulator for Architects, Designers, Software, Algorithms***

Benefits of a Unified Simulator

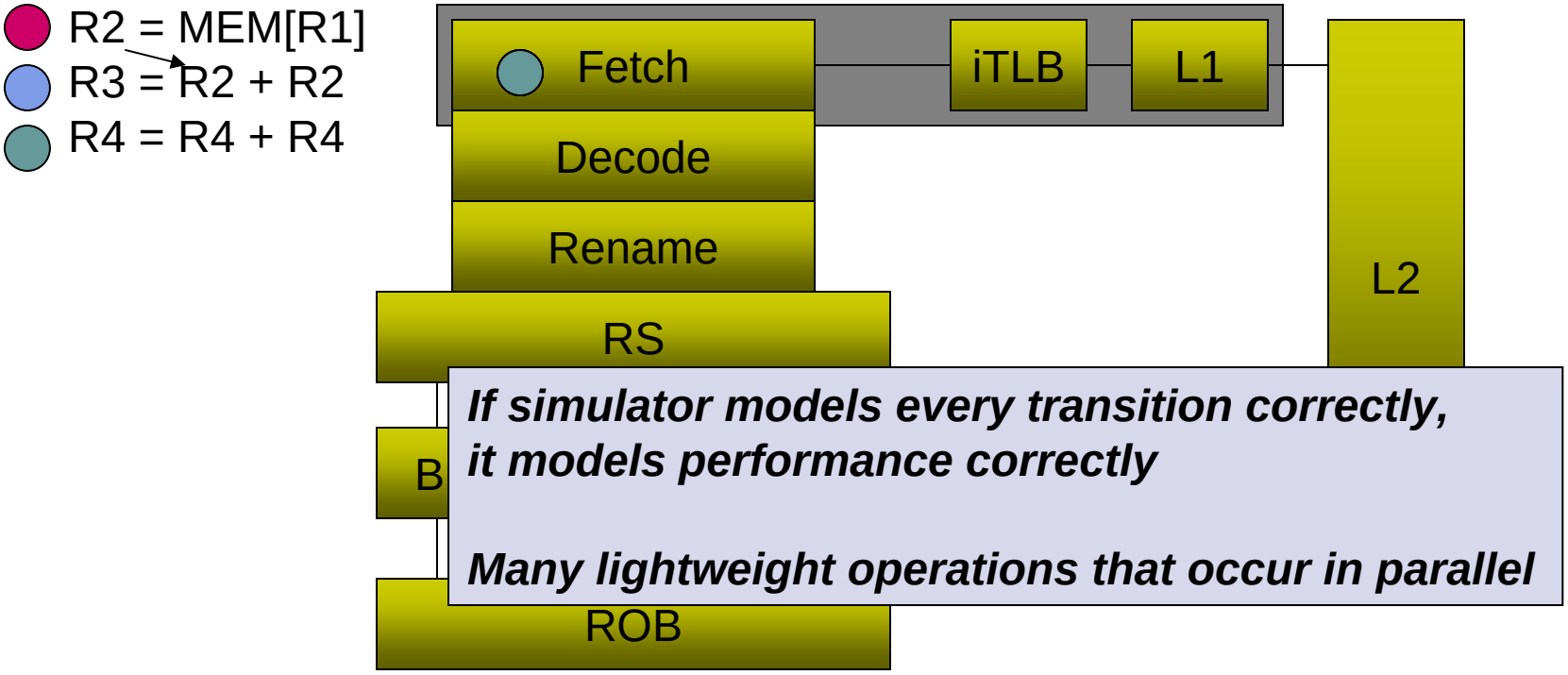


- A common sandbox for Arch/RTL/Software
  - ***Promotes sharing, information transferred immediately***
  - Write/verify once, eliminating ambiguity, wasted work, effort to keep simulators consistent
  - Work proceeds in parallel
    - ***Architecture/software/algorithms co-developed***
- Simulator runs software at interactive speeds while predicting performance, power, temperature, etc.
- Generate implementation from simulator???

But, a Unified Simulator has Unified Requirements

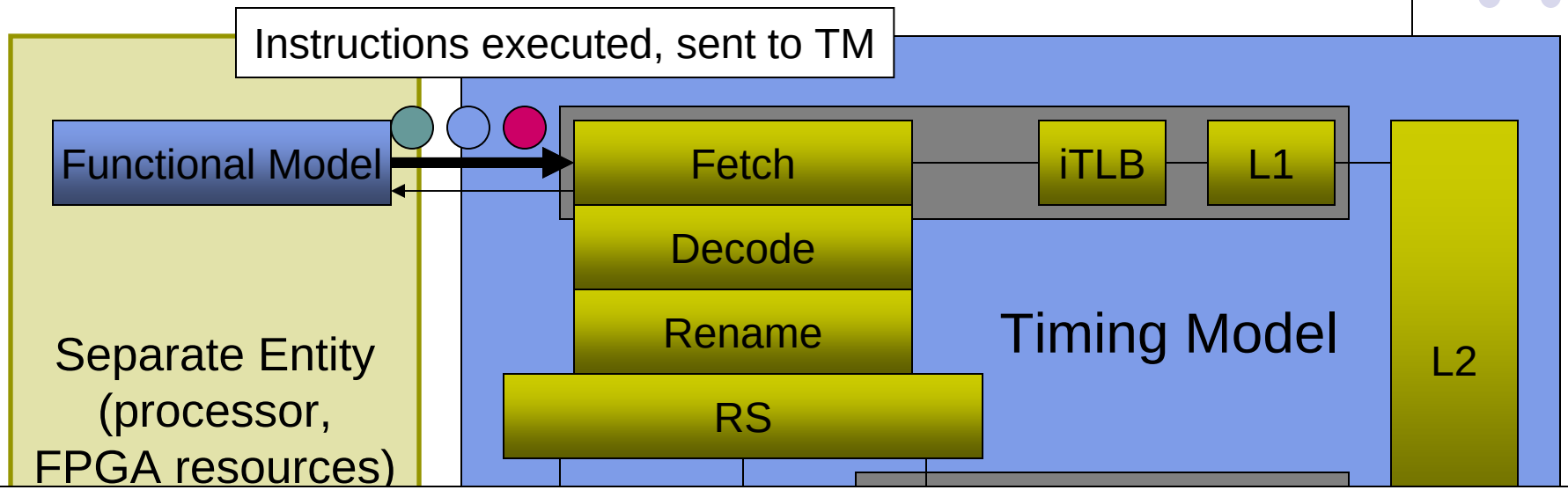
- Architecture
  - **Timely**: enough time to make decisions
  - **Accurate:** compare architectural mechanisms
  - **Flexible**: quick changes
  - **Transparent**: full visibility with little/no performance impact
  - **Power**: compare architectural mechanisms
- Software
  - **Fast**: (~10MIPS+/host in accurate mode, 20-100MIPS+ /host nearly accurate)
  - **Full-system**: run unmodified operating systems, applications,…
  - **Bottleneck Detection:** automatically find problems
- Implementation
  - **Accurate**: produce cycle-accurate numbers
  - **Synthesizable**: convert to implementation
  - **Reducible**: convert back to a simulator

Cycle-Accurate Simulators



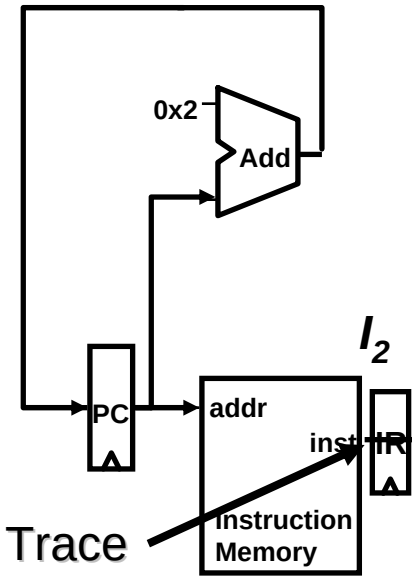
FPGA-Accelerated Simulation Technologies (FAST) Simulator Architecture

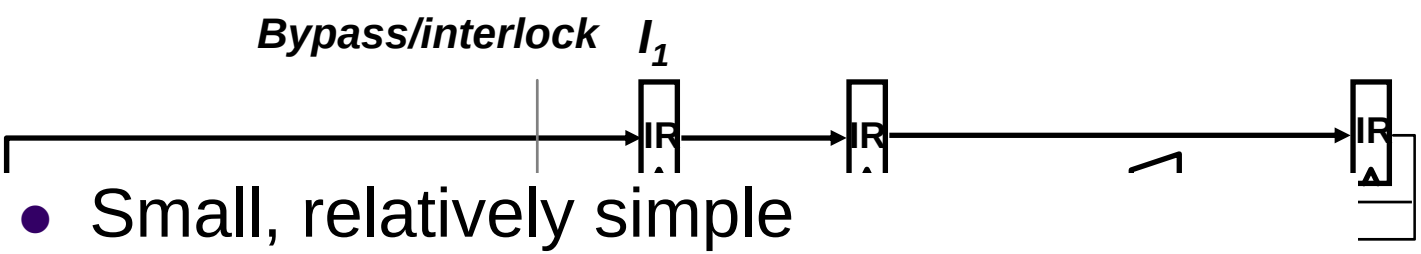
MICRO 2007



- Functional model speculates at ***simulator level***
  - Roll back required if functional path != timing path
    - Branch misprediction likely to require 2 rollbacks
  - Functional model runs ahead, improves parallelism
- Parallel slowdown due to communication?
  - FM runs ahead, speculatively, round-trip communication infrequent
  - **Round-trip communication only when functional path!=timing path**
- The better the target micro-architecture, the faster the simulator

Timing Models are Simple Compared to Implementation

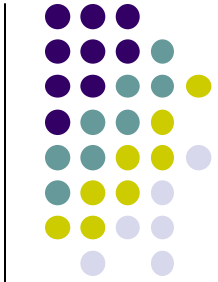




I_2

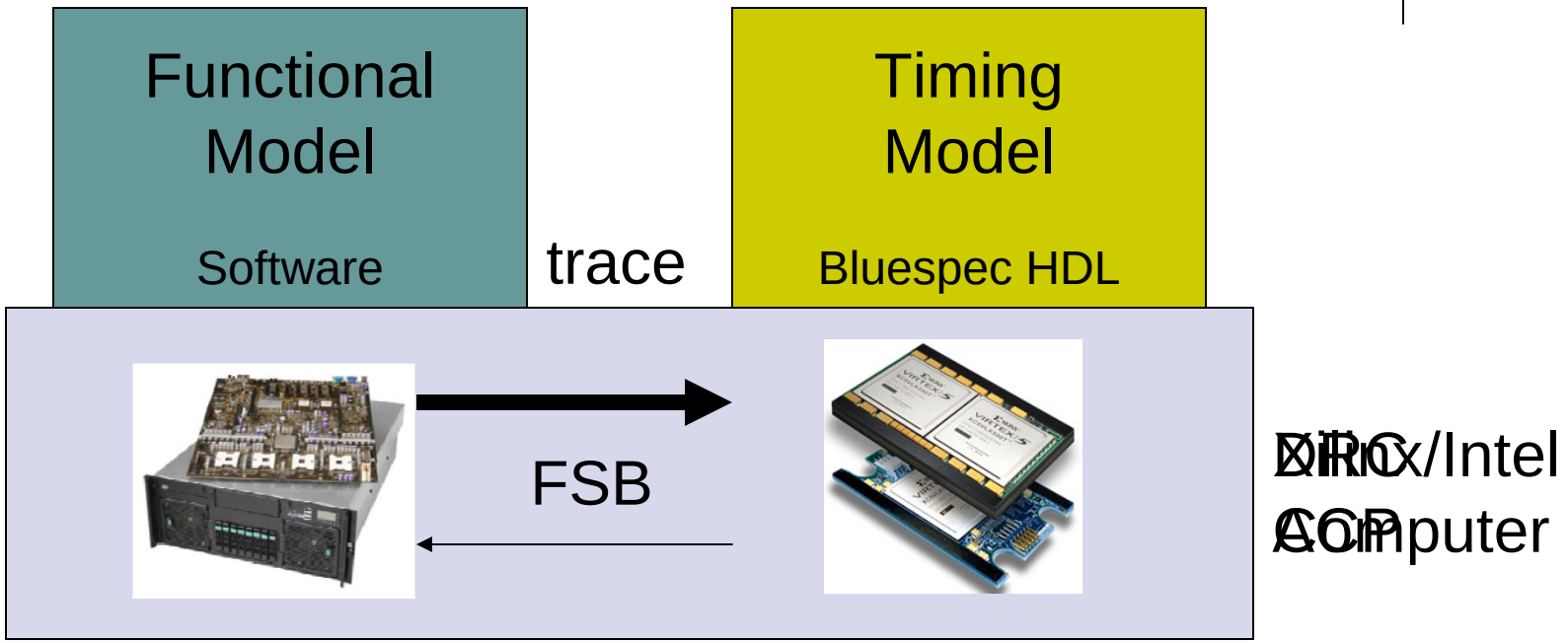
- Small, relatively simple
  - Eventually as simple as software timing model?
- Easy to modularize, compose from libraries
  - makes hardware easier

Why are FAST Timing Models Simpler than Implementation?



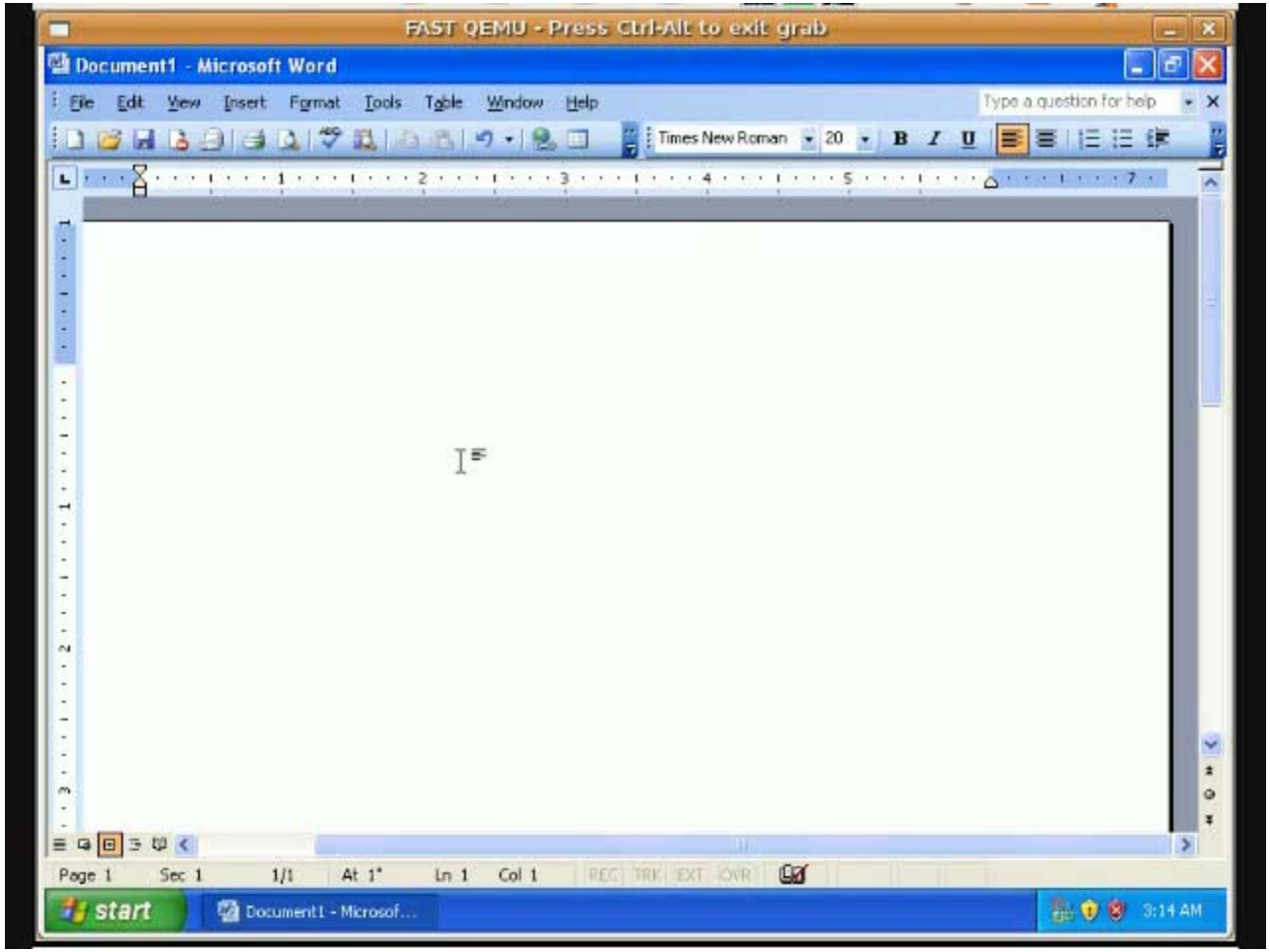
- ***Basically no functionality***
- Multiple host cycles per target cycle
  - Can write a loop for a CAM
- Not trying to meet timing
  - Multiple host cycles helps with that
- Many modules are just delays
  - Fixed cycle ALUs
- Can approximate when desired
  - Cache studies (have all addresses) without detailed core
- Often derived from previous micro-architecture

FAST Prototype Overview



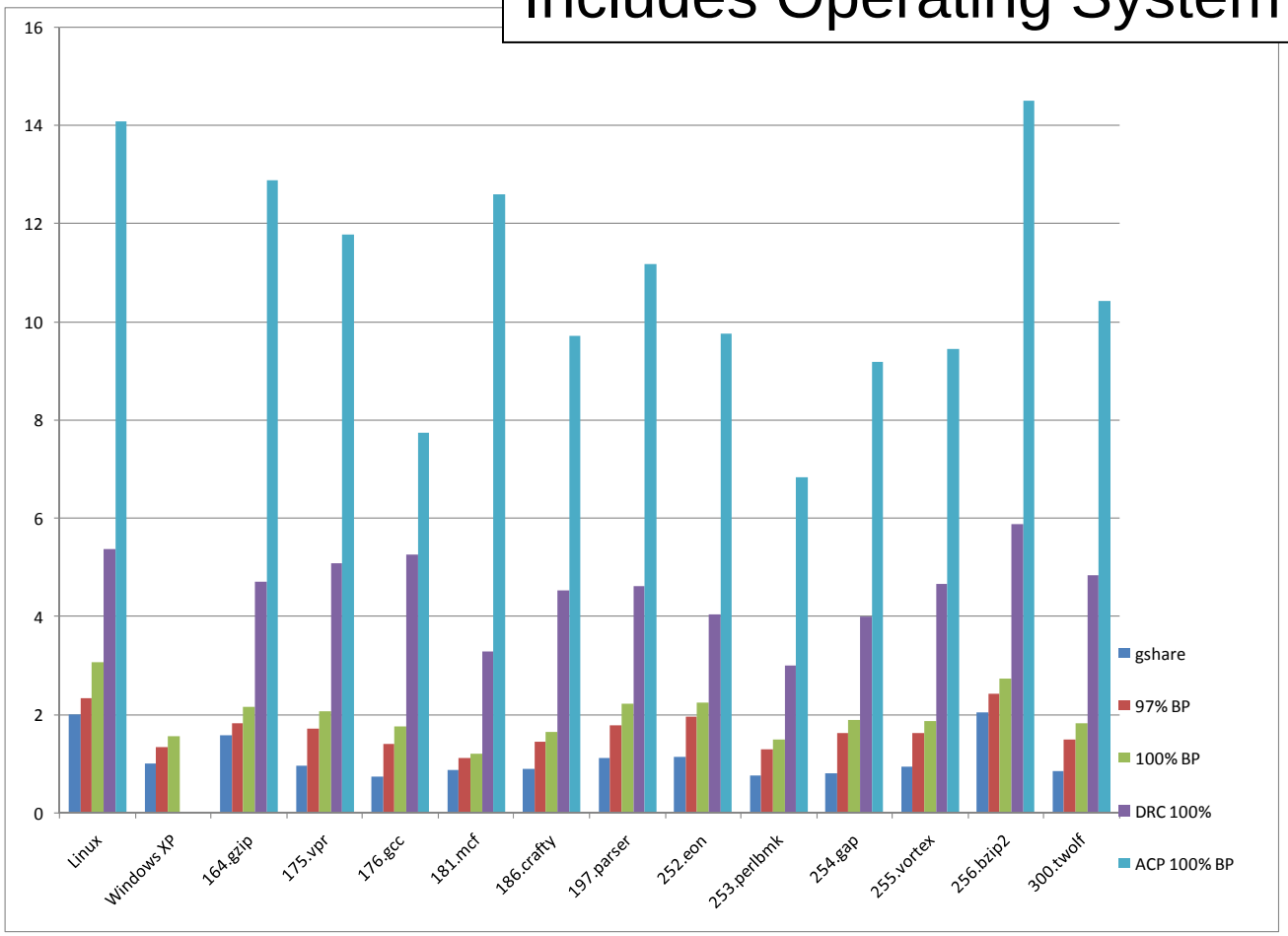
- Software (QEMU modified with trace/rollback) functional model
  - Eventually hardware functional model, but software sim exists
- FPGA-based timing model written in Bluespec
  - Complex OoO micro-architecture fits in a single FPGA

Old FAST on DRC Platform (~1.2MIPS, 1000x Intel/AMD)



Current Simulator Performance

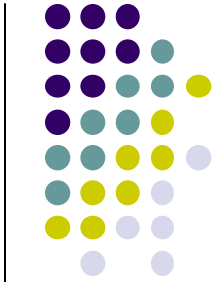
Includes Operating System Code



Are Simulation/Implementation Two Forms of Same Thing?

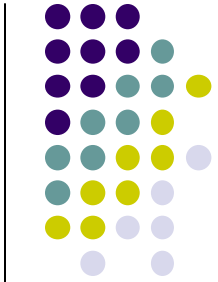
- Both can
  - Run entire software stack
  - Accurately predict performance
- Analogous to frequency vs time domain in DSP?
  - Easier to filter in freq domain than in time domain
  - Transform & operate can make things easier
- Requires “transforms” to convert back and forth
  - High speed simulation in simulator domain
  - Verify in implementation domain

FAST Capability Status



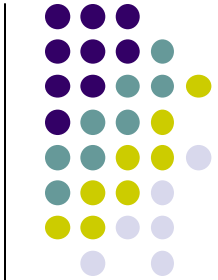
- **Architect**
  - **Timely**: enough time to make decisions
  - **Accurate:** compare architectural mechanisms
  - **Flexible**: quick changes
  - **Transparent**: full visibility with little/no performance impact
  - **Performance:** prediction
  - **Power**: prediction
- Software
  - **Fast** (~10MIPS/host, 20-50-100MIPS+ /host nearly accurate)
  - **Full-system**: run unmodified operating systems, applications,…
  - **Bottleneck Detection:** automatically find performance problems
- Implementation
  - **Accurate**: produce cycle-accurate numbers
  - **Synthesizable**: convert to implementation
  - **Reducible**: convert back to a simulator

Acknowledgements



- Students
  - Dam Sunwoo (FM)
  - Nikhil Patil (TM, tools, FAST2Imp)
  - Bill Reinhart (TM connector)
  - Eric Johnson (TM, machine learning + arch)
  - Joonsoo Kim (TM, Implementation2FAST)
  - Gene Wu (FM)
- Funding, Equipment
  - DOE, NSF, SRC, Sandia Graduate Fellowship
  - Intel, IBM, Xilinx, Freescale
- Software, tools
  - Bluespec, Xilinx
- Open-source full system simulators
  - QEMU, Bochs

Conclusions



- Accurately simulates next generation on current generation at roughly current generation speeds
  - Within a factor of 100 or so today
  - Could be within a factor of 2 to 10 in future
    - Timing model speed is current limit
  - Lower accuracy faster
- Supports integrated software and hardware exploration
- Easy to do “what-if” experiments
  - Map target software functions to arbitrary timing
    - Arbitrary sized matrix multiply in 1 target cycle
- Software performance/power tuning tool
- Simulator scaling can be purchased
  - More host systems communicating over standard host interconnect